LOW-POWER PROCESSORS AND SYSTEMS ON CHIPS

Christian Piguet
CSEM
Neuchâtel, Switzerland

Taylor & Francis
Taylor & Francis Group
Boca Raton London New York

A CRC title, part of the Taylor & Francis imprint, a member of the Taylor & Francis Group, the academic division of T&F Informa plc.

Published in 2006 by
CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-10: 0-8493-6700-X (Hardcover)
International Standard Book Number-13: 978-0-8493-6700-7 (Hardcover)
Library of Congress Card Number 2005050175

Library of Congress Cataloging-in-Publication Data

Piguet, Christian.
Low-power processors and systems on chips / Christian Piguet.
p. cm.
Includes bibliographical references and index.
ISBN 0-8493-6700-X (alk. paper)
1.Microprocessors – Power supply. 2. Systems on a chip. 3. Low voltage integrated circuits. I. Title.

TK7895.M5P54 2005
621.39'16—dc22 2005050175

T&F informa
Taylor & Francis Group
is the Academic Division of T&F Informa plc.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Preface

Purpose and Background

The present book is a part of the book "Low-Power Electronics Design," edited by Christian Piguet, published in November 2004. It contains only the chapters that describe the design of low-power processors and systems-on-chips from microprocessors, DSP cores, reconfigurable processors, memories, systems-on-chip issues, applications such as ad hoc networks and finally embedded software. All the other chapters, describing microelectronics technologies, transistor models, logic circuits and CAD tools, are also included in another smaller book entitled "Low-Power CMOS Circuits: Technology, Logic Design and CAD Tools."

The goal of the present book "Low-Power Processors and Systems on Chips" is to cover all the aspects of the design of low-power microprocessors in deep submicron technologies. Today, the power consumption of microprocessors is considered as one of the most important problems for high-performance chips as well as for portable devices. For the latter, it is due to the limited cell battery lifetime, while it is the chip cooling for the first case. As a result, for any chip design, power consumption has to be taken into account very seriously. Before 1993–1994, only speed and silicon area were important in the design of integrated circuits, and power consumption was not an issue. Just after, it was recognized that power consumption has to be taken into account as a main design parameter. Many papers and books were written to describe all the first design methodologies to save power limited to circuit design. However, today, we have to cope with many new problems implied by very deep submicron technologies, such as leakage power, interconnect delays and robustness.

Today, we are close to designing one billion transistor microprocessor chips, down to 0.10 μm and below, supplied at less than half a volt and working at some GHz. This is due to an unexpected evolution of the microelectronics technologies and to very innovative microprocessor architectures. This evolution is not yet at its end, so the next decade will also see some spectacular improvements in the design of microprocessor circuits. However, it is sure that the microprocessor architecture evolution is not always a revolution, but as pointed out by:

> "I was greatly amused few years ago — when companies were introducing pipelined microprocessors — to learn that RISC technology enabled pipelining. That this could be responsible for pipelining, which has existed for more than 30 years, illustrates the amnesia present in computer engineering"
>
> **Michael J. Flynn**

Organization

The first part of the proposed book starts with a chapter about the design of low-power microprocessors regarding the technology variations. The next three chapters present the design of Digital Signal Proces-

sors (DSP) for embedded applications. They have to provide huge power computation as well as very small power consumption. So many different DSP architectures have been proposed, well adapted to some specific DSP algorithms, working in cooperation with hardware accelerators or based on reconfigurable hardware. Asynchronous design for microprocessors is also proposed to reduce power consumption. In wireless communication, low-power baseband processors are a key issue for portable devices. However, a significant part of the power consumption is due to program and data memories, and the last three chapters of this first part present some techniques to reduce dynamic and static power at the electrical level as well as at the system level while using cache memories or specific memory organization.

The second part of the book is a set of chapters describing several aspects of low-power systems on chips (SoCs). They include hardware and embedded software aspects, such as operating systems (OS), data storage in an efficient way and networks on chips. The next chapters present some applications requiring very low power SoCs, such as ad hoc networks with very low-power radios as well as routing strategies and sensing and actuation devices.

The third part of the book presents issues about embedded software, i.e., application software and compilers. The development tools including compilers, retargetable compilers, and coverification tools are presented in details.

The key benefits for readers will be this complete picture of what is done today for reducing power for microprocessors, DSP cores, memories, systems on chips, and embedded software.

Locating Your Topic

Several avenues are available to access desired information. A complete table of contents is presented at the front of the book. Each of the chapter is also preceded with an individual table of contents. Each contributed chapter contains comprehensive references including books, journal and magazine papers, and sometimes Web pointers.

Acknowledgments

The value of this book is completely based on the many excellent contributions of experts. I am very grateful to them, as they spent a lot of time writing excellent texts without any compensation. Their sole motivation was to provide readers excellent contributions. I would like to thank all these authors, as I am sure this book will be a very good text for many readers and students interested in low-power design. I am indebted to Prof. Vojin G. Oklobjzija for asking me to edit this book and trusting me with this project. I would also like to thank Nora Konopka and Allison Taub of CRC Press for their excellent work in putting all this material in the present form. It is the work of all that made this book.

The Editor

Christian Piguet was born in Nyon, Switzerland, on January 18, 1951. He received the M. S. and Ph. D. degrees in Electrical Engineering from the Ecole Polytechnique Fédérale de Lausanne, Switzerland in 1974 and 1981, respectively.

He joined the Centre Electronique Horloger S.A., Neuchâtel, Switzerland, in 1974. He worked on CMOS digital integrated circuits for the watch industry, on low-power embedded microprocessors as well as on CAD tools based on a gate matrix approach. He is now Head of the Ultra-Low-Power Sector at the CSEM Centre Suisse d'Electronique et de Microtechnique S.A., Neuchâtel, Switzerland. He is presently involved in the design and management of low-power and high-speed integrated circuits in CMOS technology. His main interests include the design of very low-power microprocessors and DSPs, low-power standard cell libraries, gated clock and low-power techniques as well as asynchronous design.

He is Professor at the Ecole Polytechnique Fédérale Lausanne (EPFL), Switzerland, and also lectures in VLSI and microprocessor design at the University of Neuchâtel, Switzerland and in the ALaRI master program at the University of Lugano, Switzerland. He is also a lecturer for many postgraduates courses in low-power design.

Christian Piguet holds about 30 patents in digital design, microprocessors and watch systems. He is author and coauthor of more than 170 publications in technical journals and of books and book chapters in low-power digital design. He has served as reviewer for many technical journals. He also served as Guest Editor for the July 1996 JSSC Issue. He is a member of steering and program committees of numerous conferences and has served as Program Chairman of PATMOS'95 in Oldenburg, Germany, co-chairman at FTFC'99 in Paris, Chairman of the ACiD'2001 Workshop in Neuchâtel, Co-Chair of VLSI-SOC 2001 in Montpellier and Co-Chair of ISLPED 2002 in Monterey. He was Chairman of the PATMOS executive committee during 2002. He was Low-Power Topic Chair at DATE 2004 and 2005.

Christian Piguet, CSEM SA, Jaquet-Droz 1, 2000 Neuchâtel, Switzerland
Christian.piguet@csem.ch

Contents

Contributors

Claude Arm
CSEM
Neuchâtel, Switzerland

Luca Benini
University of Bologna
Bologna, Italy

Youcef Bouchebaba
University of Nantes
Nantes, France

Aimen Bouchhima
TIMA Laboratory
Grenoble, France

Erik Brockmeyer
IMEC
Leuven, Belgium

Francky Catthoor
IMEC
Leuven, Belgium
and
Katholiek University
Leuven, Belgium

Wander Cesario
TIMA Laboratory
Grenoble, France

Yuen Hui Chee
University of California–Berkeley
Berkeley, California

Lawrence T. Clark
Arizona State University
Tempe, Arizona

Fabien Coelho
Ecole des Mines
Paris, France

Stefan Cserveny
CSEM
Neuchâtel, Switzerland

Raphaël David
ENSSAT/University of Rennes
Lannion, France

Vivek De
Intel Labs
Santa Clara, California

Peter Dytrych
Philips Digital Systems Laboratories
Leuven, Belgium

Laurent Fesquet
TIMA Laboratory
Grenoble, France

Joao Fragoso
TIMA Laboratory
Grenoble, France

Simone Gambini
Universita di Pisa
Pisa, Italy

Lovic Gauthier
FLEETS
Fukuoka, Japan

Catherine H. Gebotys
University of Waterloo
Waterloo, Ontario, Canada

Cedric Ghez
IMEC
Leuven, Belgium

Gert Goossens
Target Compilers Technologies
Leuven, Belgium

Koji Inoue
Fukuoka University
Fukuoka, Japan

Ahmed A. Jerraya
TIMA Laboratory
Grenoble, France

Pradeep K. Khosla
Carnegie Mellon University
Pittsburgh, Pennsylvania

Ulrich Kremer
Rutgers University
Piscataway, New Jersey

Chidamber Kulkarni
University of California–Berkeley
Berkeley, California

Dirk Lanneer
Philips Digital Systems Laboratories
Leuven, Belgium

Dake Liu
Department of Electrical Engineering
Linköping University
Linköping, Sweden

Richard Lu
University of California–Berkeley
Berkeley, California

Alberto Macii
Politecnico di Torino
Torino, Italy

Morteza Maleki
University of Southern California
Los Angeles, California

Diana Marculescu
Carnegi Mellon University
Pittsburgh, Pennsylvania

Radu Marculescu
Carnegie Mellon University
Pittsburgh, Pennsylvania

Jean-Marc Masgonty
CSEM
Neuchâtel, Switzerland

Tycho van Meeuwen
IMEC
Leuven, Belgium

Giovanni de Micheli
Stanford University
Stanford, California

Miguel Miranda
IMEC
Leuven, Belgium

Vasily G. Moshnyaga
Fukuoka University
Fukuoka, Japan

Thierry J.-F. Omnès
Philips Semiconductors
Eindhoven, The Netherlands

Brian P. Otis
University of California–Berkeley
Berkeley, California

Massoud Pedram
University of Southern California
Los Angeles, California

Pierre-David Pfister
CSEM
Neuchâtel, Switzerland

Christian Piguet
CSEM & LAP-EPFL
Neuchâtel, Switzerland

Sébastien Pillement
ENSSAT/University of Rennes
Lannion, France

Nathan M. Pletcher
University of California–Berkeley
Berkeley, California

Jan M. Rabaey
University of California–Berkeley
Berkeley, California

Flavio Rampogna
CSEM
Neuchâtel, Switzerland

Marc Renaudin
TIMA Laboratory
Grenoble, France

Mohammed Es Sahliene
TIMA Laboratory
Grenoble, France

Olivier Sentieys
ENSSAT/University of Rennes
Lannion, France

Kamel Slimani
TIMA Laboratory
Grenoble, France

Phillip Stanley-Marbell
Carnegie Mellon University
Pittsburgh, Pennsylvania

Philip Strenski
IBM Watson Research Center
Yorktown Heights, New York

Eric Tell
Linköping University
Linköping, Sweden

Ingrid Verbauwhede
University of California–Los Angeles
Los Angeles, California

Patrick Volet
CSEM
Neuchâtel, Switzerland

Terry Tao Ye
Stanford University
Stanford, California

Sungjoo Yoo
TIMA Laboratory
Grenoble, France

Victor Zyuban
IBM Watson Research Center
Yorktown Heights, New York

I

Low-Power Processors and Memories

1

Techniques for Power and Process Variation Minimization

Lawrence T. Clark
Arizona State University

Vivek De
Intel Labs

1.1 Introduction

For more than a decade, integrated circuit (IC) power has been steadily increasing due to higher integration and performance enabled by process scaling. As shrinking transistor dimensions are fabricated, and as the absolute value of the dimensions diminish, greater device variations must be addressed. Until recently, increased power was driven primarily by active switching power. Threshold voltages must be decreased to maintain performance at the lower supply voltages required by thinner oxides, however, raising drain to source leakage exponentially. Steeper doping gradients and higher electric fields increase other leakage components, giving rise in sub-0.25-μm generations to DC leakage currents that may limit overall power and performance in future chips. This comes on top of still increasing active power dissipation, driven by architectural changes such as greater parallelism and deeper pipelining. The latter

implies fewer gates per stage and, in turn, requires more aggressive circuit techniques such as domino, which can also increase active power. Having fewer logic stages increases the susceptibility to process variations. Finally, as scaling requires lower voltages, in-die and system-level voltage variations are also increasingly problematic.

The focus of this chapter includes the design implications of increasing device variation and leakage. The mechanisms are a direct result of basic physics and will continue to grow in importance over time, requiring design effort to mitigate them. Variation in microprocessor frequency has been dealt with by "speed binning," whereby faster dies are separated and sold at a premium. Dies with inadequate speed or excessive standby current are discarded. These yield considerations are important for robust design. We also discuss design techniques, notably the application of body bias and supply voltage adjustment, which can help deal with both variation and average leakage, as well as active power. Examples from fabricated designs demonstrating the efficacy of the techniques are discussed.

1.2 Integrated Circuit Power

Increasing leakage currents are a natural by-product of transistor scaling and comprise a significant portion of the total power since the 0.25-μm-process generation. By the 90-nm technology node, it can contribute over a fifth of the total IC power on high-performance products [1]. The profusion of battery-powered "hand-held" devices introduced in recent years (e.g., cell phones and personal digital assistants) has made power management a first-order design consideration. These sections focus on circuit design approaches to alleviate leakage power using reverse body bias (RBB) "Drowsy" mode when an IC is in a standby mode and later, in Section 18.9, optimizing the active power by dynamic voltage management (DVM). Although other implementations are briefly discussed, the bulk of the discussion describes the specific implementation on the 0.18-μm XScale microprocessor cores intended for system-on-chip (SoC) applications [2].

1.2.1 Active Power and Delay

The total power of a static CMOS integrated circuit is given by

$$P_{tot} = P_{dyn} + P_{static} + P_{short\text{-}circuit} \tag{1.1}$$

representing the dynamic power (i.e., that due to charging and discharging capacitances during switching) the static leakage power, and the "short-circuit" or crowbar power due to both P and N transistors being on simultaneously during a switching event, respectively. The latter term tracks with the active power and is generally on the order of 5% or less for well-designed circuits. It is typically ignored, as it will be here. The dynamic power of a digital circuit follows the well-known

$$P_{dyn} = a/2\ C\ V_{dd}^2\ F \tag{1.2}$$

where C is the switched capacitance, V_{dd} is the power supply voltage, F is the operating frequency, and a is the switching activity factor measured in transitions per clock cycle. Leveraging the V_{dd}^2 dependency is consequently the most effective method for lowering digital system power; however, the switching speed of a digital circuit with a fixed input slope and fixed load is given by Chen and Hu [3]:

$$T_{delay} = K\ V_{dd}/(V_{dd} - V_t)^{\alpha} \tag{1.3}$$

where α* is typically 1.1 to 1.5 for modern velocity saturated devices, tending toward the former for NMOS and the latter for PMOS [4], and K is a constant depending on the process. To first order, this

* α is typically used as in the literature.

delay dependency on voltage can be treated as linear. The concept of DVM is to limit the V_{dd} and frequency such that the application latency constraints are met, but the energy to perform the application function is minimized by following the square law dependency of Equation (1.2) instead of linearly tracking F. The chosen frequency F, representing the reciprocal of the worst-case path delay, is constrained by Equation (1.3) for a given supply voltage.

1.2.2 Leakage Power

Leakage power sources are numerous [5], with the primary contributor historically being transistor off state drain to source leakage (I_{off}). For modern processes having gate dielectric thicknesses under 3 nm, gate leakage I_{gate} is becoming a larger contributor but is generally smaller than I_{off}, particularly at high temperatures, given the stronger temperature dependency of 8–12×/100°C for I_{off} vs. approximately 2×/100°C for I_{gate}. I_{off} increases on scaled transistors because, to maintain performance, V_t must be lowered to compensate for decrease in V_{dd}. This increases the leakage according to

$$I_{off} \propto e^{-V_t/(S/\ln 10)} \tag{1.4}$$

where S is the subthreshold swing given by

$$S = \frac{kT(\ln 10)}{q}\left(1+\frac{C_D}{C_{OX}}\right) \tag{1.5}$$

where k is the Boltzmann constant, T is the temperature in Kelvin, q is the elementary charge, C_D is the depletion layer capacitance, and C_{OX} is the gate oxide capacitance. Noting that C_D is nonvanishing, the subthreshold swing parameter S is essentially a fixed parameter for Si MOSFETs, typically 80–100 mV/decade depending upon the process at room temperature. Referring to Equation (1.4), it is obvious that lowering V_t affects the I_{off} exponentially.

For gate oxide thicknesses below 3 nm, quantum mechanical (direct band-to-band) tunneling current becomes significant. This leakage is extremely voltage dependent, increasing approximately with V^3 [6]. It also increases dramatically with decreasing thickness (e.g., increasing 10× for a change from 2.2 nm to 2.0 nm [7]). Gate-induced drain leakage (GIDL) at the gate-drain edge is important at low current levels and high applied voltages. It is most prevalent in the NMOS transistors where it is about two orders of magnitude greater than for PMOS devices. For a gate having a 0-V bias with the drain at V_{dd}, significant band bending occurs in the drain region, allowing electron-hole pair creation. Essentially, the gate voltage attempts to invert the drain region, but because the holes are rapidly swept out, a deep depletion condition occurs [8]. The onset of this mechanism can be lessened by limiting the drain to gate voltage. It can be exacerbated by high source or drain to body voltages. Diode area leakage components from both the source-drain diodes and the well diodes are generally negligible with respect to I_{off} and GIDL components. This is also improved by compensation implants intended to limit the junction capacitance. However, transistor scaling requires increasingly steep (often halo) doping profiles increasing band-to-band tunneling (BTBT) currents at the drain to channel edge, particularly as the drain to bulk bias is increased. This component may also limit use of RBB on sub-0.18-μm processes. Controlling these leakages will be key to effective use of body biasing and will require careful circuit design as well as appropriate transistor architecture.

1.3 Process Selection and Rationale

Thinner oxides are required to allow transistor length scaling while maintaining channel control. These scaled oxides require lower supply voltages to limit electric fields to a reliable value. Additionally, to maintain performance at lower voltage by retaining gate overdrive $V_{dd} - V_t$ it is necessary to lower V_t. For

handheld battery powered devices, V_t must be chosen to balance standby power with active power dissipation for maximum battery life. Absent clever design to mitigate leakage, the duty cycle between standby and active operation for the given application determines the optimal threshold voltage [9]. This leads to considerable divergence in future processes and considerable power constraints to scaling processes used for portable devices [10]. One of the purposes of circuit techniques to limit active and standby power is to help widen the allowable V_t and process performance range. Handheld battery lifetime requires IC standby currents below 500 μA requiring total leakage under 100 pA/μm of transistor width. This implies a V_t over 500 mV, independent of supply voltage, increasing active power at the same performance level.

Figure 1.1 plots the simulated power vs. performance for a microprocessor operating at different frequencies on processes with different V_t, assuming complete flexibility in the supply voltage or DVM (i.e., the voltage is chosen such that it is just sufficient to meet the processor frequency). The curves are based on the transistor performance metric described in Thompson [11] and normalized to the microprocessor performance with V_t of 390 mV (solid line in both plots) and 500 mV (dashed line in both plots). Figure 1.1(a) emphasizes the active power, which depicts the greater overall performance available from the lower V_t process. Note the improved power vs. the linear characteristic that would be obtained by scaling frequency alone. Figure 1.1(b) plots the log scale power for the low frequency ranges. At low frequencies, it is assumed that the power supply voltage cannot be scaled below a minimum value due to circuit functionality constraints. This value is 0.6 V for the 390-mV and 0.7 V for the 500-mV processes. Below the minimum operating voltage, the clock frequency is lowered resulting in a linear, instead of quadratic power savings. The break between square law and linear behavior is evident in the log scale plot of Figure 1.1(b). It is apparent that the lower V_t process has a higher leakage, as indicated by the zero frequency point, while it has a lower active power at the same frequency. It is also capable of higher overall performance. The lower active power is the result of reaching a given performance at a lower voltage, and its benefit was presented in Equation (1.2). The dotted line in Figure 1.1(b) demonstrates that with the addition of RBB Drowsy mode, the higher-performance process is power competitive at low effective frequencies with the slower process. The methods for achieving this comprise Section 1.4 and Section 1.5.

Nonstate-retentive sleep modes also incur power penalties. The present logical state must be saved before sleep and restored upon resuming active operation, requiring a low standby power storage medium. The data movement requires time and power that must be amortized by the leakage power savings achieved in the time in sleep. This can preclude frequent use. If the storage is off-chip, the higher IO voltages and off-chip capacitances increase the power penalty. A number of schemes, ranging from "greedy" to timeout based, have been proposed for determining when to enter a low-power state. The key considerations are achieving low energy cost to entry and exit, as well as low latency to awaken and respond to input.

1.3.1 Effective Frequency

For compute intensive applications, the active power dominates as illustrated in Figure 1.1. The leakage power is of interest when the compute demands are modest, for instance when a processor is waiting for user input or in a cell phone, in the intervals between contacts with the cell. The former can be expected to be multiple times per second, and the latter less than once per second [12]. The total computed cycles per second is very low, although the frequency of the part might be higher as described next. Here, the term "effective frequency" is used to mean the number of cycles of computation accomplished over a given period. The actual frequency may vary during that time, according to whether the processor is running or is in a low-power Drowsy mode. Effective frequency is a measure of the average actual work performed by the processor. For example, assume that the processor receives interrupts at an average frequency determined by the application, for instance, from keystrokes on a keypad. Each interrupt awakens the processor where it computes for a number of cycles required to process the input (e.g., add it to the display buffer). The computational requirements might be quite different depending on the type of interrupt that is being serviced — it may be a command to sort mail messages. The effective frequency

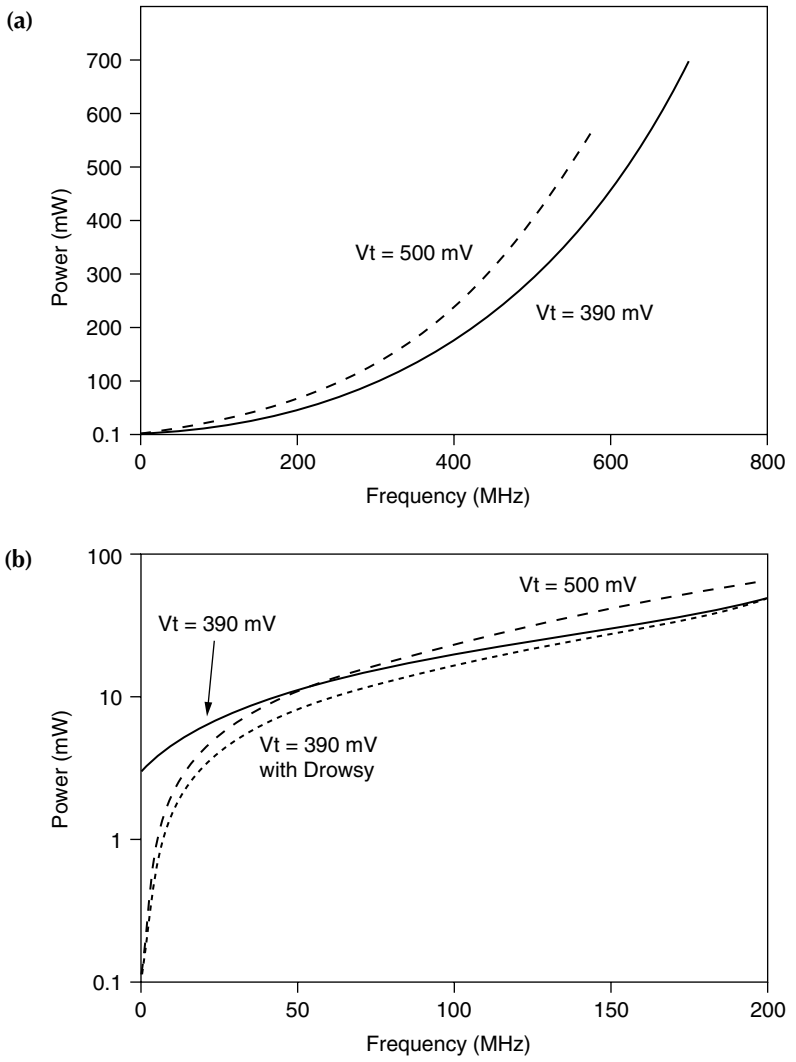


FIGURE 1.1 (a) The effect of V_t on power vs. frequency, and (b) the low frequency, leakage dominated power levels. In the upper plot, the low V_t with Drowsy is coincident with the non-Drowsy.

is then the total long-term average number of useful clocks per unit time (i.e., the number of instructions per interrupt times the number of interrupts). For example, with a 100-Hz interrupt rate and 100,000 instructions per interrupt, the processor will have an effective frequency of (100 Hz * $100(10)^3$) = 10 MHz, although the clock rate may be much higher (e.g., 300 MHz).

1.4 Leakage Control via Reverse Body Bias

RBB has been suggested for leakage control for some time [13,14]. Essentially, this leverages the well-known body effect, that raises the V_t of a transistor having a source voltage above the bulk, as commonly occurs in the upper transistors of an NMOS stack during switching. Although normally a designer's bane that reduces circuit speed, it can be used to advantage because

$$V_t = V_{FB} + \gamma\sqrt{\phi_s - V_{bs}} - K_2(\phi_s - V_{bs}) - \eta V_{ds} \tag{1.6}$$

where γ is the body effect coefficient, which, along with K_2 models nonuniform doping [15]. These coefficients represent the efficacy of a change in the source to body voltage in modulating I_{off}. η is the drain induced barrier lowering (DIBL) coefficient, which represents the ability to Control V_t by applying drain bias. Drain and body bias also affect the subthreshold slope.

RBB to modulate leakage has a number of advantages:

1. It is a circuit design approach.
2. It does not adversely affect the active performance.
3. It is state retentive.

The first point allows this approach to be utilized on any process. Longer channel lengths generally have a stronger body effect [16], under designer control at the resolution of the drawing grid. The second assumes that the implementation does not incur a significant IR drop or alternatively, it allows improved active power at the same standby current level vs. a device not so equipped. The final point is the advantage over "sleep" modes where the power supply is completely disconnected. With RBB, data is not lost when entering and exiting the low-power state — important in that it allows the power control to be transparent to the operating system and application software and saves significant energy. It is frequently difficult to predict *a priori* how long a device will be in a standby state, particularly when this depends upon user interaction. Retaining precisely the state of the IC before the entrance, as well as minimizing any power penalty to enter or exit the low-power mode makes the mode usable more frequently.

Body bias was used to limit leakage on a 1.8 V microprocessor implemented in a dual-well 0.25-μm process described in Mizuno et al. [17]. This device used separate supplies for both the NMOS and PMOS bulk connections. A strong negative bias greater than 1 V was applied to the NMOS bulk via a charge pump and the PMOS bulks (N wells) were connected to the 3.3-V power supply rail during standby. Hundreds of local switches, distributed across the device, apply the body bias and provide a low impedance bulk connection, at the expense of routing the controls and supplies throughout the layout. This strong biasing is inappropriate for smaller geometry processes, where more abrupt doping and thinner oxides increase second order effects. This implementation of RBB increases GIDL, which can thus be the limiting leakage mechanism. Direct BTBT leakage in the source diodes of sub-130-nm halo doped transistors can be increased to also limit total standby current by reverse biasing the junctions. Consequently, to use RBB effectively on processes beyond 0.25 μm, it will need to be comprehended in the transistor design and the RBB operation should use the lowest effective voltages.

1.4.1 RBB on a 0.18-μM IC

The Intel 80200 microprocessor is an implementation of the XScale microarchitecture implemented in a 0.18-μm process. Although sold commercially as a high-performance embedded device, it was also used as a development vehicle to develop Drowsy mode [18] circuitry and techniques. This mode utilizes RBB as well as $V_{dd} - V_{ss}$ collapse to limit leakage power, achieved via large supply gating transistors that allow the source to be raised. They also allow full collapse of the core voltage, which produces the nonstate-retentive "sleep" mode, essentially the classical multi-threshold CMOS (MTCMOS) approach to leakage control [19]. The manner in which RBB is applied, utilizing lower source to bulk voltages while collapsing the $V_{dd} - V_{ss}$, alleviates second order components. Drowsy mode retains state in all storage elements on the die and is exited on any interrupt. Sleep mode is not state-retentive, requiring a "cold-start." Consequently, asserting reset instead of an interrupt terminates it. The Drowsy implementation and results are described in detail in the following sections.

1.4.2 Circuit Configuration

The circuit configuration is depicted in Figure 1.2. Power pads are on the V_{dd}, $V_{dd(IO)}$, and $V_{ss(GND)}$ pins. Large N channel devices M1 provide V_{ss} to the active circuitry during active operation. Simultaneously, large P channel devices M2 provide clamping of the N well ($V_{dd(SUP)}$) providing the PMOS bulk connection

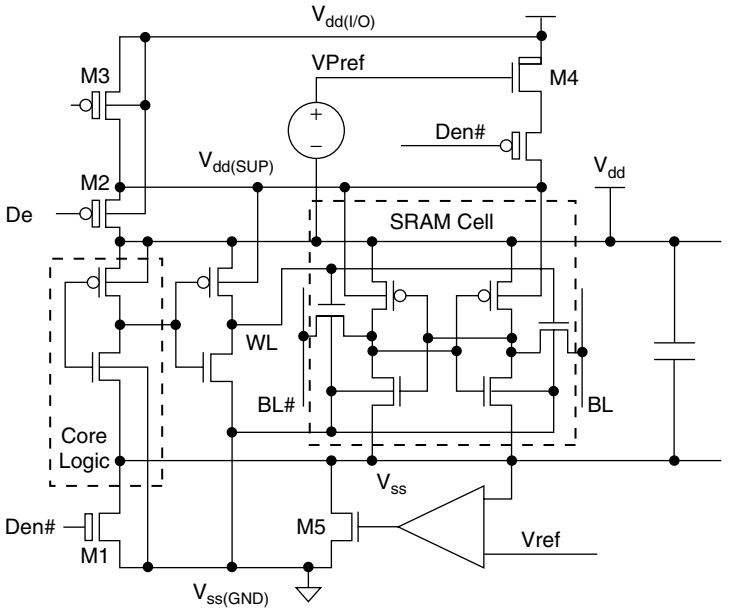


FIGURE 1.2 Circuit configuration for RBB Drowsy mode.

to V_{dd}. These transistors must be thick oxide because they are exposed to high voltages as indicated — here, the thick gate IO transistors are used. The PMOS clamping transistors carry no DC current and are 15 mm in total width. The NMOS clamp transistors carry the entire power supply current during operation and must do so with minimal IR drop. They are 85 mm in total width, which is less than 2% of the total transistor width of the microprocessor. This high ratio between the rest of the core is indicative of the low activity factor achieved by the design and relies on adequate on-die decoupling capacitance to provide instantaneous current demand. To this end, total of 55 nF of decoupling capacitance was interspersed among the active circuitry.

For sleep as well as Drowsy modes, the transistors comprising M1 are in cutoff. In the former cases, the core V_{ss} is allowed to float to V_{dd}, and power consumption is dominated by the leakage current through the NMOS clamp devices. The clamp devices should be high V_t to minimize this current because they do not have body bias applied. In Drowsy mode, to apply body bias to the NMOS devices, V_{ss} is allowed to rise toward V_{dd} but regulated to avoid losing state. Raising the NMOS source voltage instead of decreasing the NMOS body voltage is advantageous because it does not require a twin-tub or triple-well process, nor charge pump circuitry. It also lowers the I_{off} by the η coefficient of Equation (1.6) as well as limiting GIDL components because drain to bulk voltage is not increased. Because gate current is strongly affected by the drain to gate voltage, it is substantially reduced on processes with thin oxides. Another regulator provides a high voltage to the $V_{dd(SUP)}$ node, to reverse body bias the PMOS transistors. In the static random access memory (SRAM), the word-lines are driven to $V_{ss(GND)}$ as presented in Figure 1.2. This places a negative gate-to-source bias on the SRAM pass devices as presented, lowering the SRAM current a further 40%. This may not be desirable for thin oxides as it can increase the gate leakage component beyond the I_{off} savings.

Simulated waveforms of the V_{ss} and $V_{dd(SUP)}$ nodes are plotted in Figure 1.3, at 110°C. Minimal overshoot can be discerned in the figure. In Drowsy mode, the V_{ss} node rises to approximately 650 mV, with some PVT variation. $V_{dd(SUP)}$ is driven to 750 mV above V_{dd}. At room temperature, the V_{ss} node takes approximately half an msec to rise because it is pulled toward V_{dd} solely by leakage. The advantage of this passive V_{ss} rise is that movement of this highly capacitive node is limited if Drowsy is exited soon after entrance, limiting the power cost of using this mode. No energy is explicitly expended to enter the mode because it achieved by transferring charge from the core nodes to the V_{ss} node, instead of supplying

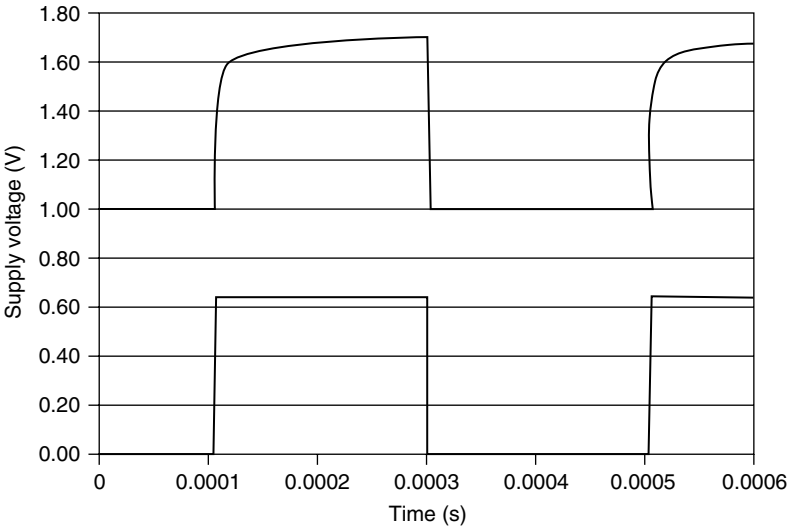


FIGURE 1.3 Simulated V_{ss} and $V_{dd(SUP)}$ waveforms at 110°C.

it from the IC power pins. This is not possible on the PMOS bulk node. This regulator circuit is designed with limited drive, as that node is less capacitive at 5 nF, and with low current demand, generally just the diode contributions of the N-well and PMOS source-and-drain diodes.

1.4.3 Layout

Application of any body bias requires separate bulk and source supplies for both P and N transistors. This design opts for minimal intrusion due to the separate body connections. The power supply clamping transistors are provided in the pad ring only, occupying otherwise empty (or IO decoupling capacitor) space within the supply pins. Because the core was over 4000 µm per side, circuits could be over 2000 µm from the nearest clamp. Additionally, the bulk connections are routed sparsely through the logic circuitry, limiting the density impact. This is feasible because these provide no DC power, making resistance less important. A two-layer routing grid with 50 µm between bulk supplies was utilized. The substrate is highly doped, providing an effective short circuit between V_{ss} (ground) rails and limiting noise due to switching. N-wells are intentionally contiguous, forming a grid at the substrate level for $V_{dd(SUP)}$.

1.4.4 Regulator Design

The V_{ss} regulator comprises Figure 1.4(a) and strictly limits the regulator overhead power. The output voltage must be essentially constant over 3 decades of current demand at all process, temperature, and voltage (P, V, T) corners (see Section 1.6). At the high end, when entering the low-power state directly from high frequency operation the die may be hot, where MOS drain to source leakage may be over 100× the RBB low temperature leakage and must be provided to avoid collapsing logic state. As expected, the amplifier compares the voltage on V_{ss} with a reference voltage. A PMOS stack simulating a resistor string, which allows it to vary with power supply variations, generates this reference. In this manner, higher supplies allow larger body bias — this flexibility was desirable for a test device. The resistor stack current is under 100 nA and is continuously biased in all modes. The regulator is a three-stage amplifier with an NMOS output transistor M5. Three stages were required due to the bias conditions and low current requirements to keep the regulator power consumption less than 5% of the total standby power at the typical process corner. The output transistor is sized to provide the full IC leakage current at high temperature and the worst-case process corner. The first stage is a differential operational transductance amplifier (OTA), while the second buffer stage provides increased voltage output range and current drive to the gate of M5. The first and second stages combined use less than 4 µA at typical operating conditions.

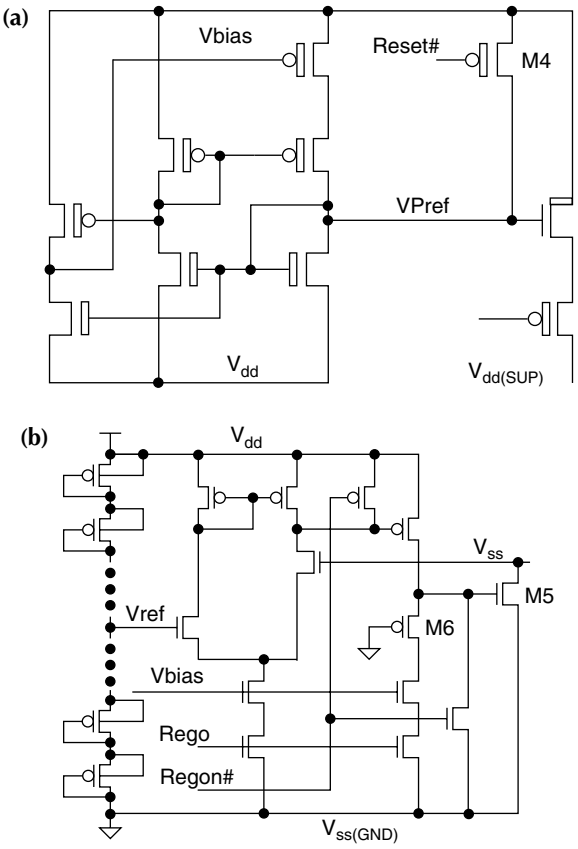


FIGURE 1.4 V_{ss} (a) and $V_{dd(SUP)}$ (b) supply regulation circuits. All NMOS share substrate $V_{ss(SUP)}$, and all PMOS in (a) have V_{dd} and in (b) $V_{dd(SUP)}$ body connections.

At such low current levels, gain is limited, which improves stability, as discussed next. Slew rate also suffers, which makes the step response poor. To address this, the buffer stage includes the diode connected transistor M6, which, combined with proper sizing keeps transistor M5 from completely cutting off, except in sleep mode. The enables are evident in the figure.

Stability must be ensured at all P, V, T conditions and overshoot on V_{ss} must be limited. Entering the body bias state, which is essentially a voltage step on V_{ss} represents the worst-case stability condition. Adequate phase margin ensures stability of the system comprised of the regulator and V_{ss} node on the IC. Overshoot on V_{ss}, even momentarily, can cause state loss. The circuit poles may be approximated by the dominant terms to simplify the analysis. The V_{ss} node is controlled to first order by the output conductance of transistor M5, while the amplifier pole is dominant. The former pole is at approximately 670 kHz calculated from the small signal parameters, while the latter is at 9 kHz. The low gain of the amplifier produces a low unity gain bandwidth and greater than 60 degrees of phase margin at the typical process. Essentially, the highly capacitive V_{ss} node low-pass characteristic does not require high amplifier speed for stability.

To back-bias the PMOS devices, two schemes may be used. At low IO voltages (e.g., 1.8V), the PMOS transistor bulk node may be directly connected to this voltage via M3. For higher IO voltages diminishing leakage reduction does not offset the greater charge switched in raising the well voltage. Therefore, in this case, this voltage is regulated. The open loop regulator is depicted in Figure 1.4(b), which derives a constant voltage from the IO supply $V_{dd(IO)}$. It is worth noting that as long as circuit configurations that accumulate the gates of the PMOS transistors are avoided, high voltages may be applied to the bulk without oxide stress or damage. The regulator is a bootstrapped voltage reference driving a wide NMOS vertical drain transistor in a source follower configuration as presented in Figure 1.4(b). This device (M4

in Figure 1.2) has a naturally low V_t and operates in subthreshold, providing a negligible voltage drop from the reference voltage to $V_{dd(SUP)}$ in operation. The vertical drain configuration allows the thin gate oxide device to tolerate high drain to gate voltages as in Clark [21].

The relatively high active current of the phase-locked loop (PLL) necessitates disabling it in Drowsy mode. Leaving standby mode requires the PLL to restart and lock, triggered by an external interrupt. Because this takes approximately 20 μs, the mode is usable often (e.g., between keystrokes). On the prototype, the lock time is set by a counter to enable deterministic testing. In actuality, the PLL lock time can be as low as 2 μs depending on voltage. Faster interrupt latencies can be supported by providing the PLL reference clock directly to the IC, while the PLL locks. Consequently, PLL lock-time need not affect interrupt latency or limit the applicability of Drowsy usage.

1.4.5 Limits of Operation

All memory, such as latches, need to be able to hold a "0" or a "1" with RBB applied. Although it is more difficult from a circuit aspect, holding state in all elements greatly simplifies logic design verification. As V_{dd} and V_{ss} collapse toward one another the transistors move from saturation into subthreshold, as the reverse body bias increases V_t and the increase in V_{ss} decreases V_{gs}. In subthreshold, these "on" transistors rapidly weaken with their current following the subthreshold slope. In a memory element, the voltage level of a node is maintained by an "on" transistor being able to supply enough current to overcome the leakage of all the attached "off" transistors. In normal, high V_{ds} operation, this is not a problem due to the large I_{on} to I_{off} ratio. As transistors reach subthreshold, the on current drops rapidly with V_{ds} (= V_{gs}) due to

$$I_{ds,sat} = \frac{\mu C_{ox} Z}{2L}\left(V_{gs} - V_t\right)^{\alpha} \tag{1.7}$$

becoming Equation (1.4) as the gate overdrive (V_{gs}–V_t) is reduced below 0. Ideally, V_{dd}–V_{ss} can be lowered to drive all of the transistors into subthreshold operation because the I_{on}/I_{off} ratio will scale for all transistors. Assuming an 80-mV/decade transistor subthreshold characteristic, over three decades of current difference between on and off transistors will be maintained with 250 mV of V_{ds}. Lowering the voltage too far on future ultra-small devices will reach thermodynamic constraints [22]. The relative size and strength of the N and P transistors, including local channel length and V_t variation must be considered. In practice, state loss depends upon many factors such as the type of latch, the transistor ratios, the logic state being held, the local transistor V_t and the temperature. Domino circuits, with the largest N to P (keeper) width ratios, are the first to fail.

The fail point as a function of the PMOS body voltage and NMOS body voltage as measured on silicon is presented in Figure 1.5. Because in this design the V_{ss} is referenced to the $V_{ss(GND)}$ supply node, V_{ss} is the applied NMOS body bias. Points lower on the vertical axis have higher NMOS V_t and further right have higher PMOS V_t. Measured parts retained state below the curve (Pass) and lost state above it after application of that level of reverse body bias (Fail). As V_{ss} is increased, the NMOS transistors have increasing reverse body bias applied to them, so "on" devices are in subthreshold. The right side of the curve represents a memory element failing as logic "0" is flipped to a "1." As $V_{dd(sup)}$ increases the PMOS transistors leakage is reduced, so that the amount of reverse body bias that can be applied to the NMOS transistors can be increased, continuing until a maximum value of $V_{dd(SUP)}$ and V_{ss} is reached. The left part of the curve represents the converse case where the PMOS transistors are weakened with respect to the NMOS. With a large $V_{dd(SUP)}$ applied "on" devices are in subthreshold and are eventually unable to supply enough current to overcome leakage from NMOS transistors. This left part of the curve represents a memory element holding a "1" flipping to a "0." The flat zone depicts the saturation of any body effect as voltage increases.

1.4.6 Measured Results

When the leakage current from the microprocessor is low the voltage on V_{ss} will not rise to the reference voltage because the regulator does not actively drive its output. At V_{dd} of 1.05 V, the regulator clamps at

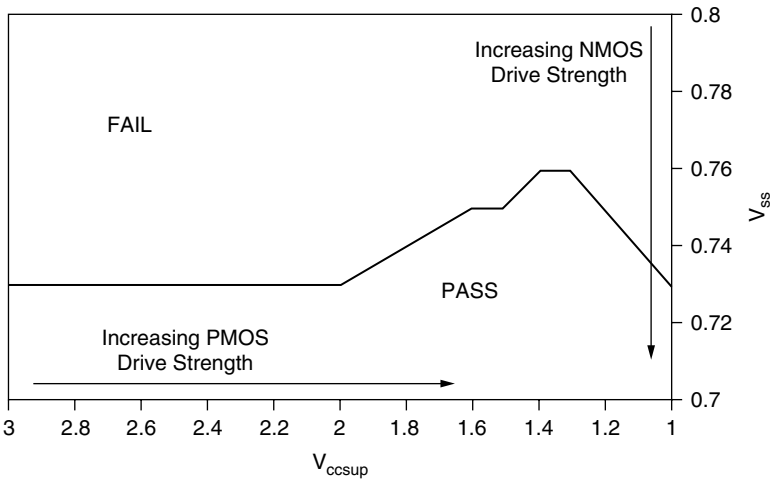


FIGURE 1.5 Shmoo plot of state retention with PMOS and NMOS body bias as parameters.

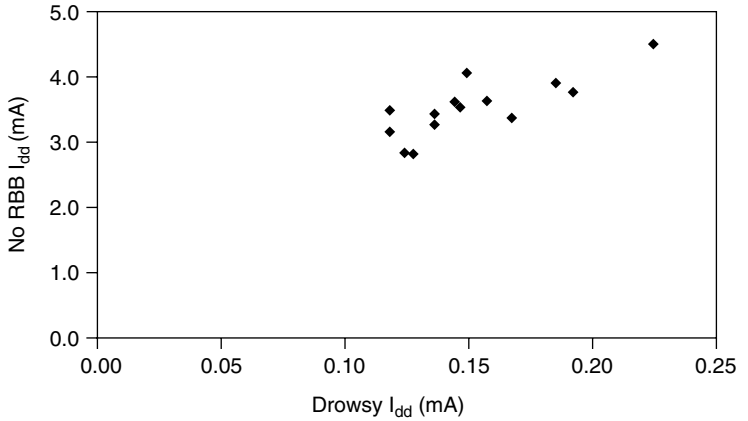


FIGURE 1.6 Standby current of the microprocessor with and without body bias.

the reference voltage, about 0.73 V for high leakage. The leakage current is reduced by a factor of over 25 across most devices when the body bias is applied. Figure 1.6 plots the no body-bias (NBB) standby vs. the RBB Drowsy mode current. Figure 1.7 gives the distribution of the current with reverse body bias for all die on one wafer. A wide variation, due to variations in the process (e.g., threshold voltage and channel length) as well as the regulator output, is evident.

1.5 System Level Performance

This section describes experiments using Drowsy mode to simulate low leakage by running an IC in short bursts of operation interspersed with time in the leakage control mode, time domain multiplexing (TDM) Drowsy mode [23]. The IC power is dominated by different components in different operating modes described in Section 1.1. First, the active power component dominates during intervals of operation. Second, there are the two primary leakage components, the active component, potentially multiplied by die heating, but still small compared with the active power, and leakage during Drowsy mode. Third, the PLL and clock-generation power, as well as that of the interrupt circuitry required to wake up the device from Drowsy mode. The former provide an active power component that runs during active operation and for 20 μs before each active interval. There is a small non-RBB leakage component

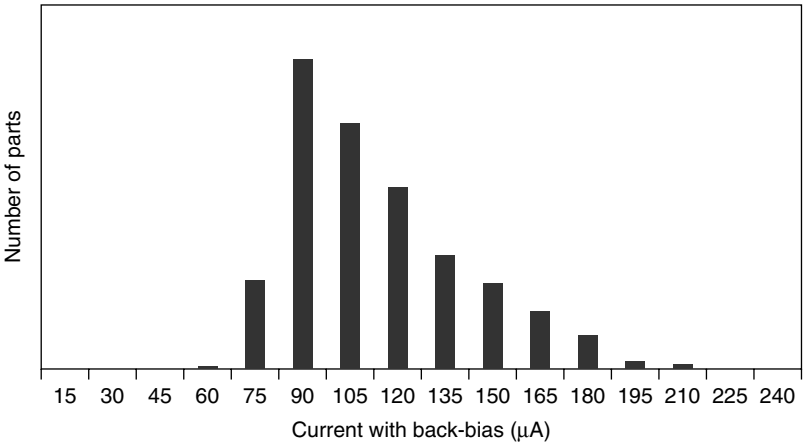


FIGURE 1.7 Standby current of the microprocessor with body bias.

during PLL startup, but the time is small enough to make this component negligible. Finally, the power cost of each power supply movement represents the "penalty" power of entering and exiting Drowsy mode. This low frequency high capacitance switching power is mitigated by the low-voltage swing utilized. The energy to transition the clamp transistors and their driver circuitry is small enough to be considered negligible because the total gate capacitance of the clamp transistors is 119 pF. Entry into the standby mode consumes no power on the V_{ss} node due to its being driven high by passive leakage of the core (i.e., redistribution of charge from the nodes within the core logic to V_{ss}). Power dissipation is incurred only when leaving the mode. The total energy cost of a single entry into the low-power mode is calculated to be 30.6 nJ from measurements.

The experiments were performed on a microprocessor board [24] at 1 V V_{dd} running IO at 100 MHz and with a core frequency of 300 MHz. Separate core and analog PLL supplies connected to external power supply and ammeter connections allowed these currents to be distinguished. The Drowsy circuitry allows high performance — the device under test was run on the board to 800 MHz at 1.55 V. Instantaneous current demand was measured, while the interrupt signal was asserted at the chosen interrupt frequency. Each interrupt runs code comprised of a simple loop, intended to be representative of the power that would be consumed by the typical instructions, which are generally quite similar [25]. The number of instructions to run at each interrupt is set by a loop count parameter. At the end of the loop, the IC returns to Drowsy mode. Subsequent interrupts wake the microprocessor and begin the loop anew. Due to branch prediction, the processor executes one instruction per clock in the loop (i.e., there are no stall cycles). State retention while Drowsy maintains the cached instructions, so there are no misses after the first compulsory ones. The operating voltage was adjusted based on the reading from a locally connected voltmeter in order to account for IR loss in the power supply leads. Figure 1.8 is a representative power measurement.

1.5.1 System Measurement Results

Drowsy power was measured to be 0.1 mA at 1 V on the IC used in these measurements. In a DC condition, the I_{sb} at room temperature (i.e., the standby core supply current with no clock running) was 2.8 mA at 1 V. The PLL consumes 6.6 mW at the same voltage. The processor was run at a number of interrupt frequencies and instruction per interrupt rates with the results plotted in Figure 1.9. As expected, the power shows a linear dependency on the effective frequency at high rates, where the active power dominates, while at low rates a floor due to leakage components is evident. The energy per instruction is calculated to be 0.5 nJ. All interrupt and instruction rates fall on the same curve as presented in the figure. Measurements made in "idle" mode, in which the PLL is kept active, no RBB is applied, but clocks that are gated at the PLL generate a relatively high power floor due to PLL power.

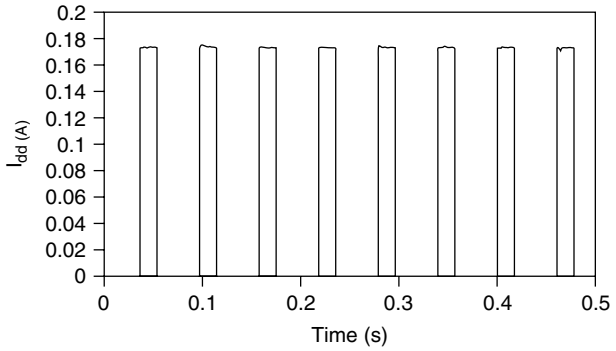


FIGURE 1.8 Current measurement with the time in standby and active modes evident.

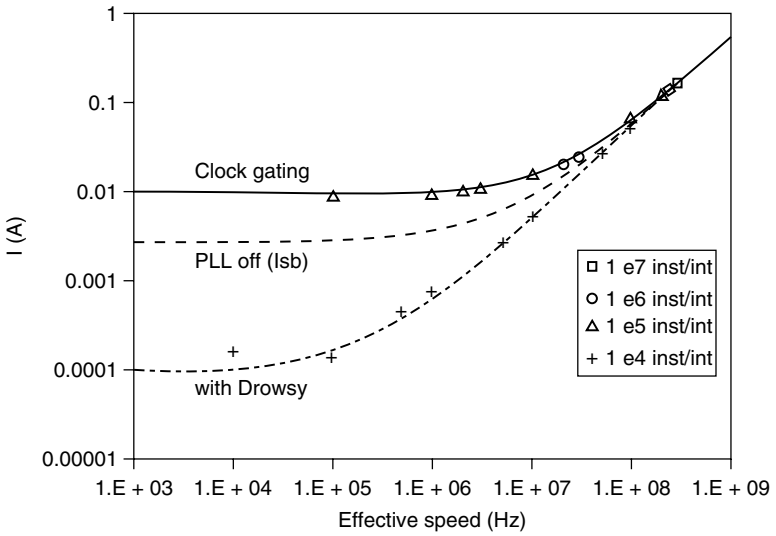


FIGURE 1.9 Current measurement depicting active and leakage power dominated frequencies.

The power savings of using Drowsy mode over clock gating alone is approximately 100×. This is over 25× less than the I_{sb} leakage power floor that would be obtained without Drowsy (e.g., by simply lowering the clock to a very low rate). Power is saved and the response time to external stimulus is improved by running in short bursts at high frequencies. Effective frequency allows direct comparison with devices running at lower frequency demonstrating that the efficacy of TDM Drowsy mode, matching the theoretical curve of Figure 1.1. By raising the V_t with RBB to achieve low standby power, it is combined with improved low voltage and higher maximum performance. The active power improvement can be estimated by considering the V_t increase required to match the I_{off} reduction and the required V_{dd} increase to achieve the same performance. By simulating the circuit metric mentioned previously, calibrated to the measured frequency vs. voltage performance of the microprocessor, a V_t increase of 110 mV (to a typical value of 500 mV) results in the same reduction. At this V_t, the same frequency at $V_{dd} = 0.75$ V is obtained by an increase to 0.86 V demonstrating an active power savings of 24% by using Drowsy mode instead of a higher V_t.

1.6 Process, Voltage, and Temperature Variations

Systematic and random variations in P, V, and T are posing a major challenge to future high-performance microprocessor design [26,27]. Technology scaling beyond 90 nm is causing higher levels of device

parameter variations, which are changing the design problem from deterministic to probabilistic [28,29]. The demand for low power and thinner gate oxides causes supply voltage scaling. Then, voltage variations become a significant part of the overall challenge. Finally, the quest for growth in operating frequency is manifested in significantly high junction temperature and within die temperature variation.

1.6.1 Process Variation

Distributions of frequency and standby leakage current (I_{sb}) of microprocessors on a wafer are presented in Figure 1.10. The spread in frequency and leakage distributions is due to variation in transistor parameters, causing about 20× maximum variation in chip leakage and 30% maximum spread in chip frequency. This variation in frequency has led to the concept of "frequency binning" to maximize revenue from each wafer. Note that the highest frequency chips have a wide distribution of leakage, and for a given leakage, there is a wide distribution in the frequency of the chips. The highest-frequency chips with large I_{sb}, and low-frequency chips with too high I_{sb} may have to be discarded, thus affecting yield. Limits to maximum acceptable I_{sb} are dictated by total active power as affordable by cost-effective cooling and current delivery capabilities, as well as idle power required to achieve a target battery life in portable applications. The spreads in standby current and frequency are due to variations in channel length and threshold voltage, both within die and from die to die. That leakages are affected exponentially, while delay is affected approximately linearly is evident in their relative magnitudes. Figure 1.11 illustrates the die-to-die V_t distribution and its resulting chip I_{sb} variation. V_t variation is normally distributed and its 3-σ variation is about 30 mV in an 180-nm CMOS logic technology. This variation causes significant spreads in circuit performance and leakage. The most critical paths in a chip may be different from chip to chip. Figure 1.11 also presents the 20× I_{sb} variation distribution in detail.

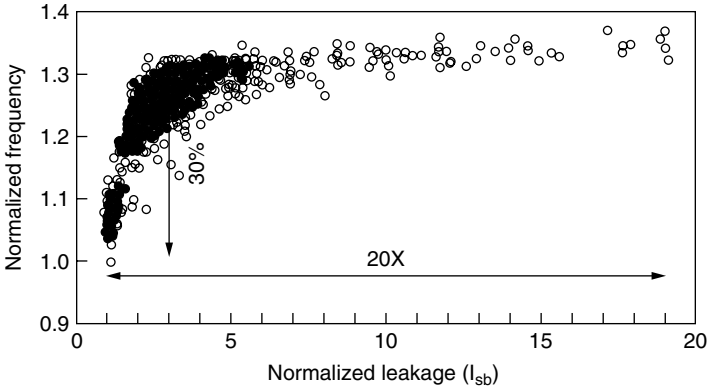


FIGURE 1.10 Leakage and frequency variations.

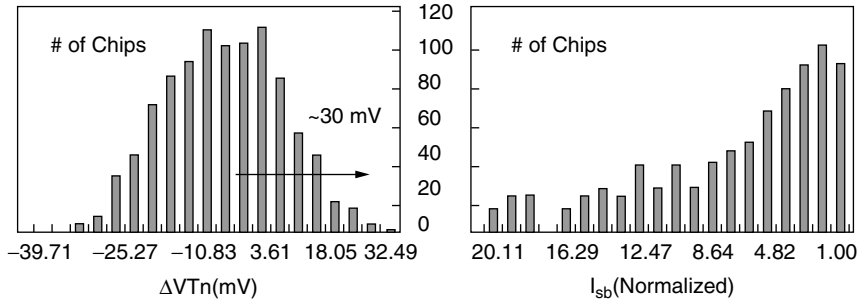


FIGURE 1.11 Die-to-die V_t, I_{sb} variation.

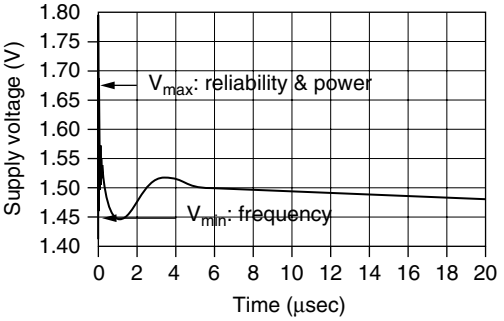


FIGURE 1.12 Supply voltage variation.

1.6.2 Supply Voltage Variation

Uneven and variable switching activity across the die and diversity of the type of logic, result in uneven power dissipation across the die. This variation results in uneven supply voltage distribution and temperature hot spots, across a die, causing transistor subthreshold leakage variation across the die. Supply voltage (V_{dd}) will continue to scale modestly by 15%, not by the historic 30% per generation, first, due to difficulties in scaling threshold voltage and second, to meet the transistor performance goals. Maximum V_{dd} is specified as a reliability limit for a process, and minimum V_{dd} is required for the target performance. V_{dd} variation inside the max–min window is plotted in Figure 1.12. This figure depicts a droop in V_{dd}, when IC current demand changes rapidly, which degrades the performance. This is the result of platform, package, and IC inductances and resistances that do not follow the scaling trends of CMOS process. Specifically, the time "0" point is relatively inactive, while a rapid change in power demand, by the processor leads to the large supply droop pictured. This problem is increased by good low-power design (e.g., clock gating). Power delivery impedance does not scale with V_{dd} and ΔV_{dd} has become a significant percentage of V_{dd}.

1.6.3 Temperature Variation

Figure 1.13 illustrates the thermal image of a leading microprocessor die with hot spots as high as 120°C. Within die temperature fluctuations have existed as a major performance and packaging challenge for

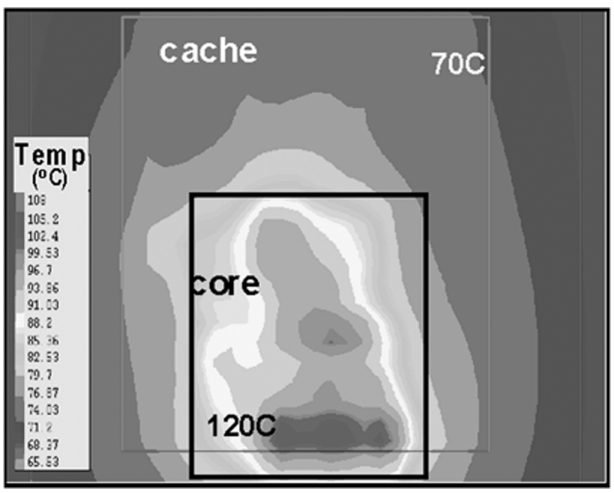


FIGURE 1.13 Within die temperature variation.

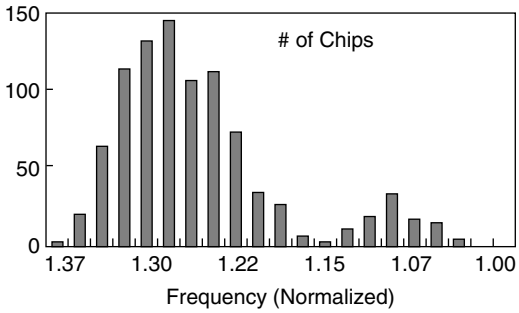


FIGURE 1.14 Die-to-die frequency variation.

many years. Both the device and interconnect performance have temperature dependence, with higher temperature causing performance degradation. Additionally, temperature variation across communicating blocks on the same chip may cause performance mismatches, which may lead to logic or functional failures. Because these thermal variations are the result of uneven local heating, they can be ignored in standby, where lower power dissipation creates minimal heating. Additionally, it can be assumed that the die temperature equals that of the ambient, typically room temperature.

1.7 Variation Impact on Circuits and Microarchitecture

A primary consequence of the P, V, T variation manifests itself as maximum operating frequency (F_{max}) variation. Figure 1.14 presents the distribution of microprocessor dies in 180-nm technology across a frequency range. The data is taken at a fixed voltage and temperature, and thus this F_{max} variation is caused by the process variations discussed previously. This frequency distribution has serious cost implications associated with it — low performing parts need to be discarded, which in turn affects the yield and hence the cost. The P, V, T variations consequently impact all levels of design. For instance, products that have only one operating frequency of interest (e.g., networking devices that either do or do not meet a specific standard) must be designed conservatively. Frequently this means designing all circuits to the worst-case P, V, T corner. This section highlights some of the impact that process has on circuit and microarchitecture design choices.

1.7.1 Design Choice Impact

Dual-V_t circuit designs [30,31] can reduce leakage power during active operation, burn-in, and standby. Two V_t are provided by the process technology for each transistor. High-V_t transistors in performance critical paths are either upsized or are made low-V_t to provide the target chip performance. Because upsizing has limited benefit in gate-dominated paths, as capacitive load is added at the same rate as current drive, lower V_t can be beneficial. Larger transistor sizes increase the relative probability of achieving the target frequency at the expense of switching power. Increasing low-V_t usage also boosts the probability of achieving the desired frequency, but with a penalty in leakage power. It was demonstrated in Karnik et al. and Tschanz et al. [30,31], that by carefully employing low-V_t devices, 24% delay improvement is possible to trade off leakage and switching power components, while maintaining the same total power. However, a design optimized for lowest power by careful assignment of transistor sizes and V_t values is more susceptible to frequency impact due to within-die variations because they sharpen the path delay distributions making a larger number of paths and transistors critical.

1.7.2 Microarchitecture Choice Impact

The number of critical paths that determine the target frequency vary depending on both microarchitecture and circuit design choices. Microarchitecture designs that demand increased parallelism and/or

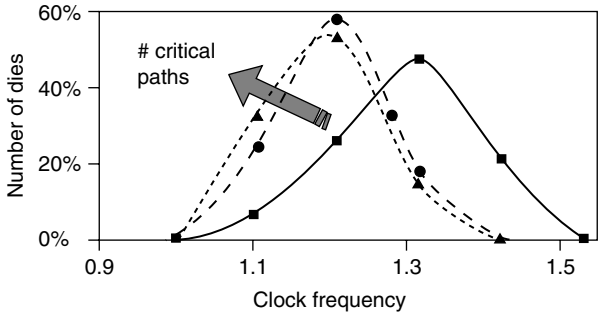


FIGURE 1.15 Die-to-die critical path distribution.

functionality require increase in the number of critical paths. Designs that require deeper pipelining, to support higher frequency of operation, require increase in the number of critical paths and decrease in the logic depth. The impact process variation has on these choices are described next. Test chip measurements in Figure 1.15 demonstrate that as the number of critical paths on a die increases, within-die delay variations among critical paths cause both mean (μ) and standard deviation (σ) of the die frequency distribution to become smaller. This is consistent with statistical simulation results [26] indicating that the impact of within-die parameter variations on die frequency distribution is significant. As the number of critical paths exceeds 14, there is no noticeable change in the frequency distribution. So, microarchitecture designs that increase the number of critical paths will result in reduced mean frequency because the probability that at least one of the paths is slower will increase.

Historically, the logic depth of microarchitecture critical paths has been decreasing to accommodate a 2× growth in the operating frequency every generation, faster than the 42% supported by technology scaling. As the number of logic gates per pipeline stage that determine the frequency of operation reduces, the impact of variation in device parameter increases. Measurement on 49-stage ring oscillators demonstrated that σ of the within-die frequency distribution was 4× smaller than σ of the device saturation current distribution [26]; however, measurements on a test chip containing 16-stage critical paths demonstrate that σ of within die (WID) critical path delay distributions and NMOS/PMOS drive current distributions are comparable. Specifically, NMOS I_{dsat} σ/μ = 5.6%, PMOS I_{dsat} σ/μ = 3.0%, while the 16-stage delay σ/μ = 4.2%. The impact of process variation on the microarchitecture design choices can be summarized as follows: with either smaller logic depth or with increasing number of microarchitecture critical paths, performance improvement is possible. The probability of achieving the target frequency that translates to performance, however, drops due to the impact of within-die process variation.

1.8 Adaptive Techniques and Variation Tolerance

This section describes some of the research and design work to enhance the variation tolerance of circuits and microarchitecture and to reduce the variations by clever circuit and microarchitectural techniques. These techniques expand on those discussed previously by expanding the use of body bias from only RBB to include forward body bias (FBB) to reduce V_t and thereby improve circuit speed. Adjusting the voltage to the optimal required as determined at test time is introduced as another method to increase yield in the presence of variation.

1.8.1 Body Bias Control Techniques

Lowering V_t can improve device performance, with the commensurate increase in leakage and standby current (I_{sb}) as described earlier. One possible method to trade off performance with leakage power is to apply a separate bias to critical devices. In addition to application of RBB to reduce leakage, V_t can be modulated for higher performance by forward body bias (FBB). This method also reduces the impact

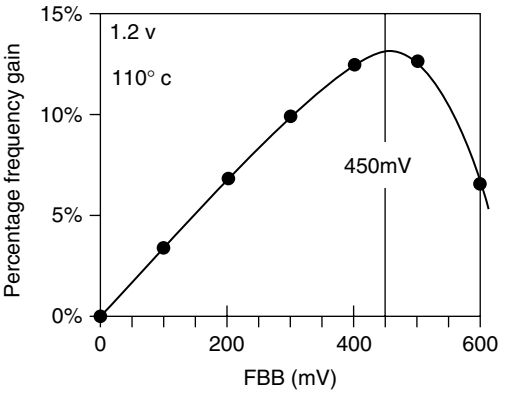


FIGURE 1.16 Optimal FBB for sub-90-nm generations.

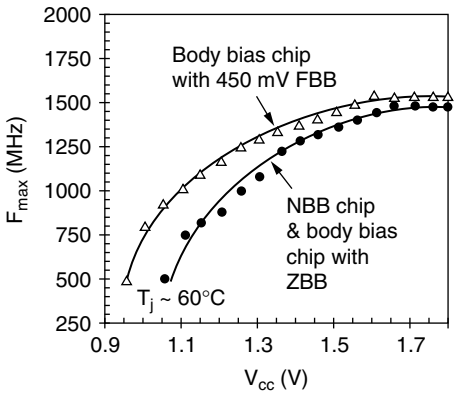


FIGURE 1.17 Forward body bias results.

of short channel effects, hence reducing V_t variations. Figure 1.16 plots the percentage frequency gain as a function of FBB. It was demonstrated empirically that 450 mV is the optimal FBB for sub-90-nm generations at high temperature [32]. A 6.6-M transistor communications router chip [33], with on-chip circuitry to provide FBB to PMOS transistors during active operation and zero body bias (ZBB) during standby mode, was implemented in a 150-nm CMOS technology. Performance of the chip is compared with the original design that has no body bias (NBB) in Figure 1.16. The maximum operating frequency (F_{max}) of the NBB and FBB router chips are compared from 0.9 V to 1.8 V V_{dd} at 60°C (see Figure 1.17). The FBB chip with forward body bias achieves 1GHz operation at 1.1 V, compared with 1.25 V required for the NBB chip, or 23% less switching power at 1 GHz. The frequency of FBB is 33% higher than NBB at 1.1 V. Area overhead supporting ABB was approximately 2%, while the power overhead was 1%.

RBB was also applied to the same device to reduce leakage. Figure 1.18 plots the leakage current for the worst-case channel length (L_{wc} dashed) and the nominal channel length (L_{nom} dotted) as a function of RBB. The measured full-chip leakage current is within these upper and lower leakage current bounds over a range of RBB values. The optimum RBB value derived from the measured chip for minimum leakage is 500 mV [34]. Higher RBB values cause the junction leakage current to increase and overall leakage power to go up because, as in Mizuno et al. [17], the V_{dd} was not collapsed; however, effectiveness of RBB reduces as channel lengths become smaller or V_t is lowered. Essentially, the V_t-modulation capability by RBB weakens as short-channel effects become worse or body effect diminishes due to lower channel doping.

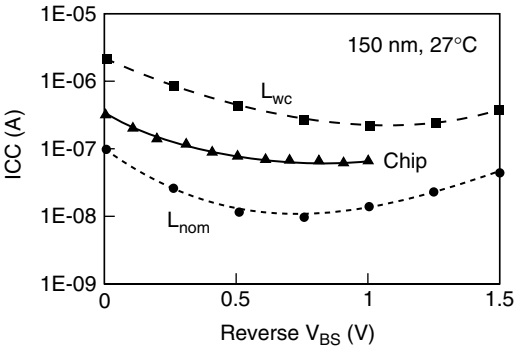


FIGURE 1.18 Leakage reduction by reverse body bias.

1.8.2 Adaptive Body Bias and Supply Bias

The previous two subsections presented the advantages of both FBB and RBB. It is possible to utilize both of these approaches as depicted in Figure 1.19. Due to the frequency spread in fabricated parts caused by process variations, the low frequency parts may be discarded for lower performance and the high frequency parts may be discarded for higher leakage power. As presented on the right side, devices can be adaptively biased to increase the performance of the slow parts by FBB and to decrease leakage power of the fast parts by RBB.

A test chip was implemented in a 150-nm CMOS technology to evaluate effectiveness of the adaptive body bias (ABB) technique for minimizing impacts of both die-to-die and within-die parameter variations on processor frequency and active leakage power [35]. The bias is based on a 5-bit digital code, which provides one of 32 different body bias values with 32 mV resolution to PMOS transistors. NMOS body is biased externally across the chip. Bidirectional ABB is used for both NMOS and PMOS devices to increase the percentage of dies that meet both frequency requirement and leakage constraint. As a result, die-to-die frequency variations (σ/μ) reduce by an order of magnitude, and 100% of the dies become acceptable (see Figure 1.20). Bin 2 is the highest frequency bin, while Bin 1 is the lowest acceptable frequency bin — any dies that are slower than Bin 1 are discarded. Almost 50% of dies with NBB fell below Bin 1 but are recovered using ABB. In addition, 30% of the dies are now in the highest frequency bin allowed by the power density limit. WID-ABB (applying multiple bias values per die to compensate for within-die as well as die-to-die variation) reduces σ of the die frequency distribution by 50%, compared with ABB. In addition, almost all the dies are accepted in the highest possible frequency bin, compared with 30% for ABB. Another technique to increase yield in the high frequency bins, is to apply adaptive V_{dd}. Figure 1.21 presents the advantage of adaptive V_{dd} over fixed V_{dd}. Bin 3 is the highest

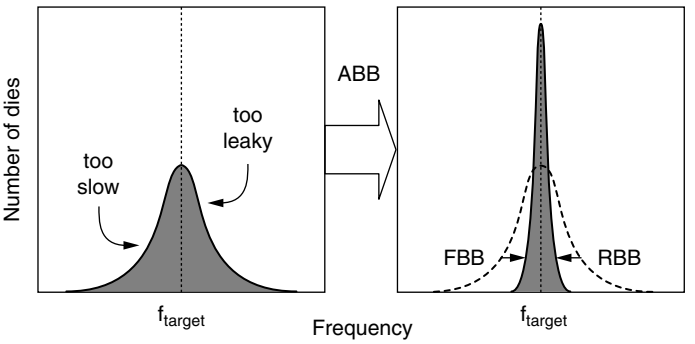


FIGURE 1.19 Target frequency binning by adaptive body bias.

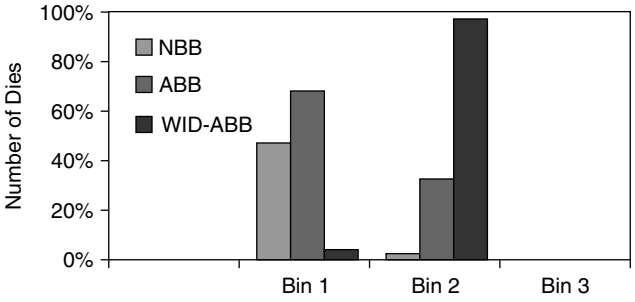


FIGURE 1.20 Adaptive body bias results.

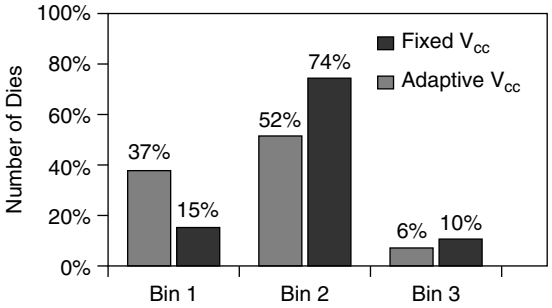


FIGURE 1.21 Bin improvement by adaptive V_{cc}.

frequency bin, while Bin 1 is the lowest acceptable frequency bin. The dark bars indicate that adaptive V_{dd} (V_{cc} in the figure) has pushed more than 20% dies from Bin 1 to Bin 2 and even Bin 3, as well as recovered those dies that fell below Bin 1.

1.9 Dynamic Voltage Scaling

Although adapting the power supply voltage to manufacturing variation was introduced previously, it may also be used to adjust power usage dynamically to the workload at hand. This section describes the dynamic variation of the power supply voltage V_{dd} appropriate to the instantaneous workload of the integrated circuit, commonly described as dynamic voltage scaling (DVS) or dynamic voltage management (DVM) [36]. The results are from system level measurements performed on the 80200 microprocessor. The basic premise is to adjust the frequency and voltage of the device to the lowest values that will simultaneously meet the required application throughput and the operating envelope of the processor. If the performance voltage curve of the device is violated (i.e., a circuit critical path is provided insufficient voltage to meet its timing constraints) then a circuit failure will occur. This implies that changing voltage and clocks must conform to two rules:

1. V_{dd} must be adjusted upward before initiating a frequency increase.
2. Frequency must be lowered before adjusting V_{dd} downward.

The power, voltage, and frequency measured on the processor are plotted in Figure 1.22. The large power range obtainable using DVM is evident, ranging from 6 mA with the clocks gated off to 1.4 W at 1 GHz.

1.9.1 Clock Generation

In conventional designs, the PLL must be allowed to relock to the new frequency when a new frequency is chosen because a divided version of the core clock itself is compared with the reference, as illustrated

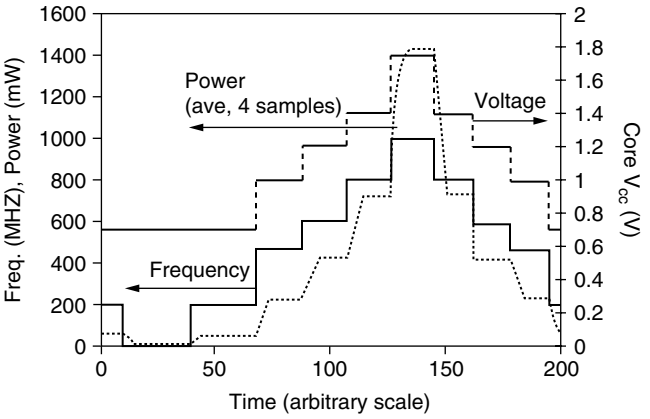


FIGURE 1.22 Frequency, voltage, and power vs. time using DVM.

in Figure 1.23(a). Because the PLL may generate clocks that are shorter than the chosen frequency during this time, clocks to the logic core must be gated off while relocking the PLL. The PLL relock time is predictable and often fixed (by comparing with a counter representing the worst-case lock time — 20 μs previously) to simplify specification and testing. V_{dd} adjustment and initiation of a frequency change may be coincident if the time to change V_{dd} is predictable and consistent (e.g., slew rate limited). In this case, the time to reach the specified voltage is dependent on the starting voltage. This clock change time introduces latency to achieving the lower power. An energy cost also occurs from moving the highly capacitive supply voltage. The latter is unavoidable, but the former can be mitigated in two ways. First, the processor can be supplied with the reference clock during clock changes. This can maintain some, generally lower performance, but allows computation to continue and avoids a "dead zone" where interrupts cannot be taken. Second, a more sophisticated PLL divider scheme allows "on the fly" changes in clock rate.

This approach, illustrated in Figure 1.23(b) keeps the PLL running at a consistent maximum frequency for all voltage and frequency configurations. This requires a separate power supply connection for the PLL, which is virtually required to keep the analog PLL supply isolated from noise. Typically, this supply is separated and is additionally filtered or regulated to improve the clock jitter component due to supply

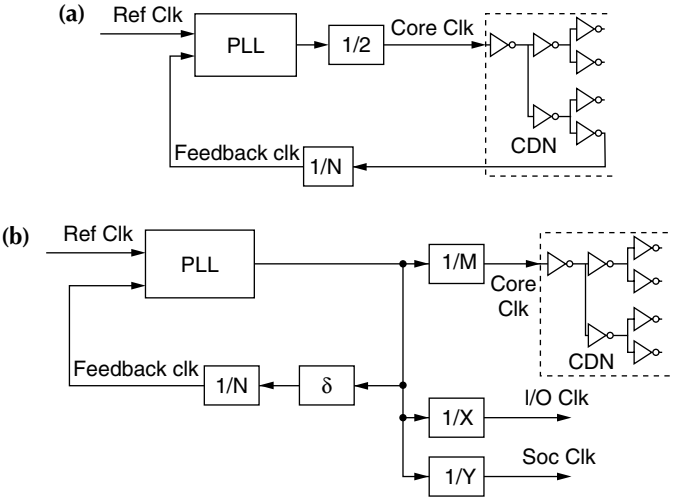


FIGURE 1.23 (a) Conventional PLL and clock generation, and (b) scheme to allow speed changes without performance penalty. The 1/M divider in (b) is dynamically adjustable.

noise. Here, the PLL supply is not scaled with the logic core. The PLL power is not strongly dependent on the VCO frequency and is a small fraction of the overall active power, so the penalty of not scaling the PLL V_{dd} is acceptable. Given the clock performance benefit accrued by regulating the PLL supply, a fixed PLL supply is the preferred approach.

In a conventional design, the core or IO clock is fed back to the phase-frequency detector (PFD) of the PLL as depicted in Figure 1.23(a). Because the point is to lock the internal clock edges to the external reference clock, this feedback is from the end of the clock distribution network (CDN) to include the insertion delay. The only insertion delay to match is that of the feedback divider. To allow on-the-fly clock changes, the clock dividers are configured as depicted in Figure 1.23(b). The feedback divider reduces the VCO frequency to that of the reference clock independent of the core clock divisor chosen. The core clock divisor can then be changed dynamically within certain constraints. First, no clock glitches can be allowed. Second, the clock changes must be predictable to allow consistent behavior when transferring data across domains, as well as for testing and validation. More dividers to other clock domains are likely in a large SoC design, as discussed. It is important to have the same insertion delay for all of the clocks, so that the version returning to the PLL tracks the others. In practice, some latitude in insertion delays can be allowed, which will show up as systematic additions to the off-chip or inter-domain clock skew. Finally, the mechanism for crossing from one domain to another must be independent of the actual frequencies. In practice, this is provided by generating separate signals from the same clock divider circuits, which anticipate coincident edges constituting allowed domain crossings. Ideally, the maximum core speed is half that of the PLL voltage controlled oscillator (VCO) because a 2× divide easily provides a 50:50 duty cycle clock.

1.9.2 Experimental Results

The important factor in implementing dynamic voltage management is the amount of work performed (e.g., the number of instructions required) instead of the number of clocks and some indicator of whether or not the machine is busy. Modern systems use an interrupt-driven model, whereby the processor will enter a low-power state, pending being awakened via interrupt, when there is no useful work to perform. Many operations are available, however, particularly memory accesses and IO, which have significant latency. Increasing core to memory frequency ratios exacerbates this. Consequently, to effectively utilize DVM, it is necessary to detect when the processor is constrained by such operations, which effectively limit the number of instructions per clock (IPC) below the peak value.

As an example, two experiments were performed using an 80200 running a modified Linux operating system (OS) kernel to monitor the work performed two different ways. The first merely determined if the scheduled tasks had been completed early, while the second determined the actual IPC using the on-core performance monitors. The interval between adjustments was 10 ms.

The experiments performed the following at the end of each interval:

1. OS only using time-slice utilization

```
If (task finished early)
Lower the voltage and frequency
Else
Raise the voltage and frequency.
```

2. OS using time slice utilization and core performance monitors (Sampled number of instructions executed and number of data dependencies every 2 ms):

```
If (work performed increases)
Raise voltage and frequency
Else
Lower the voltage and frequency
```

In the first case, whether or not a task completed early provided a coarse assessment of the needed computational power, and it was assumed that the future demands would be similar. In the second case,

the actual work was determined. It can be inferred that the latter approach may also provide a more quantitative estimate of how much the frequency should be raised and lowered. It also allows the power to be lowered, essentially matching the processor clock to memory or system clock ratios more appropriately to a given workload, automatically detecting and minimizing the energy consumption in memory bound cases.

1.10 Conclusions

Higher digital IC power and parameter variations are an inevitable consequence of scaling and promise to increase further in the future. This chapter has described some of the presently important as well as emerging limiting mechanisms. Various design techniques that mitigate these issues were discussed. These techniques rely on leveraging the often-neglected bulk terminal as well as careful selection of the supply voltage to both the specific device as manufactured as well as the computing task at hand. It has been demonstrated that the overhead of using these techniques, although nonnegligible, is modest. As transistor scaling forces future products into increasingly difficult cost, power, and performance trade-offs, we can expect to see greater reliance on these, as well as other design schemes to enable still further scaling.

References

[1] S. Borkar, Obeying Moore's law beyond 0.18 micron, *Proc. 13th Annu. ASIC/SOC Conf.,* pp. 13–16, 2000.

[2] L. Clark et al., An embedded 32b microprocessor core for low-power and high-performance applications, *IEEE J. Solid-State Circuits,* 36, p. 1599, 2001.

[3] K. Chen and C. Hu, Performance and V_{dd} scaling in deep submicrometer CMOS, *JSSC,* 33, pp. 1586–1589, Oct. 1998.

[4] Y. Taur et al., *Fundamentals of Modern VLSI Devices,* Cambridge University Press, U.K., 1998.

[5] A. Keshavarzi, S. Narenda, S. Borkar, C. Hawkins, K. Roy, and V. Dey, Technology scaling behavior of optimum reverse body bias for leakage power reduction in CMOS ICs, *Proc. ISLPED,* pp. 252–254, 1999.

[6] R. Krishnamurthy et al., High-performance and low-power challenges for sub-70-nm microprocessor circuits, *CICC Proc.,* pp. 125–128, 2002.

[7] H. Wong, D. Frank, P. Solomon, H. Wann, and J. Welser, Nanaoscale CMOS, *Proc. IEEE,* 87, pp. 537–570, 1999.

[8] S. Wolf, *Silicon Processing for the VLSI Era: Volume 3 — The Submicron MOSFET,* Lattice Press, Sunset Beach, CA, 1995.

[9] R. Gonzalez, B. Gordon, and M. Horowitz, Supply and threshold voltage scaling for low-power CMOS, *IEEE J. Solid-State Circuits,* 32, pp. 1210–1216, Aug. 1997.

[10] D. Frank, Power constrained CMOS scaling limits, *IBM J. Res. Dev.,* 46, 2/3, p. 235, 2002.

[11] S. Thompson, Technology performance: trends and challenges, IEDM short course, *IEDM '99 Tutorial,* Washington, D.C., 1999.

[12] H. Holma and A. Toskala, Eds., *WDCMA for UMTS: Radio Access for Third-Generation Mobile Communications,* John Wiley & Sons, New York, 2001.

[13] S. Thompson, I. Young, J. Greason, and M. Bohr, Dual threshold voltages and substrate bias: keys to high performance, low-power 0.1-μm logic designs, *VLSI Tech. Symp. Dig.,* pp. 69–70, 1997.

[14] M. Horiguchi, T. Sakata, and K. Itoh, Switched-source-impedance CMOS circuit for low standby subthreshold current giga-scale LSIs, *IEEE JSSC,* 28, pp. 1131–1135, Nov. 1993.

[15] B. Sheu, D. Scharfetter, P. Ko, and M. Jeng, BSIM: berkeley short-channel IGFET model for MOS transistors, *IEEE JSSC,* 22, pp. 558–566, Aug. 1987.

[16] L. Clark, N. Deutscher, S. Demmons, and F. Ricci, Standby power management for a 0.18-μm microprocessor, *Proc. ISLPED,* pp. 7–12, 2002.

[17] H. Mizuno et al., An 18-μA standby current 1.8-V, 200-MHz microprocessor with self-substrate-biased data-retention mode, *IEEE J. Solid-State Circuits,* 34, 1999, p. 1492.
[18] S. Yang et al., A high-performance 180-nm generation logic technology, *Proc. IEDM,* pp. 197–200, 1998.
[19] M. Morrow, Microarchitecture uses a low-power core, *IEEE Computer,* p. 55, April, 2001.
[20] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada, 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS, *IEEE J. Solid-State Circuits,* 30, 1995, pp. 847–854, Aug. 1995.
[21] L. Clark, A high-voltage output buffer fabricated on a 2-V CMOS technology, *VLSI Circuit Symp. Dig.,* pp. 61–62, 1999.
[22] R. Swanson and J. Meindl, Ion-implanted complementary MOS transistors in low-voltage circuits, *IEEE JSSC,* SC-7, pp. 146–153, April 1972.
[23] L. Clark, M. Morrow, and W. Brown, Reverse body bias for low effective standby power, *IEEE Trans. VLSI,* Sept. 2004.
[24] BRH Reference Platform specifications are available at http://www.adiengineering.com/products-BRH.html.
[25] M. Osqui, Evaluation of software energy consumption on microprocessors, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Oct. 2001.
[26] K. Bowman et al., Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration, *IEEE J. Solid-State Circuits,* 37, pp. 183–190, Feb. 2002.
[27] S. Borkar, Parameter variations and impact on circuits and microarchitecture, *C2S2 MARCO Review,* March 2003.
[28] G. Sery et al., Life is CMOS: why chase the life after? *DAC 2002,* pp. 78–83.
[29] T. Karnik et al., Sub-90nm technologies — challenges and opportunities for CAD, *ICCAD 2002,* pp. 203–206.
[30] T. Karnik et al., Total power optimization by simultaneous dual-V_t allocation and device sizing in high performance microprocessors, *DAC 2002,* pp. 486–491.
[31] J. Tschanz et al., Design optimizations of a high-performance microprocessor using combinations of dual-V_t allocation and transistor sizing, *VLSI Circuits Symp. 2001,* pp. 218–219.
[32] J. Tschanz et al., Dynamic-sleep transistor and body bias for active leakage power control of microprocessors, *ISSCC 2003,* pp. 102–103.
[33] S. Narendra et al., 1.1-V 1-GHz communications router with on-chip body bias in 150-nm CMOS, *ISSCC 2002,* pp. 270–271.
[34] A. Keshavarzi et al., Effectiveness of reverse body bias for leakage control in scaled dual V_t CMOS ICs, *ISLPED 2001,* pp. 207–210.
[35] J. Tschanz et al., Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage, *ISSCC 2002,* pp. 422–423.
[36] T. Burd, T. Pering, A. Stratakos, and R. Broderson., A dynamic voltage scaled microprocessor system, *IEEE J. Solid-State Circuits,* 35, pp. 1571–1580, Nov. 2000.

2

Low-Power DSPs

Ingrid Verbauwhede
University of California–Los Angeles

2.1 Introduction

Mobile wireless communications show an incredible growth, as illustrated in Figure 2.1. It is estimated that by the year 2010, wireless phones will surpass wire line phones, each having a worldwide penetration of more than 20%. The market for digital signal processors (DSPs) has a growth rate of 40%. In 1966, it was a $2 billion market, and by 1999 it had grown to a $4.4 billion market. After a dip in 2001–2002, the forecast for 2004 is $7.7 billion and a predicted $17 billion by 2008 [28]. More than 60% of all DSP shipments are used in cellular phones [28]. In the industrialized world, the numbers are even more impressive: in a small country like Belgium with a population of 10 million, more than 2 million cell phones are sold every year compared with approximately 600,000 PCs [6].

Power optimization can be done at several levels of abstraction: technology level, circuit level, gate level, architectural level, algorithm level, and system level. Multiple chapters in this book are devoted at each of these abstraction levels: at the technology level is the usage of multiple threshold voltages, a low V_t for the logic circuits and a high V_t for the memory circuits. At the circuit level, a designer has the choice of using complementary static CMOS instead of high-speed dynamic logic. At the logic level, gated clocks and power down of unused modules will reduce the power consumption. At the architectural level, an optimization of the processor components such as the datapath and the memory architecture will reduce power. At the system level, the selection of variable voltages, idle and sleep modes, etc. will contribute to the reduction of power.

The focus of this chapter is on the power and energy reduction because of optimizations at the architectural and micro-architectural level. Indeed, by tuning the processor components to the application field, a huge amount of power can be saved. This means *all* processor components, and includes the datapaths, the memory architecture, the bus network, and the control architectures, which includes instruction set design.

The first successful DSP processors were introduced in the early 1980s. Many good overview papers are available that describe the evolution of these processors and the special features to support signal processing applications [10,16,17]. Examples in this category are the Texas Instruments TMS320C1x, C2x, C5x, series or the Lucent DSP16A and DSP1600 series. This chapter focuses on the evolution of

DSP processors during the last couple of years, especially the special features in the processors to support the demands from wireless communications.

Until recently, the same DSP processors are used both in the mobile terminal (i.e., the actual cell phone) and in the base stations; however, a trend starts to emerge to place different processors in the mobile terminal and the base stations. The main drivers for the processors in the mobile are cost and very low energy consumption. This leads to processors that have a very compact but complex instruction set (CISC) and work with domain-specific or application-specific coprocessors. High-performance processors need to be included in the base stations, and they tend to become more compiler-friendly because the software complexity requires it. Thus, the success of very large instruction word (VLIW) processors and modified VLIW processors for the base station applications.

It is insightful to first define the meaning of million instructions per second (MIPS) and million operations per second (MOPS). Most traditional DSP processors belong to the class of CISC processors. This means that in 1 instruction, typically 16 bits wide, several operations, sources, and destinations for the operations are coded. For instance, in one dual-multiply-accumulate instruction of the Lode processor, six different operations are performed: two memory-read operations, two address calculations, and two multiply-accumulate (MAC) operations [30]. Assuming the processor runs at 100 MHz, this corresponds to 100 MIPS and 600 MOPS. If the multiply and adds are considered two operations, this becomes 800 MOPS. Similarly, in one dual MAC instruction on the Lucent 16210, seven different operations are executed: one three-input addition, two multiplications, two memory reads, and two address pointer updates. This corresponds to 700 MOPS.

CISC type processors are usually compared on the amount of MIPS. Sometimes, to make things confusing, the two multiply-accumulate operations are counted separately (usually by marketing or sales people). Therefore, it might be a 100-MHz processor, advertised for "200 MIPS."

One instruction of a VLIW processor consists of a set of small (e.g., six or eight), primitive instructions, issued in parallel. It is customary to multiply the clock frequency of these processors by the number of parallel units and define these as MIPS or MOPS. The processor described in Weiss et al. [33] uses a VLIW variation combined with SIMD properties, to reach 3000 MOPS with a 100 MHz clock. The processor in Igura et al. [14] runs at 50 MHz and is described as an 800-MOPS solution.

To make a fair comparison between processors, we will use the MIPS terminology when referring to the clock frequency and count the primitive operations for both the CISC and VLIW machines as MOPS.

A second insight is the means of measuring performance of DSP processors. Instead of comparing processors based on GHz or MOPS, DSP processors are usually compared on the number of instructions to get the job done. Therefore, the goal is to minimize the number of instructions, also expressed in MIPS (to make it even more confusing). For instance, the MIPS for several speech coding standards on a SH-DSP are reported in Baji et al. [5]: the simplest full-rate GSM speech codec requires 3.1 MIPS. A half-rate coder already requires 23 MIPS.

Section 2.2 introduces the driving application — in this case, wireless mobile communications. Section 2.3 identifies the most important computation-intensive functions, and gives the DSP approach for a low-power solution. Section 2.4 discusses the integration of DSP processors and coprocessors in systems on chip (SoCs). Conclusions are formulated in Section 2.5.

2.2 The Application Driver

DSP processors are made to support hard real-time signal processing applications. This translates in the rule that 10% of the code is executed 90% of the time, and 90% of the code is executed 10% of the time. The code that is executed all the time tends to sit in tight loops, of which every instruction or clock cycle counts. DSP processors are compared based on the number of instructions and the number of clock cycles it takes to execute basic DSP kernels.

The main building blocks of a wireless terminal are depicted in Figure 2.1. The computation-intensive functions can be subdivided in two main categories. The first is associated with the communication

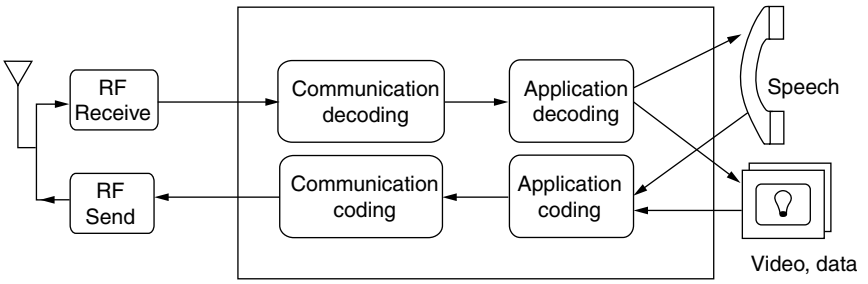


FIGURE 2.1 Application overview.

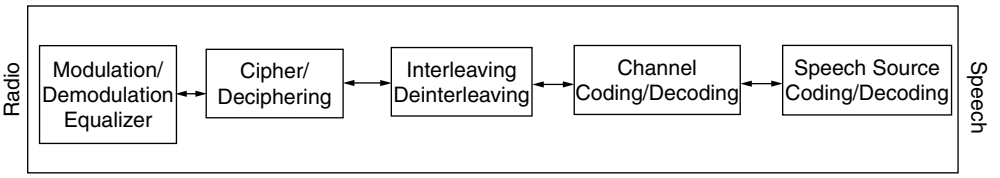


FIGURE 2.2 DSP functions of a second-generation communication system.

processing, also called baseband processing. The second is associated with the application processing, also called source coding. The main baseband building blocks of second-generation cellular phones, such as for GSM, GSM+, and IS-95, are illustrated in Figure 2.2 [11,12,23]. About half of the processing functions are at the physical layer, implementing the modulation/demodulation, the equalizer, and the channel coder and decoder. The other half of the processing occurs at the application level. For second-generation phones, this means the speech coder.

All functions of the system in Figure 2.2 can be implemented in one state-of-the-art DSP processor running at a clock frequency between 80 MHz to 150 MHz. The differentiation between the processors and implementations sits in either the power consumption or the extra features that are included in the processor, such as noise cancellation or equalizers that are more advanced.

Third-generation (3G) cellular wireless standards put higher demands on the modem functions as well as the application functions. 3G systems support not only speech, but also data, image, and video communication. These advanced applications require more processing power from the DSP. At the same time, the advanced applications put higher demands on the coding algorithms, requesting improved bit-error-rates. Thus, equalizers that are more advanced as well as coding algorithms that are more advanced, such as turbo coding algorithms, are used. This is combined with a higher bandwidth requirement.

The blocks with the largest computational requirements are the following:

- Filters (FIR, IIR), autocorrelations, and other "traditional" signal processing functions.
- Convolutional decoders based on the Viterbi algorithm.
- To support data processing requirements in 3G systems, turbo coders are introduced.
- On the application side, efficient codebook search and max–min search, etc. for speech coders and vector search algorithms are required.
- Image and video decoding is the next highly computation-intensive function requiring efficient implementation. Major examples are JPEG and MPEG.

The next subsections discuss how different DSP processors have special architecture features to support the most commonly required computational building blocks. Some of these features are tightly coupled to the DSP processor architecture and integrated in the instruction set. We call these tightly coupled coprocessor units. Some features run on separate building blocks through a bus or memory mapped interface. In this case, jobs are delegated to the coprocessors. We call these loosely coupled coprocessors.

2.3 Computation-Intensive Functions and DSP Solutions

Power consumption in CMOS circuits is mainly dynamic switching power (assuming that the leakage power is well under control, which is a separate topic). Thus, the goal is to avoid unnecessary switching in the processor and limiting the switching to actions necessary to create the outcome of the algorithm. As an example, a multiplication and addition operation is fundamental to calculate an FIR filter, assuming that the multiplication can be done without glitching power; however, the instruction read, decode, and memory accesses can be considered as "overhead." This overhead is not present in a full custom application specific integrated circuit (ASIC) that only performs an FIR filter.

A processor has four fundamental components: datapaths to calculate the algorithm and three supporting building blocks, including control (e.g., all the instruction read/decode logic), storage, and interconnect. To reduce power in a DSP processor, one should look at the supporting processor blocks and reduce or tune them toward the application domain. This will reduce the unnecessary overhead power.

This concept is illustrated in the next section, with several computationally intensive functions running on DSP processors.

2.3.1 FIR Implementation

The basic equation for an N tap FIR equation is the following:

$$y(n) = \sum_{i=0}^{i=N-1} c(i) \cdot x(n-i)$$

When this equation is executed in software, output samples $y(n)$ are computed in sequence. This means that to compute one output sample, there are N multiply–accumulate operations and 2N memory read operations to fetch the data and the coefficients. N is the number of taps in the filter. It is well-known that DSP processors include datapaths to execute multiply accumulate operations in an efficient way [17]. Therefore, we focus on the memory architecture, which is a much more fundamental design issue for DSP processors.

2.3.1.1 Memory Architectures

On a traditional von Neumann architecture, 3N access cycles are needed to compute one output: for every tap one needs to fetch one instruction, read one coefficient, and read one data sample sequentially from the unified memory space. Already early on, DSP processors were differentiated from von Neumann architectures because they implemented a Harvard or modified-Harvard architecture [16,17]. The main characteristic is the use of two memory banks instead of one common memory space in the von Neumann architecture. The Harvard architecture has a separate data memory from program memory. This reduces the number of sequential access cycles from three to two because the instruction fetch from the program memory can be done in parallel with one of the data fetches. The modified Harvard architecture improves this even further. It is combined with a "repeat" instruction. In this case, one multiply–accumulate instruction is fetched from program memory and kept in the one instruction deep instruction cache. Then, the data access cycles are performed in parallel: the coefficient is fetched from the program memory in parallel with the data sample being fetched from data memory. This architecture is found in all early DSP processors and is the foundation for all following DSP architectures. It is an illustration of the "tuning" of the processor components to the application, in this case the memory architecture and the control logic.

The newer generation of DSP processors has even more memory banks, accompanying address generation units and control hardware, such as the repeat instruction, to support multiple parallel accesses. The execution of a 32-tap FIR filter on the dual MAC architecture of the Lucent DSP 16210 is depicted in Figure 2.3. The corresponding pseudo code is the following:

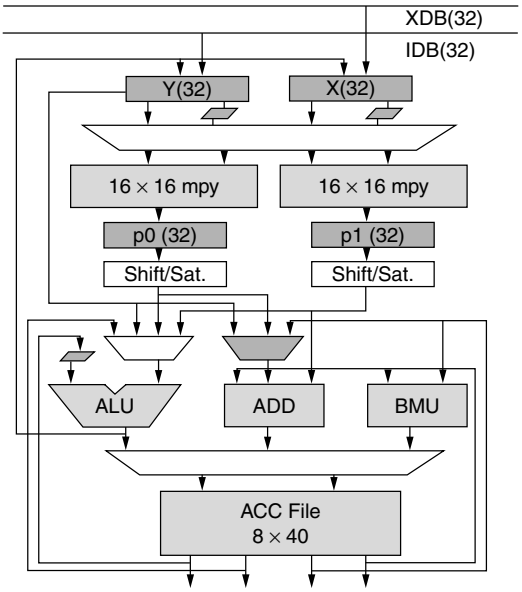


FIGURE 2.3 Lucent/Agere DSP16210 architecture.

```
do 14 {//one instruction !
a0=a0+p0+p1
p0=xh*yh p1=xl*yl
y=*r0++ x=*pt0++
}
```

This code can be executed in 19 clock cycles with only 38 bytes of instruction code. The inner loop takes one cycle to execute and as can be seen from the assembly code, seven operations are executed in parallel: one addition with three inputs, two multiplications, two memory reads, and two address pointer updates.

The difficult part in the implementation of this tight loop is the arrangement of the data samples in memory. To supply the parallel datapaths, two 32-bit data items are read from memory and stored in the X and Y register, as illustrated in Figure 2.3. Then, the data items are split in an upper half and a lower half and supplied to the two 16×16 multipliers in parallel. It requires a correct alignment of the data samples in memory, which is usually tedious work done by the programmer because compilers are not able to handle this. A similar problem exists in single instruction multiple data (SIMD) instructions on general-purpose microprocessors. If the complete word length of the memory locations is used, it requires a large effort from the programmer to align the smaller subwords (e.g., at the byte level) into larger words (e.g., 32-bit integers). A similar data alignment approach is used in Kabuo et al. [15]. Instead of two multipliers, only one multiplier working at double the frequency is used, but the problem of alignment of data items in memory remains. This approach will not reduce the total amount of bits read from memory; only the number of instructions (control overhead) is reduced.

To reduce the amount of data read from memory, more local reuse of the data items is needed. This is illustrated with the Lode architecture [30]. In this example, a delay register is introduced between the two MAC units as illustrated in Figure 2.4. This halves the amount of memory accesses. Two output samples are calculated in parallel as indicated in the pseudo code of Table 2.1. One data bus will read the coefficient from memory; the other data bus will read the data sample from memory. The first MAC will compute a multiply-accumulate for output sample $y(n)$. The second multiply–accumulate will compute in parallel on $y(n + 1)$. It will use a delayed value of the input sample. In this way, two output samples are computed at the same time.

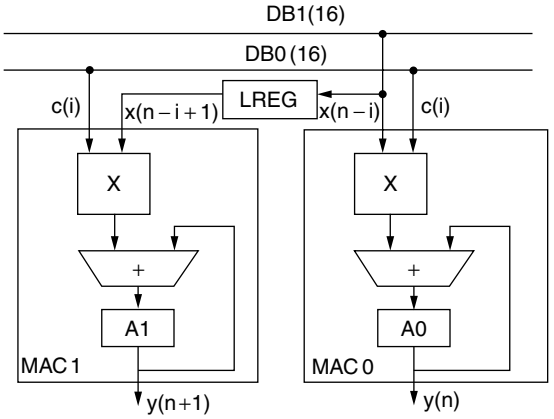


FIGURE 2.4 Lode's dual MAC architecture with delay register.

TABLE 2.1 Pseudo Code for FIR Implementation

```
y(0) = c(0)×(0) + c(1) ×(-1) + c(2) ×(2) + … + c(N-1) ×(1-N):
y(1) = c(0) ×(1) + c(1) ×(0) + c(2) ×(-1) + … + c(N-1) ×2-N):
y(2) = c(0) ×(2) + c(1) ×(1) + c(2) ×(0) + … + c(N-1) ×(3-N):
…
y(n) = c(0) ×(n) + c(1) ×(n-1) + c(2) ×(n-2) + … + c(N-1) ×(n-(N-1)):
```

This concept of inserting a delay register can be generalized. When the datapath has P MAC units, P-1 delay registers can be inserted and only $2N/(P + 1)$ memory accesses are needed. These delay registers are pipeline registers and thus if more delay registers are used, more initialization and termination cycles need to be introduced.

The TI TMS320C55x [24] is a processor with a dual MAC architecture and three 16-bit data busses. To supply both MACs with coefficients and data samples, the same principle of computing two output samples at the same time is used. One data bus will carry the coefficient and supply this to both MACs, the other two data busses will carry two different data samples and supply this to the two different MACs [3]. Figure 2.5 illustrates this.

Table 2.2 summarizes the different implementations. Note that most energy savings are first obtained from reducing the amount of memory accesses and, second, from reducing the number of instruction cycles. Both are considered overhead. Indeed, the total energy associated with the MAC operations is fixed because an N tap FIR filter requires N multiply-accumulate operations. A dual MAC computes two

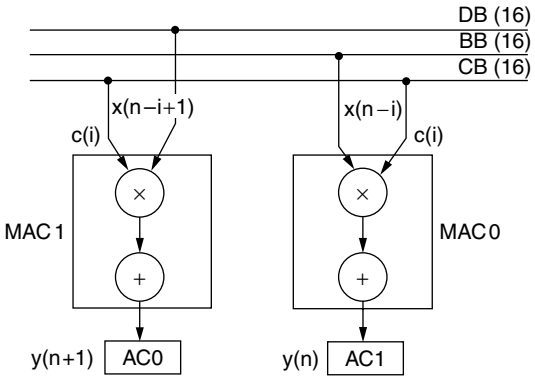


FIGURE 2.5 Dual MAC with three data buses.

TABLE 2.2 Energy Evaluation for an N Tap FIR Filter

DSP	Data Memory Access	MAC Operations	Instruction Cycles	Instructions
Von Neumann	2N	N	3N	2N
Harvard	2N	N	2N	2N
Modified Harvard	2N	N	N	2 (repeat instruction)
Dual Mac	2N	N	N/2	2 (same)
Dual Mac with 3 data busses	1.5N	N	N/2	2
Dual Mac with 1 delay reg	N	N	N/2	2 (same)
Dual Mac with P delay reg	2N/(P + 1)	N	N/(P+1)	2

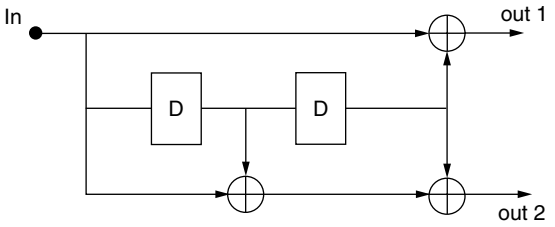


FIGURE 2.6 Example convolutional coder.

MAC operations in parallel and thus the instantaneous power could even be higher in this case. Of importance for battery-operated handsets is, however, the total energy drawn from the supply.

2.3.2 Viterbi Acceleration

The Viterbi decoders are used as forward error correction (FEC) devices in many digital communication devices, not only in cellular phones but also in digital modems and many consumer appliances that require a wireless link. The Viterbi algorithm is a dynamic programming technique to find the most likely sequence of transitions that a convolutional encoder has generated.

Most practical convolutional encoders are rate 1/n (which means that one input bit generates n coded output bits). A convolutional encoder of "constraint length K" can be represented as a finite state machine (FSM) with K-1 memory bits. This means that the FSM has 2^{K-1} possible states, also called trellis states. If the input is binary, there are two possible next states starting from a current state because the next state is computed from the current state and the input bit. This is illustrated in Figure 2.6 with a simple example of a coder with constraint length K = 3, number of states 4. The generator function is $G(D) = [1 + D^2\ 1 + D + D^2]$.

The task of the Viterbi decoding algorithm is to reconstruct the most likely sequence of state transitions based on the received bit sequence. This approach is called the "most likelihood sequence estimation." To compute this most likely path, a trellis diagram is constructed, as illustrated in Figure 2.7. It will compute from every current state, the likelihood of transitioning to one out of two next states.

This leads to the kernel of the Viterbi algorithm, called the Viterbi butterfly. From two current states, two next states are reached. The basic equations executed in this butterfly are:

$$d(2i) = \min\{d(i) + a, d(i + s/2) - a\}$$

$$d(2i+1) = \min\{d(i) - a, d(i + s/2) + a\}$$

2.3.2.1 Memory Architecture

For power and performance efficiency, DSP processors will include special logic for an efficient implementation of these two equations, mostly called an "add-compare-select" (ACS) operation. One needs

Information Data	0	1	1	0	1	0	0
Convolution Codes	00	11	10	10	00	01	11
Error Sequence	00	01	10	00	00	10	00
Received Data	00	10	00	10	00	11	11

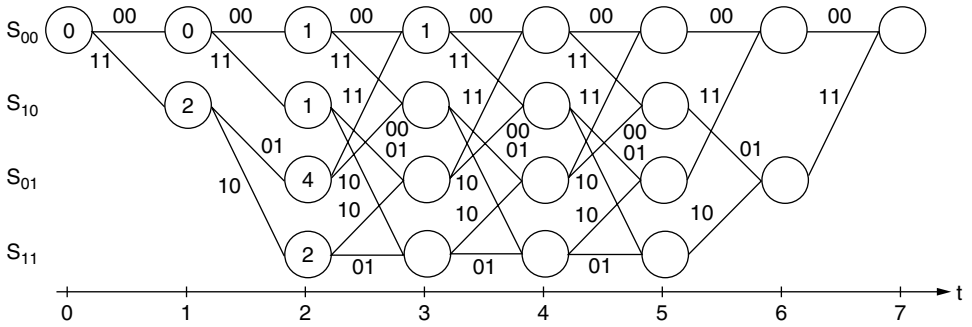


FIGURE 2.7 Example Viterbi trellis diagram.

to add or subtract the branch metric from states i and $i + s/2$, compare them and select the minimum. Similarly, state $2i + 1$ is updated. The first main power reduction comes from the butterfly arrangement because it reduces the amount of memory accesses by half; however, it does slightly complicate the address arithmetic.

2.3.2.2 Datapath Architecture

DSP processors have special hardware and instructions to implement the ACS operation in the most efficient way. The Lode architecture uses the two MAC units and the ALU to implement the ACS operation, as dipicted in Figure 2.6. The dual MAC operates as a dual add/subtract unit. The ALU finds the minimum. The shortest distance is saved to memory and the path indicator (i.e., the decision bit is saved in a special shift register A2). This results in 4 cycles per butterfly [30].

The Texas Instruments TMS320C54x and the Matsushita processor described in Okamoto et al. [20] use a different approach, which also results in four cycles per butterfly. Figure 2.8(b) illustrates this. The ALU and the accumulator are split into two halves (much like SIMD instructions), and the two halves operate independently. A special compare, select, and store unit (CSSU) compares the two halves, selects the chosen one, and writes the decision bit into a special register TRN. The processor described in Okamoto et al. [20] describes two ACS units in parallel. To illustrate the importance of an efficient implementation of the ACS butterflies, consider the IS-95 cellular standard. The IS-95 standard uses a rate 1/2 convolutional encoder with a constraint length of 9 [23], which corresponds to 256 states or 128 butterflies. It has a window size of 192 samples. This corresponds to 128 $\times 192 \times (ACS)$ operations. The most efficient implementation requires four cycles per butterfly. This still corresponds to close to 100 MIPS. One should note that without these specialized instructions and hardware, one butterfly requires 15 to 25 or more instructions, which results in a factor 5 to 10 increase in number of instructions to calculate a complete Trellis diagram.

2.3.2.3 Datapath Support

The hardware support for the Viterbi algorithm on the 16210 also allows for the automatic storage of decision bits from the ACS computations. This functionality can be switched on or off. When the built-in comparison function `cmp1()` is called, the associated decision bit is shifted into the auxiliary register `ar0`. This auxiliary register is a special shift register to move decision bits in at the LSB side. During the trace back phase, its bits are used to reconstruct the most likely path. Each ACS takes two cycles (one

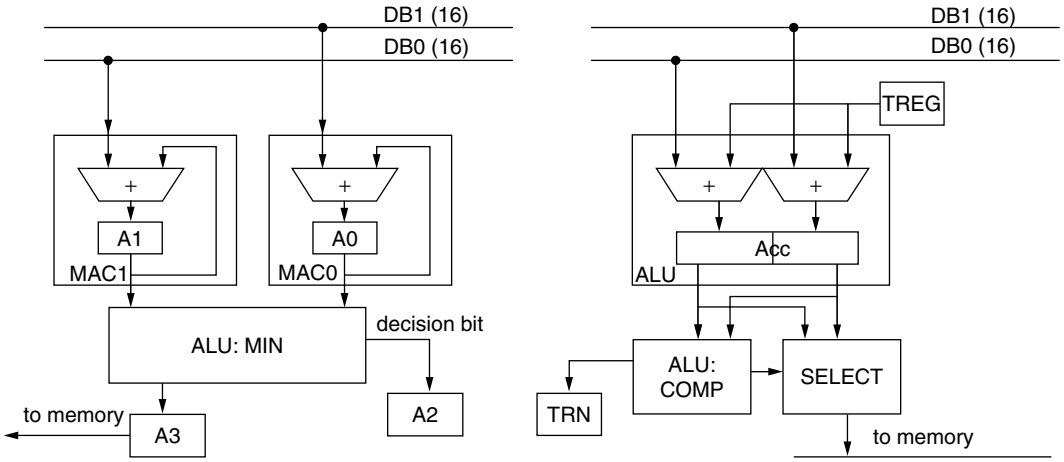


FIGURE 2.8 Add-compare-select (a) on the Lode architecture and (b) on C54x architecture and on the architecture of Okamoto and coworkers [20].

TABLE 2.3 Pseudo Code for the Viterbi Butterfly on the DSP 16210

```
do 8 {
      a0=a4+y a1=a5=y *r3++=a0h
      a2=a4y  a3=a5+y *r5++=a2h
      a0=cmpl (a1, a0) yh=*r0 r0=r1+j j=k k=* pt1++
      a2=cmpl (a3, a2) a4 5h=*pt0++
}
*r2++=ar0
```

for the additions, one for the compare/select), and thus a single butterfly takes a total of four cycles. The code segment in Table 2.3 performs the butterfly computations.

2.3.2.4 Control Architecture

The 16210 has hardware looping support, and there is only a single cycle required to initialize this looping support before the loop executes with zero overhead. When decoding a standard GSM voice channel, which has a constraint length of 5 or 16 states in the trellis, the `ar0` register is filled with 16 decision bits after the 8 butterflies are processed. Thus, with a single memory access, the decision bits can be stored in memory and the next symbol pair can be processed. This is an efficient use of memory bandwidth. For codes with higher constraint lengths and thus more states, the code segment can be executed multiple times with each decision bit word written to memory as required.

2.3.3 Turbo Decoding

Although convolutional decoding remains a top priority (the decoding requirement for EDGE has been identified as greater than 500 MIPS), the performance needed for turbo decoding is an order of magnitude greater. We therefore describe the turbo decoders needed in 3G systems. Turbo decoding (see Figure 2.9) is a collaborative structure of soft-input/soft-output (SISO) decoders with the inclusion of interleaver memories between decoders to scatter burst errors [7]. Either soft-output Viterbi algorithm (SOVA) [13] or maximum a posteriori (MAP) [4] can be used as SISO decoders. Within a turbo decoder, the two

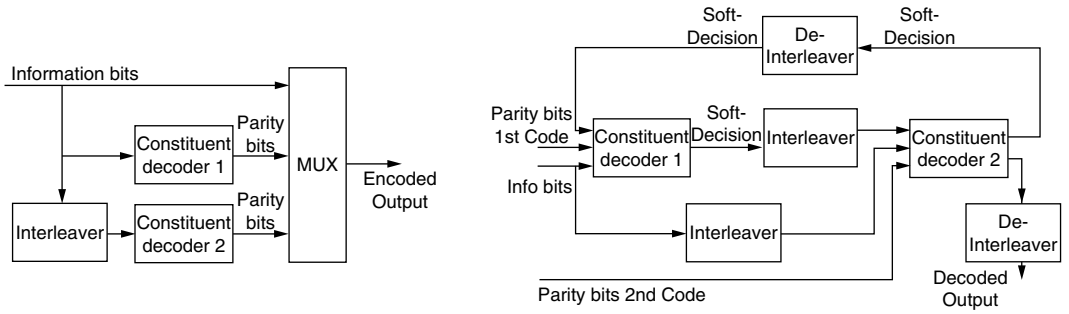


Figure 2.9 Turbo encoder and Turbo decoder.

decoders can operate on the same or different codes. Turbo codes are included to provide coding performance to within 0.7 dB of the Shannon limit (after a number of iterations).

The Log-MAP algorithm can be implemented in a manner very similar to the standard Viterbi algorithm. Perhaps the most important difference between the algorithms when they are implemented is the use of a correction factor on the new "path metric" value (the alpha, beta, and log-likelihood ratio values in Log-MAP) from each ACS, which is dependent on the difference between the values being compared. This is typically implemented using a lookup table, with the absolute value of the difference used as an index into this table and the resulting value added to the selected maximum before it is stored.

2.3.3.1 Datapath Architecture

The C55x DSP processor includes explicit instructions to support the turbo decoding process. This is illustrated in Figure 2.10. A new instruction, the *max_diff* (ACx, ACy, ACz, ACw) is introduced [24]. It makes use of the same ALU and CSSU unit as the Viterbi instructions. Again, the ALU is split into two 16-bit halves. This processor has four accumulator registers compared with two in the previous generation. All four accumulator registers are split in half. The two differences, between ACx(H) and ACy(H) and between ACx(L) and ACy(L), are stored in the ACw halves. The maximum of the ACx(H) and ACy(H) is stored in ACz(H); the maximum of the ACx(L) and ACy(L) is stored in ACz(L). Two special registers are used to store the path indicators, TRN0 and TRN1.

The preceding modifications support the requirements for wireless baseband processing. To also improve the performance for multimedia, a tightly coupled mechanism of instruction extension and

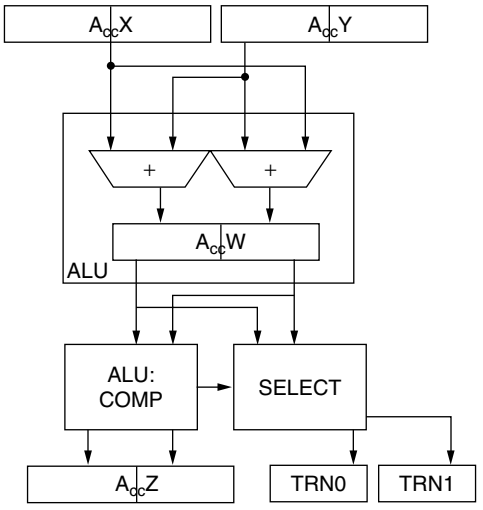


FIGURE 2.10 Turbo decoding acceleration on the C55x.

TABLE 2.4 Examples of Low-Power Programmable DSP Processors

Reference	MOPS	Technology	Threshold Voltages	Power	Standby-Power
Mutoh [19]	26	0.5 μm	Two	2.2 mW/MHz (at 1 V, 13.2 MHz)	350 μW
Lee [18]	300	0.25 μm	Two	0.21 mW/MHz (at 1 V, 63 MHz)	4.0 mW
Shiota [26]	NA	0.25 μm	Two	0.26 mW/MHz (at 1 V max, 50 MHz)	100 μW
Igura [14]	800	0.25 μm	One	2.2 mW/MHz (at 1.5 V, 50 MHz)	NA
Zhang [35]	240	0.25 μm	One	0.05 mW/MHz (at 1 V, 40 MHz)	NA

Note: NA = not available.

hardware acceleration is added to the C55x processor [9]. A special set of instructions is provided, with sources and destinations that are shared with the regular instructions. These special instructions have one set of opcodes: `copr()`. This avoids explosion of the instruction code size. Then, the application specific integrated processor (ASIP) designer has the choice to define the functionality of the hardware accelerator units addressed by these `copr()` instructions. Typical examples are video processing extensions for mobile applications [22].

Table 2.4 summarizes several low-power DSP processors. Notice that all operate with dual threshold voltages. In addition, note that although the clock frequency is not spectacularly high, the MOPS efficiency is very high for each of these processors.

2.4 DSPs as Part of SoCs

The previous section presented several modifications to the processor architecture to optimize the architecture toward the application domain of mobile wireless communications. It included examples of modifications to the memory architecture, datapath architecture, control architecture, instruction set, and bus architecture. So far, these modifications are tightly coupled (i.e., it reflects directly in the instruction set). The optimized instruction sets result in very compact program sizes and very efficient code. Yet, it is also very hard to produce efficient embedded software. The specialized CISC type instructions are extremely hard to be recognized by the compiler. Thus, the approach usually results in hand-optimized assembly code libraries for the computation-intensive functions.

In addition, the demands of next generation mobile applications are not satisfied by these instruction set modifications alone. More applications and multiple applications in parallel (and on demand) are running on the battery-operated devices.

Because of this, we see two distinct trends: one is in the direction of more powerful, but also more energy-hungry, processors used in the infrastructure. The other trend is in the direction of ultra-low power DSP solutions used in the handheld, battery operated terminals. Processors used in the basestation infrastructure are more compiler-friendly [1]. One popular type is the class of VLIW processors that are developed for wireless communications. Some examples are the TIC6x processor [2], the Lucent/Motorola Starcore [27], the ADI TigherSharc [21]. The main advantage of these processors is that they are compiler-friendly: efficient compiler techniques are available. The main disadvantage is that the program size is large and thus creates a large memory overhead [31]. This makes them mostly attractive to base station infrastructure.

It is interesting to note that some CISC features have reappeared: specialized instructions or loosely coupled coprocessors been added to the base VLIW architecture to improve the performance and to reduce the power consumption. A first example is the TIC6x processor, to which loosely coupled Viterbi and turbo coding coprocessor units are added [2]. A second example is the Starcore processor to which

some specialized instructions are added [21]. Because the main driver is base station infrastructure (i.e., the baseband part of the application), there is no explicit support for the application side of the system, such as speech or video processing.

Because one processor is not sufficient to process the multiple and widely varying applications, a second, more energy efficient trend, is the addition of specialized coprocessors or accelerator units to the main processor on the SoC. This can take several forms. Initially, an SoC had multiple but almost identical processor units, as in Igura et al. [14]. This processor contains four identical DSP processors. Global tasks (coarse-grain) are assigned to the DSP processors in a static manner. Alignment of the tasks is provided by synchronization routines and interrupts. It is demonstrated that a video codec (H.263) and a speech codec (G.723) can run at the same time within the 110-mW power budget. Memory accesses (internal, shared, and external) consume half of the power budget, which indicates again that the memory architecture and the match of the application to the memory architecture is crucial.

A second form is an SoC with heterogeneous processor units. An example of this is the OMAP architecture [22]. It consists of a specialized DSP processor, the TMS320C55x DSP and a microcontroller, an ARM9xx CPU. The microcontroller is used for the control flow, including running an operating system, user interfaces, and so on. The DSP is used for the number crunching signal processing tasks. As discussed before, it is highly optimized for the communication signal processing, and through its extension possibilities toward multimedia applications. Thus, the OMAP is a result of several strategies: domain specific instruction sets, tightly coupled instruction set acceleration through coprocessor instructions, loosely coupled coprocessors, and multiple processors on one SoC. The global flow of data as well as the corresponding interconnect architecture and memory architecture are still fixed.

This leads to a third form. To combine flexibility with energy efficiency, it is our opinion that the SoC architecture should consist of multiple heterogeneous building blocks connected together by a reconfigurable interconnect architecture. We call this a RINGS (reconfigurable interconnect for next generation systems) architecture [32], illustrated in Figure 2.11. Each of the building blocks is optimized for its specific application domain, represented by an application domain pyramid. Within an application pyramid, the reconfiguration or reprogrammability level can be determined individually. For instance,

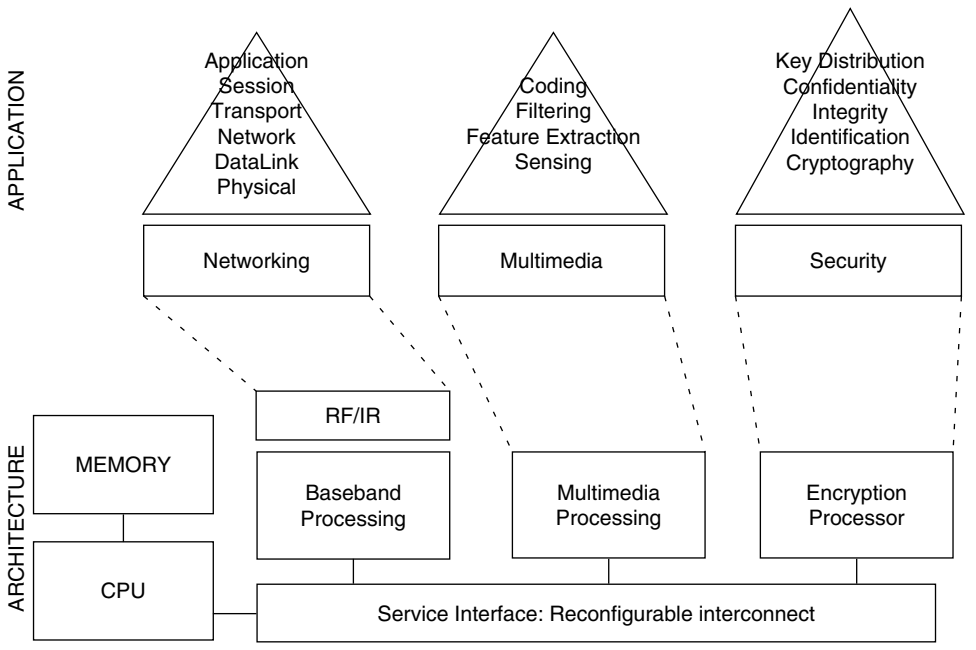


FIGURE 2.11 Generic RINGS architecture.

a baseband pyramid can be realized with a programmable DSP processor, augmented with a few coprocessors. A security pyramid can be realized with a small FSM and lowly programmable crypto acceleration engines. A network protocol stack will need a highly programmable central processing unit (CPU) approach. Multimedia applications are probably best served by a dataflow approach and a chain of hardware acceleration units, and so on. Thus, the transit line between hardware and software can be positioned at different levels in different pyramids. The top level is a general system application (in software) that connects the different pyramids together.

At the bottom, the communication is provided by means of a flexible interconnect. Reconfiguration of the interconnect is also crucial for the MAIA processor [35]. This processor contains an ARM microcontroller and hardware acceleration units (e.g., MAC, ALU, and AGU). The ARM core controls and decides the reconfiguration of the interconnect. To optimize the energy flexibility trade-off a two-level hierarchical mesh network is chosen. A local mesh connects local tightly coupled units. A global mesh with a larger granularity is provided at the top level. This global mesh has switchboxes that connect both globally and downwards to the local level. Another example is the DM310 digital media processor [29]. To obtain low power, it has dedicated coprocessors for image processing, a programmable DSP processor for audio processing, and an ARM processor to process system level tasks.

At the physical level, this is a typical example of a time and space division-based interconnect. To improve density, combined with a larger degree of programability and energy efficiency, we propose to use frequency and code division access to the interconnect medium [32]. This can be combined with the space and time division. From the programmer's viewpoint, the actual physical implementation should be hidden and a programming model should be available, that allows to model different interconnect paradigms and gives the uses the possibility to perform the energy flexibility trade-offs.

A RINGS architecture allows the platform to be changed as the target changes. This approach has been proven successful for an embedded fingerprint authentication system [25]. We are currently working on applying the same design methodology to accelerate multimedia applications for wireless embedded systems.

2.5 Conclusion and Future Trends

Low power can only be obtained by tuning the architecture platform to the application domain. This chapter presents multiple examples to illustrate this for the domain of signal processing and, more specifically, to support the signal processing algorithms for wireless communicating devices. At the same time, demand for flexibility is increasing. Thus, the designer must try to balance these conflicting requirements by providing flexibility at the right level of granularity and to the right components. It is extremely important to realize that this tuning involves *all* components of a processor: the datapaths, the instruction set, the interconnect, and the memory strategy. Traditional DSPs are CISC machines with an adapted modified Harvard interconnect and memory architecture (coming in many flavors). With increasing demands, coprocessors are added to these architectures. As SoCs grow in complexity, however, the architecture becomes one where one integrated device will contain multiple heterogeneous processors. Each processor supports an application domain and its programmability is tuned to the domain. The different components are connected together by a reconfigurable interconnect paradigm. This reconfigurable interconnect poses several research challenges are different abstraction levels: physical realization, the modeling at a higher abstraction level and the reconfigurable programming at compile time and run time.

2.6 Acknowledgments

The author acknowledges the following DSP processor experts: Chris Nicol, Dave Garrett, Wanda Gass, Mihran Touriguian, and Katsuhiko Ueda. The author also acknowledges the contributions of Frank M.C. Chang and Patrick Schaumont.

References

[1] B. Ackland and P. D'Arcy, A new generation of DSP architectures, *Proc. IEEE CICC '99,* Paper 25.1, pp. 531–536, May 1999.

[2] S. Agarwala et al. A 600-MHz VLIW DSP, *IEEE J. Solid-State Circuits,* Vol. 37, No. 11, pp. 1532–1544, Nov. 2002.

[3] D. Alter, Efficient implementation of real-valued FIR filters on the TMS320C55x DSP, Application Report SPRA655, April 2000, available from www.ti.com.

[4] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory,* Vol. IT-20, pp. 284–287, Mar. 1974.

[5] T. Baji, H. Takeyama, and T. Nakagawa, Embedded-DSP superH family and its applications, *Hitachi Review,* Vol. 47, No. 4, pp. 121–127, 1998.

[6] Belgen kopen opnieuw meer gsm's/Belgians buy again more cells phones, *De Tijd,* Sept. 17, 2003.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, Near Shannon limit error-correcting coding and decoding: turbo-codes (1), *Proc. ICC '93,* Vol. 2, pp. 1064–1070, May 1993.

[8] M. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. Davis, G. Woodward, C. Nicol, and R.-H. Yang, A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-mm CMOS, *IEEE J. Solid-State Circuits,* Vol. 37, No. 11, pp. 1555–1564, Nov. 2002.

[9] J. Chaoui, K. Cyr, S. de Gregorio, J.-P. Giacalone, J. Webb, and Y. Masse, Open multimedia application platform: enabling multimedia applications in third-generation wireless terminals through a combined RISC/DSP architecture, 2001. *Proc. (ICASSP '01). 2001 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Volume: 2,* May 7–11, 2001.

[10] W. Gass and D. Bartley, Programmable DSPs, in *Digital Signal Processing for Multimedia Systems,* chap. 9, Marcel Dekker Inc., 1999.

[11] A. Gatherer, T. Stelzler, M. McMahan, and E. Auslander, DSP-based architectures for mobile communications: past, present, and future, *IEEE Commun. Mag.,* pp. 84–90, January 2000.

[12] A. Gatherer and E. Auslander, *The Application of Programmable DSPs in Mobile Communications*, John Wiley & Sons, New York, 2002.

[13] J. Hagenauer and P. Hoeher, A Viterbi algorithm with soft-decision outputs and its applications, *Proc. Globecom '89,* pp. 47.1.1–47.1.7, Nov. 1989.

[14] H. Igura, Y. Naito, K. Kazama, I. Kuroda, M. Motomura, and M. Yamashina, An 800-MOPS, 110-mW, 1.5-V, parallel DSP for mobile multimedia processing, *IEEE J. Solid-State Circuits,* Vol. 33, pp. 1820–1828, Nov. 1998.

[15] H. Kabuo, M. Okamoto, et al., An 80-MOPS peak high-speed and low-power consumption 16-bit digital signal processor, *IEEE J. Solid-State Circuits,* Vol. 31, No. 4, pp. 494–503, 1996.

[16] P. Lapsley, J. Bier, A. Shoham, and E. Lee, *DSP Processor Fundamentals,* IEEE Press, 1997.

[17] E.A. Lee, Programmable DSP processors: part I and II, *IEEE ASSP Mag.,* Oct. 1988 and Jan. 1989.

[18] W. Lee et al. A 1-V programmable DSP for wireless communications, *IEEE J. Solid-State Circuits,* Vol. 32, No. 11, Nov. 1997.

[19] S. Mutoh, S. Shigematsu, Y. Matsuya, H. Fukuda, and J. Yamada, A 1-V multi-threshold voltage CMOS DSP with an efficient power management technique for mobile phone application, *IEEE Int. Conf. on Solid-State Circuits,* Paper FA 10.4, pp. 168–169, Feb. 1996.

[20] M. Okamoto, K. Stone, T. Sawai, H. Kabuo, S. Marui, M. Yamasaki, Y. Uto, Y. Sugisawa, Y. Sasagawa, T. Ishikawa, H. Suzuki, N, Minamida, R. Yamanaka, and K. Ueda, A high-performance DSP architecture for next generation mobile phone systems, *1998 IEEE DSP Workshop*.

[21] A. Olofsson and F. Lange, A 4.32-GOPS 1- general-purpose DSP with an enhanced instruction set for wireless communications, *Proc. ISSCC,* pp. 54–55, Feb. 2002.

[22] M. Peresse, K. Djafarian, J. Chaoui, D. Mazzocco, and Y. Masse, Enabling JPEG2000 on 3-G wireless mobiles through OMAP architecture, *Proc. Acoustics, Speech, and Signal Processing, 2002 (ICASSP '02),* Vol. 4, May 13–17, pp. IV-3796–IV-3799, 2002.

[23] T. Rappaport, *Wireless Communications, Principles & Practices,* IEEE Press, New York and Prentice Hall, New Jersey, 1996.
[24] TMS320C55x DSP Mnemonic Instruction Set Reference Guide, document SPRU374C, June 2000, available from www.ti.com.
[25] P. Schaumont and I. Verbauwhede, Domain-specific codesign for embedded security, *IEEE Comput. Mag.,* pp. 68–74, April 2003.
[26] T. Shiota, I. Fukushi, R. Ohe, W. Shibamoto, M. Hamaminato, R. Sasagawa, A. Tsuchiya, T. Ishihara, and S. Kawashima, A 1-V, 10.4-mW low-power DSP core for mobile wireless use, *1999 Symp. on VLSI,* Paper 2-2, 1999.
[27] Starcore launched first architecture, *Microprocessor Report,* Vol. 12, No. 14. p. 22, Oct. 1998.
[28] W. Strauss, DSP Market Bulletin, Forward Concepts, June 2, 2004, available from www.forward-concepts.com, June 2004.
[29] D. Talla, C. Hung, R. Talluri, F. Brill, D. Smith, D. Brier, B. Xiong, and D. Huynh, Anatomy of a portable digital mediaprocessor, *IEEE Micro.,* Vol. 24, Issue 2, pp. 32–39, March–April 2004.
[30] I. Verbauwhede and M. Touriguian, A low-power DSP engine for wireless communications, *J. VLSI Signal Process.,* Vol. 18, pp. 177–186, 1998.
[31] I. Verbauwhede and C. Nicol, Low-power DSPs for wireless communications, *Proc. Int. Symp. on Low-Power Electron. Design (ISLPED 2000),* pp. 303–310, July 2000.
[32] I. Verbauwhede and M.-C.F. Chang, Reconfigurable interconnect for next-generation systems, *Proc. ACM/Sigda 2002 Int. Workshop on System Level Interconnect Prediction (SLIP 2002),* Del Mar, CA, pp. 71–74, April 2002.
[33] M. Weiss, F. Engel, and G. Fettweis, A new scalable DSP architecture for system on chip (SoC) domains, *Proc. IEEE ICASSP Conf.,* May 1999.
[34] J. Williams, K.J. Singh, C.J. Nicol, and B. Ackland, A 3.2-GOPs multiprocessor DSP for communication applications, *Proc. IEEE ISSCC 2000,* Paper 4.2, San Francisco, February 2002.
[35] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. Rabaey; A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing, *IEEE J. Solid-State Circuits,* Vol. 35, pp. 1697–1704, Nov. 2000.

3

Energy-Efficient Reconfigurable Processors

Raphaël David
Sébastien Pillement
Olivier Sentieys
ENSSAT/University of Rennes

3.1 Introduction

Rapid advances in silicon technology and embedded computing bring two conflicting trends to the electronics industry. On the one hand, high-performance embedded applications dictate the use of complex battery-powered devices. The evolution of the battery capacity being significantly lower than that of the application complexity, energy efficiency becomes a critical issue in the design process of these systems. On the other hand, these systems have to be flexible enough to support rapidly evolving applications, restricting the use of domain-specific architectures. These trends have led to the reconfigurable computing paradigm [1,2].

Formally, configuring permits the adjustment of something or a change in the behavior of a device so that it can be used in a particular way. This definition leads to a very large design space bounded by bit-level reconfigurable architectures and by von Neumann-style processors. Common execution (i.e., reconfiguration) schemes can be extracted for different paradigms in this design-space [3,4], on the basis of the processing primitive granularity. On one side of the design space, bit-level reconfiguration is used in field programmable gate-array (FPGA). They provide bit-level reconfigurability and typically use mesh topology for their interconnection network. They allow designers to fully optimize the architecture at the bit level. The flexibility of these devices is associated with a very important configuration data volume and with performance and energy overheads. On the opposite side, system-level reconfiguration corresponds to instruction-based processors, including digital signal processors (DSP). They achieve flexibility through a set of instructions that dynamically modify the behavior of statically connected components. Their performance is limited by the amount of operator parallelism. Futhermore, their power-hungry data and instruction access mechanisms lower their energy efficiency.

In between, to increase the optimization potential of programmable processors without the bit-level reconfigurable architecture drawbacks, the functional-level reconfiguration has been introduced for reconfigurable processors. In such architectures, functional units as well as their interconnection network are reconfigurable and handle worldwide data. Most of these architectures use two-dimensional network topologies, usually hierarchical [5], for communications between functional units. In this context, numerous approaches have been proposed, such as DReAM [6], Morphosys [7], Piperench [8], FPFA [9], RaPiD [10], or Pleiades [11]. The main concern of these architectures is to introduce flexibility while maintaining high performance and reducing reconfiguration cost.

Reconfigurable architectures such as the Chameleon [12] have demonstrated their efficiency on implementing 3G base stations. More generally, reconfigurable architectures have demonstrated their efficiency on computation-hungry signal processing applications. Unfortunately, energy efficiency has been rarely a topic of interest in the reconfigurable framework. In this chapter we focus on the energy/flexibility trade-off for high-performance reconfigurable architectures.

Section 3.2 presents the energy efficiency criterion and highlights energy wastes in the reconfigurable design space as well as the opportunities to reduce energy consumption. Section 3.3 presents the DART architecture implementing energy aware design techniques and innovative reconfiguration schemes. Finally, Section 3.4 discusses the implementation results of a key application of next-generation mobile communication systems.

3.2 Energy Efficiency of Reconfigurable Architectures

3.2.1 Problem Definition

The energy efficiency (E.E.) of an architecture can be quantified by considering the number of operations it processes per second when consuming one mW. This parameter can be defined by Equation (3.1) [13]:

$$E.E. = \frac{N_{OP}.F_{clk}}{A_{Chip}.\alpha.C_N.F_{clk}.V_{DD}^2} \quad \left[\frac{MOPS}{mW}\right] \tag{3.1}$$

where *NOP* is the number of operations computed at each cycle and *Fclk* the operating frequency [*MHz*]. *AChip* is the total area of the chip [*mm*2], *CN* the normalized capacitance by area unit [*mF*/*mm*2], α the average activity, and *VDD* the supply voltage [*V*]. The product *NOP.Fclk* thus represents the computation power of the architecture and is given in millions of operations per second (MOPS). The product $A_{Chip}.\alpha.C_N.F_{clk}.V_{DD}^2$ gives the power consumed during the execution of the *NOP* operations. *AChip* parameter is obtained by Equation (3.2):

$$A_{Chip} = N_{opr}.A_{opr} + A_{mem} + A_{ctrl} \tag{3.2}$$

where *Aopr* is the average area per operator, *Nopr* the number of operator in the design. *Nopr.Aopr* thus represents the operator area in the design. *Amem* is the memory area and *Actrl* the area of the control and configuration management resources.

These two equations can be used to find out which parameters could best be optimized to design an energy efficient architecture. The $N_{OP}.F_{clk}$ product has to cover the needs of the implemented application (i.e., the architecture has to be powerful enough to compute the application). Consequently, N_{OP} and F_{clk} need to be jointly optimized. The normalized capacitance mainly depends on the technology. So, its optimization was not studied for this work.

The definition of an energy aware architecture dictates the optimization of the remaining parameters: average operator area, storage, and control resource area as well as activity through the circuit and of course clock frequency and supply voltage. To define an optimal *delay* × *power* product, parallelism inherent to the implemented system must finally be fully exploited.

According to the energy efficiency criterion, application-specific integrated circuits (ASIC) can be considered as the ultimate solution. Because they are dedicated only for one specific processing, no computational unit is larger or more complicated than it has to be. In such devices, the operator area (A_{opr}) is thus minimized. Moreover, no architectural mechanisms have to be introduced to support flexibility (i.e., there is no need to fetch and decode instructions). The execution is fully deterministic and all known optimizations, such as using wires instead of shifters, can be used. The circuit is controlled thanks to a finite state machine and the control area of the chip A_{ctrl} is also minimized.

In such designs, processing parallelism can be fully exploited. By increasing the parallelism level, the operating frequency along with the supply voltage can be reduced, and therefore an optimal energy delay product can be achieved. Moreover, because there is no resource waste, design area is reduced. Finally, clock distribution energy waste can also be minimized by defining several clock domains.

With these devices, data accesses are fully determined at the synthesis time. Thus, data can be placed as near as possible to the functional units that will handle them. A memory hierarchy can also be defined in order to minimize the energy consumed by data transactions within the architecture. Furthermore, optimizations such as first-in first-out (FIFO) memory instead of static random-access memory (SRAM) can be used. Consequently, memory area (A_{mem}) is reduced along with energy.

Beside classical high-performance and low-energy consumption constraints, flexibility becomes a major concern to the development of multimedia and mobile communication systems. This dictates the use of programmable or even reconfigurable devices [14]. The next section discusses energy efficiency optimization techniques that can be applied in the case of reconfigurable processors.

3.2.2 Energy Efficiency Optimization

3.2.2.1 Energy in Computations

An architecture is considered as energy efficient only if its operators are the main source of energy consumption. Nevertheless, the optimization effort for these components has also to be important. Programmable processors integrate in their datapath, general-purpose functional units designed to perform a large variety of computations. They are thus significantly more complicated than they need to be, and are a source of energy waste. Moreover, if their bit-width is larger than the data length used in the algorithm, additional energy is wasted.

On the contrary, in bit-level reconfigurable architectures, each operator is built to execute only one operation. The very fine granularity of the computation primitive (e.g., look-up tables) dictates the association of numerous cells. Consequently, the power dissipated in such an operator is mainly issued from the interconnection network (60 to 70%) [15,16]. Even if the operators are tailored-made to execute only one operation on fixed-size data, they are inefficient from an energy point of view.

To reduce energy waste, the amount of operations supported by functional units has to be limited. A functional decomposition of these units leads to the isolation of its different parts by using latches. In this case, only transistors useful to the execution consume dynamic power.

Many application domains handle several data sizes during time (e.g., 8, 11, 13, and 16 bits). To support all these data sizes, very flexible functional units have to be designed. Consequently, latency and energy penalties occur. Another alternative is to optimize functional units only for a subset of these data sizes by designing subword parallelism (SWP) operators [17]. This technique consists of dividing an operator working on N-bit data to allow the execution of k operations in parallel on part of the input data of N/k length. Integrating such operators allows to increase computation power while keeping the consumed energy per operator nearly constant during processings with data-level parallelism. Therefore, SWP can increase energy efficiency of the architecture.

3.2.2.2 Exploiting the Parallelism

To minimize energy consumption, supply voltage has to be reduced aggressively. To compensate the associated performance loss, concurrent execution must be supported. Digital signal processing algorithms provide several levels of parallelism that can be exploited to achieve this objective.

Operation- or instruction-level parallelism (ILP) is inherent to every computation algorithm. Although it is constrained by data dependencies, its exploitation is generally quite easy. It requires the introduction of several functional units working independently with each other. To exploit this parallelism, the architecture controller has to be able to jointly specify, to several operators, the operation to be executed.

Thread-level parallelism (TLP) represents the number of processings, which may be executed concurrently to implement an algorithm. The TLP is far more complicated to exploit than ILP. TLP strongly varies from one application to another, and even more between two descriptions of the same application. To exploit this parallelism while guaranteeing a good computation density, the architecture must be able to adapt its organization of processing resources [18]. The trade-off between ILP and TLP must thus be adapted to the application to be executed. This can be realized by organizing the architecture into a hierarchy, the lowest level of which being a set of functional units (e.g., a datapath). Each datapath should be able to be controlled independently to implement a particular thread of the application. On the contrary, the datapaths should be interconnected as a single resource exhibiting a large amount of ILP.

Application or algorithm parallelism can be considered as an extension of thread parallelism. The goal is here to identify the applications that may be implemented concurrently on the architecture. On the opposite of threads, applications executed in parallel are working on distinct data sets. To exploit this kind of parallelism, a second level of hierarchy needs to be added to the architecture. The architecture may be divided into clusters working independently on several applications. These clusters have their own control, storage, and processing resources.

3.2.2.3 Reducing the Control Overhead

In a reconfigurable processor, two types of information are needed to manage the architecture: the configuration data which specify the hardware structure of the architecture (operators, logic, interconnections), and the control, which manages the data transactions within the architecture. Distributing the configuration and control information has a significant impact on performance and energy efficiency of the system. This is mainly issued from the configuration and control data volume needed to execute an application and to the reconfiguration frequency.

The architectural paradigms included in the reconfigurable design space have very different strategies to distribute these information. Bit- and system-level reconfigurable architectures have the two most extreme reconfiguration schemes. On the one hand, a very large amount of configuration data (several thousands or millions of bits) is distributed in FPGA architecture. The reconfiguration cost is very high but once specified, there is no control overhead. On the other hand, programmable processors eliminate the overhead linked to the specification of the datapath because it is fixed. The control cost of the architecture is very important, however, and corresponds to fetch and decode instructions at each cycle.

The 80/20 rule asserts that 80% of the execution time is consumed by 20% of the program code [19]. Few portions of source code are thus executed during long periods of time. These blocks of code are described as regular and are typically loop kernels during which a same computation pattern is used for long time. Between these blocks of regular code, instructions follow one another without particular order and in a nonrepetitive way. These portions of code are described as irregular. Because of their lack of parallelism, they present few optimization opportunities.

To minimize the architecture control cost, the distribution strategy can be adapted to the implemented processing. For this purpose, regular and irregular processings have to be distinguished to define two reconfiguration modes. The first one is used to specify the architecture configurations which will allow optimal implementations of regular processings. The second reconfiguration mode is used to specify the control information that will allow to execute irregular processings. By reducing the amount of reconfiguration targets, functional-level reconfigurable architectures limit the configuration data volume associated to the datapath structure specification.

To reduce even more the configuration data volume, redundancy in datapath can also be exploited. It allows to distribute simultaneously the same configuration information to several targets, whenever these targets execute the same processing.

3.2.2.4 Reducing the Data Access Cost

Data access cost also has a significant impact on the energy efficiency of the architecture. It depends on the amount of memory accesses and on the cost of one memory access. In programmable processors, each computation step dictates register file accesses. These architectures cannot completely exploit spatial and temporal locality of data because all the data have to be stored in registers. Thanks to bit- or functional-level reconfiguration, operators may be interconnected to efficiently exploit the locality of data. Spatial locality is used by directly connecting operators. Producer and consumer of data can be directly connected, no memory transfers are necessary to store intermediate results. Temporal locality can be exploited thanks to one-to-all connections. That kind of connection allows the transfer of one data element toward several targets in a single transaction and to skip redundant data accesses. This temporal locality may moreover be exploited thanks to delay chains, which reduce the amount of memory accesses when several samples of a same signal are concurrently handled in an application.

Defining a memory hierarchy reduces the data access cost while providing a high memory bandwidth [20]. This hierarchy has to combine high capacity, high bandwidth, and energy efficiency.

Because multi-port memories are characterized by high-energy consumption, it is more efficient to integrate several single-port memories. High-bandwidth and low-energy constraints thus require the integration of a large amount of small memories. Moreover, to provide a quite important storage space, a second level of hierarchy can be added. Finally, to reduce memory management costs, the address generation task is distributed along with the memories.

Associating flexibility with high-performance and energy efficiency, is a critical issue for embedded applications. Beside the dynamically reconfigurable device Xc6200 from Xilinx [21], numerous research projects have contributed to the simplification of the reconfiguration process to increase performance and flexibility (e.g., Singh et al. [7], Goldstein et al. [8], Cronquist et al. [10], and Callahan et al. [22]). Despite the energy optimization potential of reconfigurable architectures, few projects have integrated this constraint. In Abnous and Rabaey [11], the authors propose a low-power reconfigurable processor. Because it is a domain-specific platform, its flexibility is limited. Furthermore, the validation of this platform has only been proposed for medium-complexity application domain, such as speech coding [23].

The next section discusses a reconfigurable processor associating energy efficiency, high performance and flexibility. This architecture is based on the optimization mechanisms presented in this section.

3.3 The DART Architecture

DART is a hierarchical architecture supporting the different levels of parallelism. To exploit task parallelism, DART has been broken up into clusters. Distinct tasks can be processed concurrently by clusters because each of them has its own control and storage resources. At the system level, tasks are distributed to clusters by a controller. This controller supports the real-time operating system which assigns tasks to clusters according to urgency and resources availability constraints. The system level of DART also includes shared memories (data, configuration) and an I/O block which allows its interfacing with external components through a standard bus (e.g., AMBA and VCI).

This section first describes the architecture of DART clusters. Next, the processing primitives are presented. Finally, dynamic reconfiguration and development tools are discussed.

3.3.1 Cluster Architecture

Each cluster of DART (Figure 3.1) integrates two types of processing primitives: some reconfigurable datapath (RDP) used for arithmetic processing and an FPGA core processing data at the bit level. The RDPs, detailed in the next section, are reconfigurable at the functional level to optimize the interconnections between arithmetic operators according to the calculation pattern. The FPGA core is reconfigurable at the gate level to efficiently support bit-level parallelism of processings (e.g., generation of Gold or Kasami codes in wideband code division multiple access (WCDMA) or channel coders [24]). Using these two kinds of operators allows an architecture to be defined in adequacy with the algorithm for a

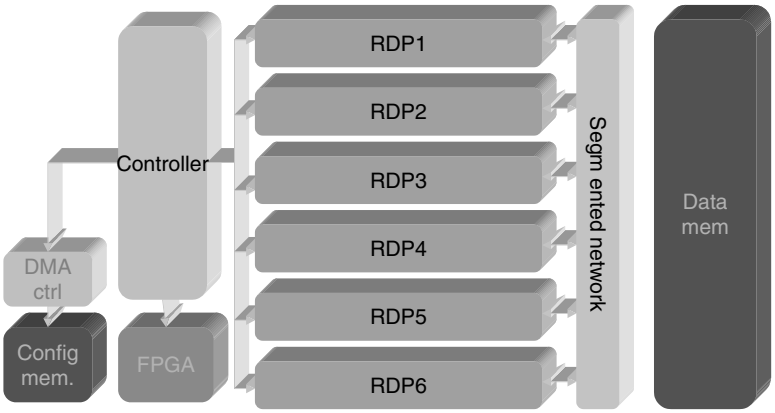


FIGURE 3.1 Architecture of a DART cluster.

large set of applications. Experiments have demonstrated that integrating one FPGA core and six RDPs in each cluster of DART delivers enough calculation power.

The RDPs are interconnected thanks to a segmented mesh network. Depending on the parallelism level of the application, the RDPs can be interconnected to compute in a highly parallel fashion to support high ILP, or can be disconnected to work independently on different threads. The segmented network allows dynamic adaptation of the instruction- and thread-level parallelism of the architecture, depending on the processing needs.

This hierarchical organization of DART allows not only the distribution of control but also that of the processing resources. Thus, it is possible to efficiently connect a very large number of resources without being too penalized by the interconnection cost. The processing resources distribution allows the definition of a hierarchical interconnect network, which is significantly more energy efficient for complex design than a typical global interconnection networks [5]. With this kind of network, the lowest level of the resource hierarchy is completely connected while the higher levels communicate via the segmented network.

Moreover, thanks to the flexibility of this topology, the resulting architecture becomes a better target for the development tools.

All the processing primitives (i.e., the FPGA and RDPs) access the same data memory space and their reconfigurations are managed by a controller. To minimize the associated control overhead, reconfigurations of the FPGA are realized via a DMA controller. The cluster controller has only to specify an address bound to the DMA controller, which will transfer the data from a configuration memory toward the FPGA. Besides that, the cluster controller also manages the reconfiguration of the RDPs via instructions. Its architecture is similar to that of a typical programmable processor, but it distributes configurations instead of instructions. Consequently, it does not have to access an instruction memory at each cycle. Fetch and decode operations are only realized when a reconfiguration occurs and are hence very infrequent. This drastic reduction of the instruction memory readings and decodings leads to very significant energy savings (cf. Section 3.4).

3.3.2 RDP Architecture

The arithmetic processing primitives in DART are the RDPs (Figure 3.2). They are organized around functional units and memories that are interconnected according to a very powerful communication network. Every RDP has four functional units (two multipliers/adders and two ALUs) handling double-precision 16-bit data, followed by a register. They support SWP (subword parallelism) processings and have been designed with low-power concerns [25].

The functional units are dynamically reconfigurable (see next section) and are working on data stored in four small local memories. On the top of each memory, four local controllers (the *AGi* on the top of

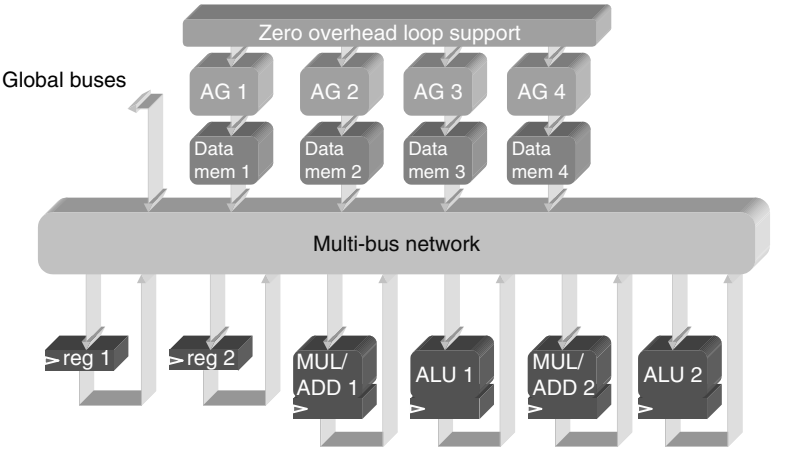


FIGURE 3.2 Architecture of an RDP.

Figure 3.2) are in charge of providing the addresses of the data handled inside the RDPs. These local controllers are like tiny reduced instruction-set computer (RISC) processors and support a large set of addressing patterns. The four local controllers of each RDP share a zero-overhead loop support. In addition to the memories, two registers are also available in every RDP. These registers are used to build delay chains, and hence to realize time data sharing.

All these resources are connected through a completely connected network. The hierarchical organization of DART permits these connections to be kept relatively small and hence to limit their energy consumption. Thanks to this network, each resource of the RDP can communicate with every other resource and hence, the datapath can be optimized for every calculation pattern. Moreover, this flexibility eases data sharing. Indeed, because a memory can simultaneously be accessed by several functional units, some energy savings can also be achieved. The upper left part of Figure 3.2 depicts the connections with global buses that allow the connection of several RDPs to implement massively parallel processing.

3.3.3 Dynamic Reconfiguration

One of the main features of DART is to support two RDP reconfiguration modes which ensues from the 80/20 rule (see Section 3.2). During regular processing, the RDPs are dynamically reconfigured to be adapted to the calculation pattern. This reconfiguration — hardware reconfiguration — may take a few cycles, but is used for long period of time. On the contrary, during irregular processing, the calculation pattern is changing very often. In that case, the reconfiguration time has to be minimized, and the RDPs structure is modified thanks to software reconfiguration. Another important feature of a DART cluster is to exploit the redundancy in the RDPs to minimize the configuration data volume.

3.3.3.1 SCMD Concept

A portion of code is usually qualified as regular when it is used for a long period of time, and applied to a large set of data, without being suspended by another processing. Loop kernels support this qualification because their computation patterns are maintained during all the loop iterations. Instruction-level parallelism of such regular processing is often exhibited by compilation techniques such as loop unrolling or software pipelining [26]. With such techniques, the computation pattern of the loop kernel is repeated several times, which leads to a highly regular architecture. If this loop kernel is implemented on several RDPs, their configuration might be redundant. Specifying several times the same configuration is an energy waste, we then introduce a concept called single configuration multiple data (SCMD). It may be considered as an extension of SIMD (single instruction multiple data), in which several operators execute the same operation on different data sets. Within the framework of SCMD, the configuration data sharing is no longer limited to the operators but is extended to the RDPs.

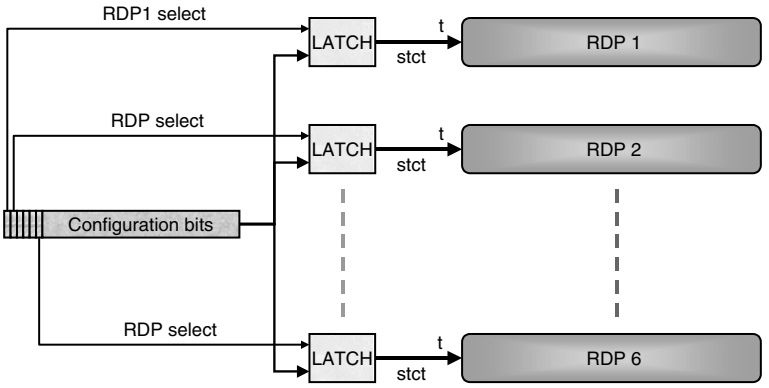


FIGURE 3.3 SCMD implementation for DART.

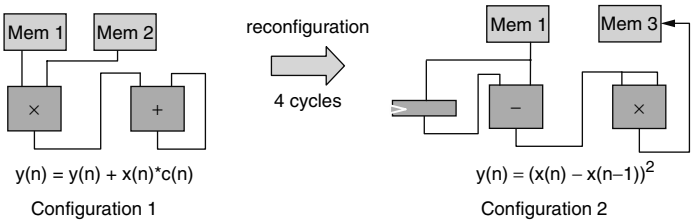


FIGURE 3.4 Hardware reconfiguration example.

The SCMD concept allows the simultaneous configuration of several RDPs. Practically, a field is concatenated to the configuration instructions to specify the targets of the configuration bits. With six RDPs, six bits have to be added to the instruction. Each RDP validates the configuration instructions according to the value of its select bit (Figure 3.3). This allows the reduction of the configuration data volume for regular computations, where there is a lot of redundancy between the RDP configurations.

3.3.3.2 Hardware Reconfiguration

During regular processing, a complete flexibility of the RDPs will be allowed by the full use of the functional-level reconfiguration paradigm. By allowing the modification of the way in which functional resources and memories are interconnected, the architecture can be optimized for the computation pattern that has to be implemented. With six RDPs, the configuration data volume for a cluster is 826 bits. According to the regularity of the computation pattern and the redundancy of the RDP configurations (which influences the SCMD performances), between three and nineteen 52-bit instructions will be required to reconfigure all the RDPs and their interconnections. Once these configuration instructions have been specified, no other instruction readings and decodings have to be done until the end of the loop execution.

This kind of configuration can for example be illustrated by Figure 3.4. The datapath is optimized at first in order to compute a digital filter based on multiply-accumulate operations. Once this configuration has been specified, the data-flow computation model is maintained as long as this computation pattern is used. At the end of the computation, after a reconfiguration step that needs four cycles, a new datapath is specified in order to be in adequacy with the calculation of the square of the difference between $x(n)$ and $x(n\text{-}1)$. Once again, no control is necessary to end this computation.

3.3.3.3 Software Reconfiguration

For irregular processing, which implies frequent modifications of the RDP configuration, a software reconfiguration has also been defined. To be able to reconfigure the RDP in one cycle with an instruction of reasonable size, their flexibility has been limited. In that case, DART uses a read-modify-write behavior,

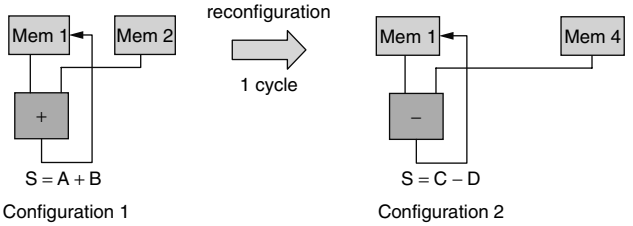


FIGURE 3.5 Software reconfiguration example.

such as that of very long instruction word (VLIW) processors. For each operator used, the data are read in memory, computed, and then the result is stored back to the memory at each cycle.

This software reconfiguration thus concerns only the functionality of the operators, the size of the data and their origin. Thanks to these flexibility limitations, the RDP may be reconfigured at each cycle with only one 52-bit instruction. This is illustrated on Figure 3.5, which represents the reconfiguration needed to replace an addition of data stored in the memories 1 and 2 by a subtraction on data stored in the memories 1 and 4.

Thanks to these two reconfiguration modes and to the SCMD concept, DART supports every kind of processing while being able to be optimized for the critical (i.e., regular) ones. These two types of reconfiguration can moreover be mixed without any constraint, and have a great influence on the development methodology.

3.3.4 Development Flow

To exploit the computation power of DART, an efficient development flow is the key to enhance the status of the architecture. Hence, a compilation framework, which allows the exploitation of the previously mentioned programming models, has been defined. It is based on the joint use of a front-end allowing the transformation and the optimization of a C code [27], a retargetable compiler [28], and an architectural synthesis tool [29]. As in most development methodologies for reconfigurable hardware, the key of the problem has been to distinguish the different kinds of processing. This approach has already been used with success in the program-in chip-out (PICO) project developed at HP labs in order to distinguish regular codes, implemented in systolic array, and irregular codes, executed in a VLIW processor [30]. Other related works such as the Pleiades project [31] or GARP [32] are also distinguishing regular processings and irregular ones. Massively parallel processings are implemented on circuits respectively reconfigurable at the functional and at the bit level, and irregular codes are executed on a RISC processor.

The development flow allows the user to describe its applications in the C language. This high-level description is translated at first into control and data flow graph (CDFG), from which some automatic transformations (e.g., loop unrolling and loop kernel extractions) [33] are done to optimize the execution time. After these transformations, the distinction between regular and irregular codes and data manipulations permits to translate, thanks to the compilation and the architectural synthesis, a high-level description of the application into binary executable codes for DART [34]. A cycle-accurate bit-accurate simulator developed in SystemC finally allows to validate the implementation and to evaluate its performance and energy consumption.

3.4 Validation Results

This section presents significant results stemming from a WCDMA receiver implementation on DART. Energy distribution between the different components of the architecture is also discussed. Performance and energy efficiency of DART is finally compared with typical reconfigurable architectures and programmable processors.

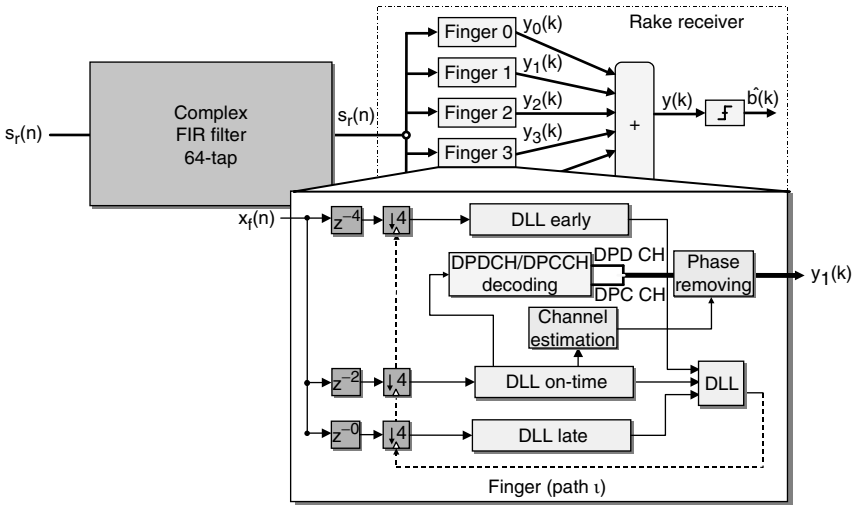


FIGURE 3.6 WCDMA receiver synoptic.

3.4.1 Implementation of a WCDMA Receiver

WCDMA is typically considered as one of the most critical applications of next-generation telecommunication systems [35]. A synoptic of this receiver is given in Figure 3.6. Within a WCDMA receiver, real and imaginary parts of data received on the antenna, after demodulation and digital-to-analog conversion, are filtered at first thanks to two real FIR (finite impulse response) filters. These two 64-tap filters operate at a high frequency (15.36 MHz), which lead to a tremendous complexity of 1966 millions of MAC per second (MMACS). Next, a rake receiver has to extract the usable information in the filtered samples and to retrieve the transmitted symbol. Because the transmitted signal reflects in obstacles like buildings or trees, the receiver gets several replica of the same signal with different delays and phases. By combining the different paths, the decision quality is drastically improved and consequently, a rake receiver is constituted of several fingers, which have to despread one part of the signal, corresponding to one path of the transmitted information. The decision is finally done on the combination of all these despreaded paths. The complexity of this complex despreading is 184.3 MOPS for six paths. To improve the decision quality, amplitude and delay of each path have to be estimated and removed from the signal. The synchronization between the received signal and the codes internally generated, i.e., the delay estimation and removing, is done in two times. The first part of this processing operates at a high-frequency (chip rate: F_c = 3.84 MHz) and has a complexity of 331 MOPS. The second one operates at the symbol frequency (F_s), which depends on the required bit-rate, and has a low complexity (e.g., 1.3 MOPS for a spreading factor of 256). Finally, the channel estimation is a low-complexity process and also operates at F_s.

Therefore, five configurations of the architecture may be distinguished: filtering, chip-rate synchronization, symbol-rate synchronization, channel estimation, and complex despreading. They follow one another on the architecture as depicted in Figure 3.7.

DART clusters have been designed under a 1.9-V, 0.18-μm technology. The synthesis has led to an operating frequency of 130 MHz. Running at this frequency, DART is able to provide up to 3120 MMACS per cluster, on 8-bit data. Thanks to the cycle-accurate bit-accurate simulator, the overall energy consumption of the architecture is evaluated from the activity of the different modules (e.g., functional units, memories, interconnection networks, control and registers) and their average energy per access. The latter has been estimated at the gate level.

Thanks to the configuration data volume minimization, reconfiguration stages are very short, and thus represent only 0.05% of the overall execution time. The effective computation power proposed by DART on these applications is 6.2 giga operations per second (GOPS). In such conditions, the processing of a

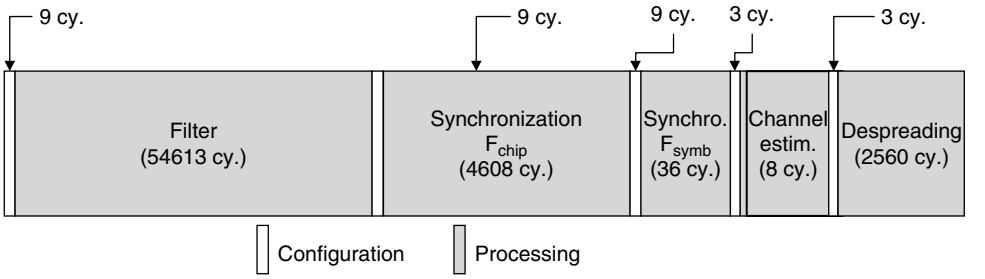


FIGURE 3.7 DART reconfigurations during the processing of one slot.

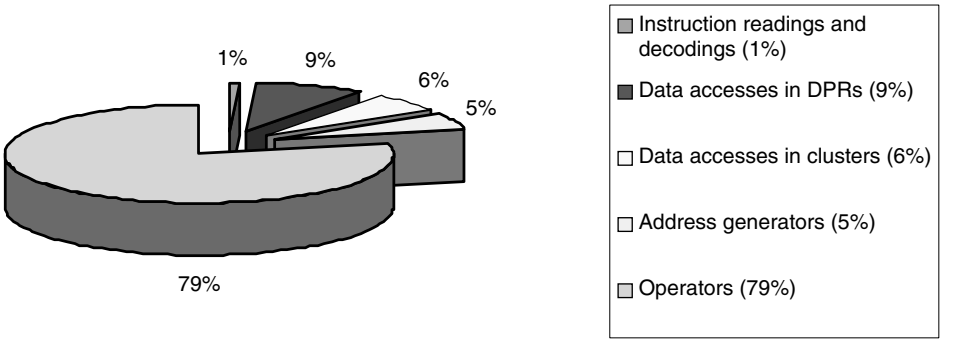


FIGURE 3.8 Power consumption distribution in a DART cluster during the processing of the WCDMA receiver.

WCDMA receiver on DART leads to a cluster usage rate of 72.6%. This performance level is done possible by the flexibility of the DART interconnection network, which allows a nearly optimal use of the RDP internal processing resources.

3.4.2 Energy Distribution in DART

The average energy efficiency of DART during the implementation of this WCDMA receiver is 38.8 MOPS/mW. Figure 3.8 represents the power consumption distribution between the different components of the architecture. We can notice that the main part of the cluster consumption comes from the operators (79%). Thanks to the configuration data volume minimization and to the reconfiguration frequency reduction, the energy overhead associated to the control of the architecture is negligible. During this processing, only 0.9 mW are consumed to fetch and decode control information, that is to say less than 0.8% of the 114.8 mW needed for the processing of a WCDMA receiver.

The power consumption issuing from data accesses is also reduced (20% for memory accesses and address generation). This is notably due to the minimization of the energy cost of local memory access, obtained by the definition of an appropriate memory hierarchy. In the same time, one-to-all connections allow to significantly reduce the amount of data memory accesses. In particular, on filtering and complex despreading applications, which exploit a thread-level parallelism, the simultaneous use of several functional units with a same data-flow allows to drastically reduce the number of accesses to the data memory. The use of delay chains also allows to benefit from data temporal locality and to skip a lot of data memory accesses. For this WCDMA receiver, the joint use of delay chains and of one-to-all connections permit to save 46 mW, representing a 32% reduction of the overall consumption of a cluster.

3.4.3 Performance Comparisons

This section compares the performances of DART with bit- and system-level reconfigurable architectures. The first considered architecture is the Virtex Xcv200E FPGA from Xilinx [36]. This choice is justified

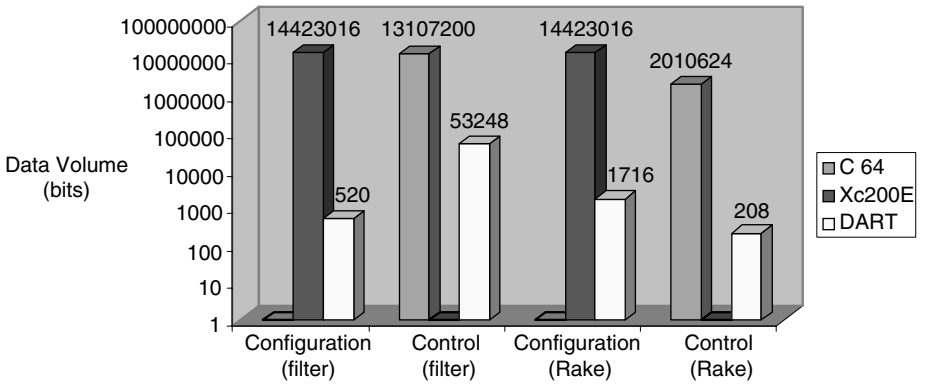


FIGURE 3.9 Configuration and control data volume of the Xc200E, the C64x, and DART for filter and rake receiver.

by the need to study the reconfiguration cost. This component has been dimensioned to have a computation power strictly corresponding to the FIR filters implementation. Two configurations will thus follow one another on the FPGA: the filter and the rake receiver. The configuration data volume for this circuit is about 1.4 Mbits. As DART, it is distributed on a 0.18-μ*m* technology. The second considered architecture is the TMS320C64x digital signal processor (DSP) from Texas Instruments. This DSP is a VLIW architecture able to exploit an ILP of eight as well as data-level parallelism thanks to SWP capabilities [37]. This processor is distributed on a 0.12-μ*m* technology. With a 720-MHz clock frequency, it may deliver up to 5760 MOPS.

Configuration and control cost has a critical impact on the performance and the energy efficiency of the system. Figure 3.9 represents the information volume needed for these two operations during the filtering and the rake receiver applications, for the C64, the Xc200E, and DART. Figure 3.9 clearly illustrates the conceptual divergences between the bit-level reconfiguration and the system-level reconfiguration. In the case of FPGA, a very large amount of information is distributed to the component before executing the application. The reconfiguration cost is here very important but once specified, the reconfiguration has no influence on the execution time. On the other hand, system-level reconfigurable architectures do not need to configure the datapath structure. On the contrary, the architecture control cost is critical. It corresponds, for the C64, to a 256-bit instruction reading and decoding at each cycle. DART allows the trade-off of these two operations, and thus minimizes energy waste. The hardware configuration between the processing is limited to the distribution of few hundreds of bits, while the architecture control is limited to the specification of reset instructions.

These considerations, relative to the architecture management, partly explain the results appearing in Figure 3.10, which represents the computation time associated to the three architectures according to

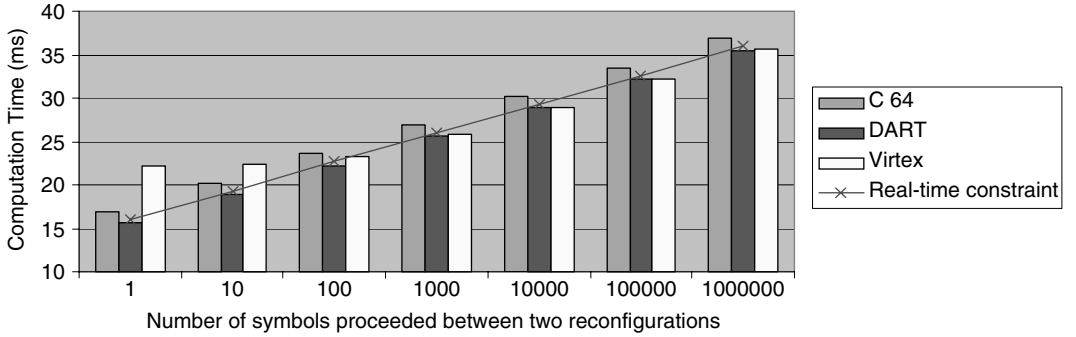


FIGURE 3.10 Xc200E, TMS320C64x, and DART performance on a WCDMA receiver.

the number of symbols processed between two reconfigurations. The normalized real-time deadline represented on Figure 3.10 demonstrates that the DSP performance does not allow the implementation of the WCDMA receiver in real time, even when SWP is fully exploited. The FPGA is able to support the real-time constraint when the number of symbols processed between two reconfigurations exceeds 150. In such conditions, the configurations have to be stable during at least 10 ms because the reconfiguration time of this component is 2.7 ms. This remark highlights the impact of the reconfiguration overhead. In such conditions, it is essential to filter all the samples of a frame (153,600 samples), to store the filtered data into the memory, then to reconfigure the component to decode these data.

The power consumption associated with the implementations on the FPGA has been estimated at the gate level with the Xpower tool from Xilinx. The FPGA consumes 670 mW during the filtering and 180 mW during the rake receiver. In other words, the energy efficiency of the Xcv200E is 5.8 MOPS/mW during the filters and 2.9 MOPS/mW during the rake receiver.

An important drawback of the implementation with the WCDMA receiver on the Xc200E FPGA thus comes from the large delay separating data receiving and data decoding. This temporal shift exceeds 10 ms and might be unacceptable in mobile applications. Another problem associated with this solution comes from the volume of temporary data. The need to store the filtered samples before to decode them implies the use of a large amount of memory. To store a frame, 1.2 Mbits of memory are needed, which exceeds the storage capacity of the Xc200E. It is thus necessary to use external memory, which implies an important energy overhead.

It has to be noticed that these drawbacks can be overcome by using larger chips. For example, the Xcv1000E, from the same FPGA family, allows the implementation of the WCDMA receiver in one configuration. In that case, no reconfiguration occurs and the real-time constraints can always be achieved. Obviously, this solution leads to a drastic increase of the device cost and to an energy efficiency reduction (about 20%).

The DSP power consumption has been estimated thanks to the results presented in Texas Instruments [38]. We have estimated this consumption to be 1.48 watt during the filters and 1.06 watt during the rake receiver. The energy efficiency of this architecture is therefore 2.6 MOPS/mW during the filters and 1.8 MOPS/mW during the rake receiver.

By minimizing the energy waste related to the architecture control and to the data accesses, DART is able to execute nearly 39 MOPS for each mW consumed. Unlike high-performance DSP and FPGA, flexibility does not come with a significant energy efficiency reduction.

3.5 Conclusions

This chapter discussed how to improve energy efficiency in flexible architectures. In such context, reconfigurable processors offer opportunities. They allow energy waste in control, storage as well as in computation resources to be reduced by adapting their datapath structure and by minimizing reconfiguration data volume. The association of key concepts, and of an energy aware design, lead to the definition of the DART architecture. Innovative reconfiguration schemes allow to deal concurrently with high-performance, flexibility, and low-energy constraints. We have validated this architecture by presenting implementation results of a WCDMA receiver. A computation power of 6.2 GOPS combined with an energy efficiency of 40 MOPS/mW demonstrate its potential in the context of multimedia mobile computing applications.

3.6 Acknowledgments

This project was conducted in collaboration with STMicroelectronics and UBO, and received funding from the French government. The authors would like to thank Dr. Tarek Ben Ismail and Dr. Osvaldo Colavin from STMicroelectronics, as well as Professors Bernard Pottier and Loïc Lagadec from UBO for their contributions.

References

[1] S. Hauck. The roles of FPGAs in reprogrammable systems. *Proc. IEEE*, 86:615–638, April 1998.

[2] E. Sanchez, M. Sipper, J.O. Haenni, J.L. Beuchat, A. Stauffer, and A. Perez-Uribe. Static and dynamic configurable systems. *IEEE Trans. on Computers*, 48(6):556–564, 1999.

[3] J. M. Rabaey. Reconfigurable processing: the solution to low-power programmable DSP. In *Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, April 1997.

[4] A. Dehon. Reconfigurable architectures for general-purpose computing. Ph.D. thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, October 1996.

[5] H. Zhang, M. Wan, V. George, and J. Rabaey. Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs. *Int. Workshop on VLSI*, April 1999.

[6] J. Becker, T. Pionteck, and M. Glesner. DReAM: a dynamically reconfigurable architecture for future mobile communication applications. *Int. Workshop on Field Programmable Logic and Applications (FPL '00)*, pp. 312–321, Villach, Austria, August 2000. Lecture Notes in Computer Science 1896.

[7] H. Singh, G. Lu, M. Lee, E. Filho, and R. Maestre. MorphoSys: case study of a reconfigurable computing system targeting multimedia applications. *Int. Design Automation Conf.*, pp. 573–578, Los Angeles, CA, June 2000.

[8] S. Goldstein, H. Schmit, M. Moe, M. Budiu, and S. Cadambi. PipeRench: a coprocessor for streaming media acceleration. *Int. Symp. on Comput. Architecture (ISCA '99)*, Atlanta, GA, May 1999.

[9] G. Smit, P. Havinga, P. Heysters, and M. Rosien. Dynamic reconfiguration in mobile systems. *Int. Conf. on Field Programmable Logic and Applications (FPL '02)*, pp. 171–181, Montpellier, France, September 2002. Lecture Notes in Computer Sciences 2438.

[10] D. C. Cronquist, P. Franklin, C. Fisher, M. Figueroa, and C. Ebeling. Architecture design of reconfigurable pipelined datapath. *Advance Research in VLSI (ARVLSI '99)*, pp. 23–40, Atlanta, GA, March 1999.

[11] A. Abnous and J. Rabaey. Ultra low-power specific multimedia processors. *In VLSI Signal Processing IX.* IEEE Press, November 1996.

[12] Chameleon Systems. Wireless base station design using reconfigurable communications processors. Technical report, 2000.

[13] B. Brodersen. Wireless systems-on-a-chip design. *Int. Symp. on Quality Electronic Design (ISQED '02).* Invited paper, San Jose, CA, March 2002.

[14] R. Hartenstein, M. Hertz, Th. Hoffman, and U. Nageldinger. Generation of design suggestions for coarse-grain reconfigurable architectures. *Int. Workshop on Field Programmable Logic and Applications,* Villach, Austria, August 2000. Lecture notes in Computer Science 1896.

[15] K.K.W. Poon. Power estimation for field programmable gate arrays. Master's thesis, University of British Columbia, Vancouver, Canada, 2002.

[16] L. Shang, A.S. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. *Int. Symp. on Field Programmable Gate Arrays (FPGA '02),* pp. 157–164, Monterey, CA, February 2002.

[17] J. Fridman. Sub-word parallelism in digital signal processing. *IEEE Signal Process. Mag.,* 17(2):27–35, March 2000.

[18] J.P. Wittenburg, P. Pirsh, and G. Meyer. A multithreaded architecture approach to parallel dsps for high-performance image processing applications. *Workshop on Signal Process. Syst. (SIPS '99),* Taipei, Taiwan, October 1999.

[19] G. Stitt, B. Grattan, J. Villarreal, and F. Vahid. Using on-chip configurable logic to reduce system software energy. *Symp. on Field-Programmable Custom Computing Machines (FCCM '02),* Napa, CA, September 2002.

[20] S. Wuytack, J.Ph. Diguet, F. Catthoor, and H. De Man. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings. *IEEE Trans. on VLSI Syst.,* 6(4):529–537, December 1998.

[21] Xilinx. *Xilinx 6200 Preliminary Data Sheet.* San Jose, CA, 1996.

[22] T.J. Callahan, J.R. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. *IEEE Comput.,* 33(4):62–69, April 2000.

[23] X. Zhang and K.W. Ng. A review of high-level synthesis for dynamically reconfigurable FPGAs. *Microprocessors and Microsystems,* 24:199–211, 2000.

[24] E. Dinan and B. Jabbari. Spreading codes for direct sequence CDMA and wideband CDMA cellular network. *IEEE Commun. Mag.,* 36(9):48–54, September 1998.

[25] R. David, D. Chillet, S. Pillement, and O. Sentieys. DART: a dynamically reconfigurable architecture dealing with next-generation telecommunications constraints. *Int. Reconfigurable Architecture Workshop (RAW '02),* Fort Lauderdale, FL, April 2002.

[26] P. Faraboshi, J.A. Fisher, and C. Young. Instruction scheduling for instruction level parallel processors. *Proc. IEEE,* 89(11):1638–1659, November 2001.

[27] R. Wilson et al. SUIF: an infrastructure for research on parallelizing and optimizing compilers. Technical report, Computer Systems Laboratory, Stanford University, Stanford, CA, May 1994.

[28] F. Charot and V. Messe. A flexible code generation framework for the design of application specific programmable processors. *Int. Symp. on Hardware/Software Codesign,* Rome, Italy, May 1999.

[29] O. Sentieys, J.P. Diguet, and J.L. Philippe. A high-level synthesis tool dedicated to real time signal processing applications. *European Design Automation Conf. (EURODAC '95),* Brighton, U.K., September 1995.

[30] R. Schreiber, S. Aditya, S. Mahlke, V. Kathail, B. Ramakrishna Rau, D. Cronquist, and M. Sivaraman. PICO-NPA: high-level synthesis of non-programmable hardware accelerators. Technical report HPL-2001-249, Hewlett-Packard Laboratories, Palo Alto, CA, 2001.

[31] M. Wan. Design methodology for low-power heterogeneous digital signal processors. Ph.D. thesis, University of California at Berkeley, Berkeley Wireless Design Center, 2001.

[32] J. Hauser. Augmenting a microprocessor with reconfigurable hardware. Ph.D. thesis, University of California at Berkeley, 2000.

[33] A. Fraboulet, K. Godary, and A. Mignotte. Loop fusion for memory space optimization. *Int. Symp. on Syst. Synthesis (ISSS '01),* Montreal, Canada, October 2001.

[34] R. David, D. Chillet, S. Pillement, and O. Sentieys. A compilation framework for a dynamically reconfigurable architecture. *Int. Conf. on Field Programmable Logic and Applications,* pp. 1058–1067, Montpellier, France, September 2002. Lecture Notes in Computer Science 2438.

[35] T. Ojanpera and R. Prasad. *Wideband CDMA for Third-Generation Mobile Communication.* Artech House Publishers, London, 1998.

[36] Xilinx. *VirtexE Series Field Programmable Gate Arrays.* Xilinx, San Jose, CA, July 2001.

[37] Texas Instruments. *TMS320C64x Technical Overview.* Texas Instruments, Dallas, TX, February 2000.

[38] Texas Instruments. *TMS320C6414/15/16 Power Consumption Summary.* Application report, spra811a, Dallas, TX, March 2002.

4

Macgic, a Low-Power Reconfigurable DSP

Flavio Rampogna
Pierre-David Pfister
Claude Arm
Patrick Volet
Jean-Marc Masgonty
Christian Piguet
CSEM SA

4.1 Introduction

Low-power programmable digital signal processors (DSP) can be found nowadays in a broad range of battery-operated consumer devices, such as MP3/CD/DVD players, or in the ubiquitous cellular phone. The trend being in the software implementation of more complex signal processing algorithms, very high-performance programmable DSP microprocessors having both a very high computational power capability and a very low-power consumption will be required in the near future to seamlessly implement such algorithms.

4.1.1 DSP Architectures Evolution

The first programmable DSPs were relatively simple microprocessors, specialized in the handling of very specific data formats: either fixed-point or floating-point, depending on their architecture [9,10,13,17,19]. These processors were very efficient in transferring data between the memory and their data processing unit, as well as in the processing of the data itself. The data processing unit was typically optimized for the handling of multiply-and-accumulate (MAC) operations between two data words read from two different memories. Memory accesses were indirect, and most DSPs supported modulo indirect addressing modes especially useful in convolutions or for implementing circular buffers. Sometimes, a special reverse-carry addressing mode was also available, and was useful for the reordering of the data in fast-Fourier-transform (FFT) computations. The address computation hardware was typically located into address generation units (AGUs). An AGU usually contains a set of specialized index, offset and modulo registers.

Historically, programmable DSPs implemented a very limited and specialized set of registers: temporary registers, accumulators, and AGU registers. This nonorthogonality of the architecture and the limited resources made these processors very difficult for a high-level language compiler to generate program code. To fully exploit the processing power and available instruction-level-parallelism (ILP) of these DSPs, they had to be programmed in assembly language, which very often was quite a tedious task.

In the last few years, the tendency has been to limit the number of specialized registers by implementing large sets of general-purpose registers that can be used as operands for most, if not all, instructions, and to have the hardware providing support for multiple data types [6,11,12,14,15,16,18]. The latest high-performance DSP architectures generally provide a very high data transfer bandwidth that can be exploited by a large number of parallel processing units. There are still different kinds of processing units, specialized for a given kind of processing. Several general-purpose ALUs are typically available, as well as a branch/loop unit, and specialized address generation ALUs. These recent architectures provide a relatively good hardware support for high-level language compilers. Code generation is indeed eased by the availability of multiple relatively basic parallel processing elements (ALUs), by the sufficiently large number of general-purpose registers, and by the support of standard data types (i.e., chars, integers, long integers, and floating-point).

In some modern DSPs, and to reduce both the power consumption and the hardware complexity of the circuit, the instruction-level parallelism (ILP) made available by these architectures is made explicit [12,16] (i.e., operations to be executed are grouped together into clusters, which are typically between 128 and 256 bits wide and may contain between 4 to 8 operations). Within a cluster it is sometimes possible to specify what are the operations that can be executed in parallel, and which ones need to be executed sequentially. The simplest approach, however, being to have all operations in a cluster executed in parallel and a direct and simple mapping between available hardware resources and operations coding in a cluster. In this approach, the scheduling of operations execution has to be explicitly specified by the programmer (or the compiler), and not chosen by the hardware, such as in superscalar architectures.

An alternative to programmable DSPs comes from configurable but nonprogrammable DSP coprocessors [7], optimized and specialized for the computation of a very specific signal processing task, such as FFT, FIR, IIR, Viterbi decoding, or image motion estimation computation. Such coprocessors may use the system's direct memory access (DMA) mechanisms to fetch the data needed to compute their algorithm, or implement their own memory address generation mechanisms. Future high-performance DSP systems may well include one or more of these coprocessors, together with one or more programmable DSPs or general-purpose microprocessors. Indeed, use of coprocessors may easily improve a system's performance by an order of magnitude or even more, by allowing very efficient parallel implementation of specific algorithms (e.g., Viterbi decoder or FFT computation).

4.1.2 Parallelism, Instruction Coding, Scheduling, and Execution

Today's high-performance DSP microprocessors can often execute up to eight different operations in parallel coded in a single instruction word (e.g., four MAC, two data address computations together with two memory accesses, one branch, and one bit manipulation operation). The packing of a large number of parallel operations into instruction word(s) can be performed using different approaches, leading to different DSP architectures.

A first possible approach consists in keeping all parallel operations as separate and independent instructions, and defining an instruction-set in which instructions are relatively small in terms of the number of bits required for the coding of an operation. The processor would read multiple instruction words from the memory at once and schedule their execution. Scheduling could either be automatically performed by the processor's hardware, as in a superscalar microprocessor, or predefined by the programmer or the compiler (Figure 4.1(a)). If the scheduling is predefined, the architecture is called static superscalar. Explicit scheduling information has to be encoded in an instruction's opcode (Figure 4.1(b)). The kind of explicitly provided scheduling information could be: Execute this instruction in parallel or in sequence with the previous instruction.

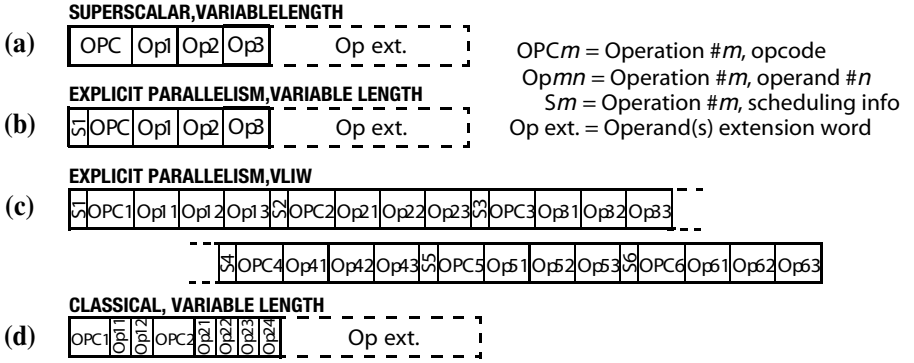


FIGURE 4.1 Parallelism and instruction coding.

A second, slightly different, approach may consist in the explicit coding of different parallel operations into a single very large instruction word (VLIW), typically of 128 bits or more. The processor fetches such a large instruction and executes all parallel operations contained in it. The operation execution scheduling is explicit and the simplest possible scheduling mechanism would be that all operations coded into the VLIW instruction are executed in parallel. Such an approach could prove to be quite memory wasting, however, especially in situations where low parallelism is actually available in a program. To solve this problem, a more advanced scheduling mechanism may be implemented by encoding some additional information on the need for parallel or sequential execution to each operation coded in a VLIW instruction (Figure 4.1(c)). By using this additional scheduling information, and by an appropriate ordering of operations within an instruction word, it is possible to fulfill any instruction execution scheduling needs, while still keeping an optimum code memory density because no-operations (NOPs) are not required to fill up VLIW instructions.

A third possible approach consists in using instruction words of relatively small size, typically between 24- to 64-bit, and by packing very few parallel operations in such words (Figure 4.1(d)). This is the approach that has originally been followed in the first programmable DSP microprocessors, but which is still widely used today [11,14,17,19]. In this approach, an instruction word consists of up to four different operations that are executed in parallel. It is indeed common to find DSP architectures allowing for the encoding of an ALU operation together with the encoding of two indirect memory accesses and address indexes updates operations in a single instruction word. The limitation of this approach, when applied to high-performance DSPs, is that these relatively small instruction words do not allow for the encoding of a very large number of parallel operations, therefore limiting the maximum available parallelism that can be programmatically exploited at the instruction level. To overcome part of this limitation it is possible to define operations that actually perform a unique computation on multiple data. These operations are of the single-instruction-multiple-data (SIMD) category [8]. For example, an MUL4 operation allows performing four multiplications on four pairs of data. As for VLIW instruction packing, here too, if little parallelism can be extracted from a program, the available parallel operations of an instruction word cannot be fully exploited and are to be replaced with NOPs, thus needlessly increasing program code size. By using variable-length instruction words, it is possible to reduce such a program memory occupation overhead, while unfortunately somewhat complicating the instruction fetch and decoding hardware's task.

4.1.3 High Performance for Low-Power Systems

Power consumption reduction in a system-on-chip (SoC) can be achieved at different levels, ranging from the careful analysis of an application's needs, to appropriate algorithms selection, as well as a good knowledge of precision requirements for the data to be processed. Selection of appropriate hardware architecture for implementing these algorithms, taking into account the DSP processor

core(s), the memory subsystem (i.e., internal/external RAMs, ROMs, and DMA availability) and the data acquisition chain (i.e., ADC/DAC, and I/Os), may allow a significant power reduction [20]. In addition, the use of an appropriate semiconductor technology together with a good trade-off between the hardware's computational power, and operating voltage and frequency may allow for a large power consumption reduction.

Modern high-performance DSPs and general-purpose microprocessor circuits and systems usually implement multilevel memory access hierarchies: typically, two cache-memory levels followed by a high-speed internal RAM or external (S)DRAM memory containing data and instructions. The memory is usually seen as unified for the programmer; data and instructions can be intermixed. Internally, however, the DSP uses multiple independent memory busses that actually implement distinct memory spaces. Generally, a DSP implements a program memory bus and one or two data memory busses. Each memory bus is typically connected to a specific level-1 (L1) cache memory. Unification of memory spaces may occur after the L1 cache or after the level 2 (L2). In very low-power systems, where memory needs and maximum operating speed is not very high, caches can be avoided and the memory spaces may remain distinct. This, together with a simpler hierarchical memory subsystem may help reduce power consumption. The power consumption of the memory subsystem can quite easily be reduced simply by placing the most often accessed data into smaller memories, and the less often accessed data in larger ones because smaller memories are faster and consume less energy per access than larger ones.

4.1.4 DSP Performance and Reconfigurability

With the increasingly high costs for accessing advanced semiconductor technologies, and with implementation of more computationally demanding signal processing algorithms, a modern SoC implementing programmable DSP(s) should be as efficient and generic as possible to allow for the implementation of the larger possible number of applications. Therefore, a DSP core has to be as power efficient as specialized hardware, and be retargetable to different algorithms and applications without any significant loss of performance. Fortunately, in some applications, performance vs. power-consumption vs. reconfigurability goals may be met by the implementation of an appropriate programmable signal processing architecture. For example, a system's computational performance may be increased by allowing multiple parallel processing units to compute an algorithm on different data, different parts of an algorithm on pipelined data, or different algorithms on identical or different data. The maximum achievable parallelism depends on the algorithms to be executed and available hardware resources: processors, coprocessors, and memories.

Power consumption of a system can be reduced by appropriate selection of power supply voltage, operating circuit frequency, and available parallelism. Indeed, by increasing the execution parallelism, the timing constraints are relaxed and the circuit's operating frequency can be decreased, which allows to lower its operating voltage and therefore to reduce its dynamic power consumption.

If an SoC system has to be reconfigured to support a new application, or multiple applications, reconfiguration can be achieved at various levels, the main being at the program code level by the programming of a new application's algorithms. The program has then to be stored into a reprogrammable memory, such as an EEPROM. Sometimes, an external serial EEPROM chip can be used to initialize the content of an internal RAM at reset-time, and thereby allowing the configuration of the system. Additional reconfiguration levels are obtained when the program actually reconfigures the SoC's hardware: coprocessors, direct memory access (DMA) hardware, peripherals, or even the actual DSP processor [1,3,4,5].

The runtime reconfiguration of a programmable DSP core may be achieved in different ways. The Macgic DSP architecture allows the programmer to reconfigure and use a small set of extended instructions, which allow a fine-grain control over specific datapaths: the address generation unit (AGU) and the data processing unit (DPU) datapaths. This fine-grain control increases the programmatically exploitable hardware parallelism. Such extended instructions are typically used in algorithm's kernels to significantly speed up their execution, while keeping the program code density performance of the DSP at a good level.

4.2 Macgic DSP Architecture

4.2.1 General Architecture

Macgic is a low-power programmable DSP core for SoC designs. It can be used as a stand-alone DSP, or as a coprocessor for any general-purpose microprocessor or DSP. It has been designed to be efficient for a broad range of DSP applications. This is done by providing the designers with the possibility to tailor some Macgic features to best fit an application class (e.g., audio, video, or baseband radio). In particular, the various specifiable word sizes (e.g., data and address), the DSP modularity, and the instruction-set specialization are key features allowing Macgic to be very efficient both in terms of processing speed and energy consumption. This customization of the DSP must be performed before hardware synthesis.

Figure 4.2 presents the Macgic DSP architecture. The DSP core is made of four distinct operating units, each playing a specific role in the architecture: The program sequencing unit (PSU) handles branches, exceptions, and instruction fetch. The host and debug unit (HDU) handles data transfers with a host microprocessor and the debugging of Macgic programs through a specific debugging bus. The data move unit (DMU), containing the X and Y address generation units (AGU), handles data transfers between registers and between registers and external data memories. The data processing unit (DPU) handles the processing of the data.

Macgic uses relatively small (32-bit) instruction words. The data word size can be freely specified (e.g., dw = 12 to 32 bits) before synthesis. The DSP implements two distinct data memory spaces (X, Y). Concurrent accesses to these two memory spaces are supported. Up to four data words per memory space can be transferred between the DSP and the external memory per clock cycle. Macgic is a load/store architecture [8] and implements two banks of eight wide general-purpose (GP) registers, one bank per memory space. A wide register can store four data words. A GP register can be accessed either as a single data word, as half-wide, or as wide data words. Data processing operations can access up to 16 data words per clock cycle, from up to four wide GP registers, two per data space. The program and data address space sizes can be independently specified (paw, daw = 16 to 32 bits) before synthesis.

Complex addressing modes are made available by the two customizable and software reconfigurable address generation units (AGUs). The data processing unit (DPU) can also be customized and specialized before synthesis. Extended operations of the DPU can be software reconfigured. An HDU allows the control of Macgic from a host microprocessor or from a remote software debugger. The HDU also allows the exchange of data with the host microprocessor through specific data transfer FIFOs and registers.

4.2.2 Program Sequencing Unit

The program sequencing unit (PSU) is responsible for handling the instruction fetch, the global instruction decoding and the execution of branches, subroutine calls, hardware loops, and exceptions. This unit handles external interrupt requests as well as internal software exceptions. Eight prioritized and vectorized external interrupts requests lines are available to an external interrupt controller.

An external hardware stack stores the return address (of subroutines, exceptions) and the loop status when needed. The number of hardware loops, subroutines and interrupts that can be nested is given by the size of the hardware stack, which is a customization parameter.

The PSU contains eight 16-bit flag registers (IN, EX, EC, PF, HD, DM, PA, and PB). Two of these registers are actually controlled by the DPU (PA and PB), one by the DMU (DM), one by the HDU (HD) and the remaining ones by the PSU. DPU flags are typically the Z, N, C, and V flags of each ALU.

The PSU handles the conditional execution of operations in a manner similar to the conditional branches (i.e., operations are executed or not depending on the value of a flag taken from one of the eight flag registers). It implements hardware loops and instruction repetition mechanisms. Hardware loops automatically handle the iteration counting and branching to the beginning of the loop at the end of an iteration. Only one clock cycle is necessary to initialize a loop and there are no additional clock cycles penalties during its execution. Hardware loops can help reduce the clock cycles count of an

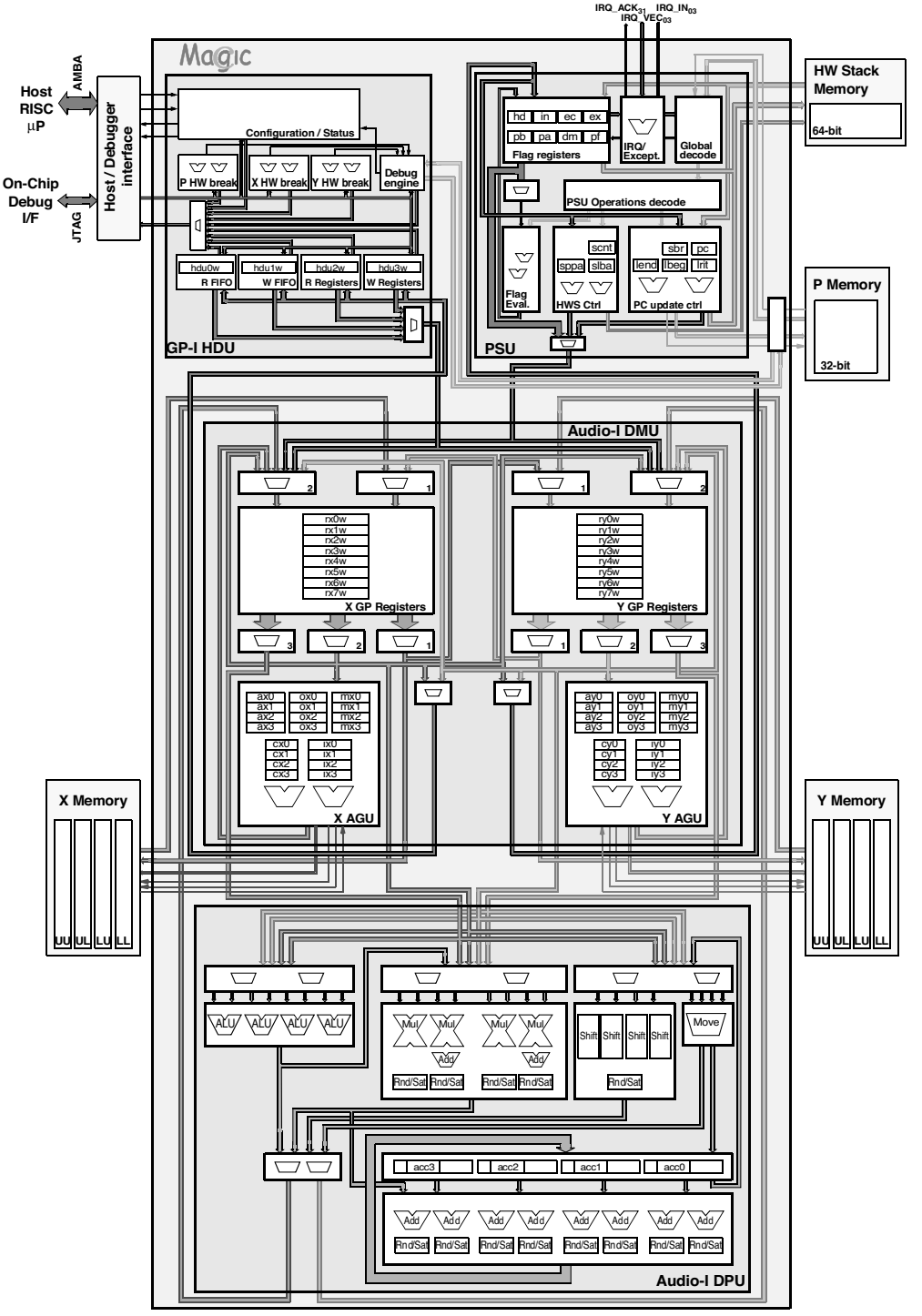


FIGURE 4.2 Macgic DSP architecture.

algorithm's execution in a significant manner, particularly for small loops that are iterated a large number of times. In instruction-repeat operations, the instruction to be repeated is fetched only once from the program memory, thus saving unnecessary, power-consuming, program memory accesses.

4.2.3 Data Move Unit

The DMU implements the data transfer mechanisms of the Macgic DSP. Data can be transferred between the DMU and the external memory, as well as between the DMU and the other units: DPU, HDU, and PSU. All data transfers use at least one GP register, either as a source or as a destination register.

The large number of data busses between the DMU and the DPU allows a very high data transfer bandwidth between these units: up to 16 data words can be transferred from the DMU to the DPU per clock cycle, and up to 8 data words from the DPU to the DMU.

Two address generation units (AGUs) are available in the DSP: one per data memory space. The AGUs are used to generate addresses for data memory accesses. Each AGU has four index register sets. In addition to the traditional base address, offset, and modulo registers, the configuration and extended-instruction registers allow the configuration and customization of the AGUs to best fit the targeted algorithms memory addressing needs. The two independent AGUs allow concurrent accesses to the two memory spaces.

4.2.4 Data Processing Unit

The data processing unit (DPU) implements the data processing capabilities of the Macgic DSP. Because the DSP architecture is modular, new DPUs can be developed and specialized to obtain the best possible performance for the class of algorithms to be executed. The first implementation of this unit has been a general-purpose one but slightly specialized towards audio processing. This first Audio-I DPU implements four ALUs, four multipliers, four shifters, together with four accumulator registers and their associated adders. The accumulator registers (width: 2 dw + 8 bits) allow storing the results of multiply-and-accumulate operations.

The DPU can handle data either as fixed-point, signed, or unsigned integer, depending on the operation selected. It implements round-to-nearest rounding, and a saturation mechanism can be enabled to ensure accurate computations.

4.2.5 Host and Debug Unit

The HDU is the link between Macgic and an external host microprocessor or a software debugger. This unit implements both data transfers and software debugging mechanisms.

The host microprocessor or debug interface access the HDU through the host/debug bus. A set of registers is then available and allows configuring and controlling the HDU. For this bus, the HDU acts as the slave and the host microprocessor or debug interface as the master.

The HDU allows the transfer of data between a host/debug bus master and the HDU registers. For this, two FIFOs are available, one per data transfer direction, as well as two groups of four registers, one per data transfer direction. The depths of the FIFOs are customizable. Flow-control mechanisms have been implemented. Writing or reading data into or from the FIFOs or registers can, for example, trigger the generation of an event to the bus master, or of an exception to Macgic.

In addition to data transfer mechanisms, the HDU implements a set of hardware breakpoints engines, one per memory space. Each engine allows monitoring accessed memory addresses and is able to generate a breakpoint if either an address range or a single address matches a given memory access kind: read, write, read or write.

The hardware breakpoint engines use the HDU debug engine to actually implement the breakpoint. The debug engine allows controlling the Macgic program execution. It allows to stop the DSP, execute instructions step-by-step, insert instructions in the DSP pipeline, access the program memory (e.g., to

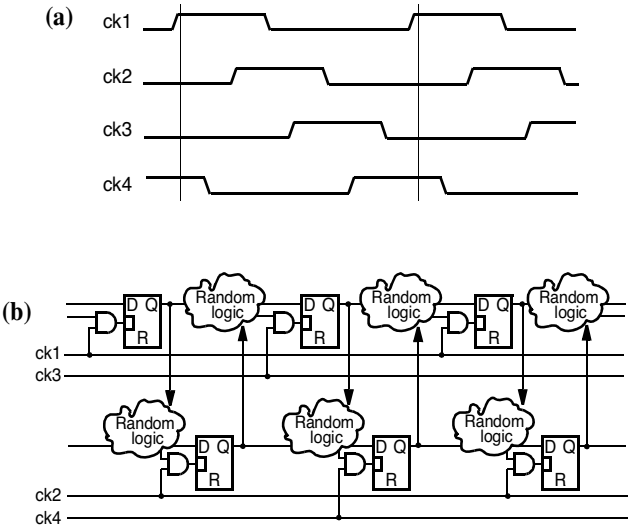


FIGURE 4.3 (a) Macgic DSP clock signals; (b) clock gating and pipeline.

place software breakpoints instructions or to download a program), or to get information on the DSP processor state.

4.2.6 Clocking Scheme

The Macgic DSP core uses four clocks signals ck1 to ck4. These signals must be nonoverlapping by pairs (i.e., ck1 must not overlap ck3, and ck2 must not overlap ck4). The DSP uses latches as data storage elements instead of flip-flops. The Macgic DSP uses these two pairs of nonoverlapping signals (ck1/ck3 and ck2/ck4) to implement the various pipeline stages and clock gating. Figure 4.3(a) depicts the partial overlapping of the four clock signals. Figure 4.3(b) illustrates how the level-sensitive storage elements implement the pipeline stages as well as clock gating. Clock-gating signals are generated either from signals latched during the previous clock phase or from two clock phases before the clock phase they are supposed to enable. They must be stable before the activation of the clock signal they enable.

The appropriate use of level-sensitive storage elements [2], makes the hardware less sensitive to clocks jitter than by using edge-sensitive storage elements, therefore allowing to implement more robust circuits, capable of working under very extreme operating conditions (e.g., voltage, temperature, and technology corner). With this approach, any trade-off between maximum clock frequency and power consumption related to jitter minimization in clock distribution trees can be made, without actually compromising the good working of the circuit — only the achievable maximum operating frequency.

The large number of clock phases enables a finer control over the pipeline, the clock gating mechanism, and simplify the generation of clean I/O data and control signals on the various DSP's external busses.

4.2.7 Pipeline

To simplify the description of the pipeline, the following clock-phase notation *c.n* is used. Where *c* represents the clock cycle number ($c = 1..x$) and *n* the clock phase number ($n = 1..4$). When $n = 1$, it means that the clock signal ck1 is asserted, when $n = 2$ that clock signal ck2 is asserted, and so on.

The Macgic DSP has been targeted to very lower-power applications. To keep both the design complexity and the power consumption to acceptable levels, the DSP pipeline depth has been made relatively short. Figure 4.4 depicts the various pipeline stages. Most instructions are typically executed in only three clock cycles.

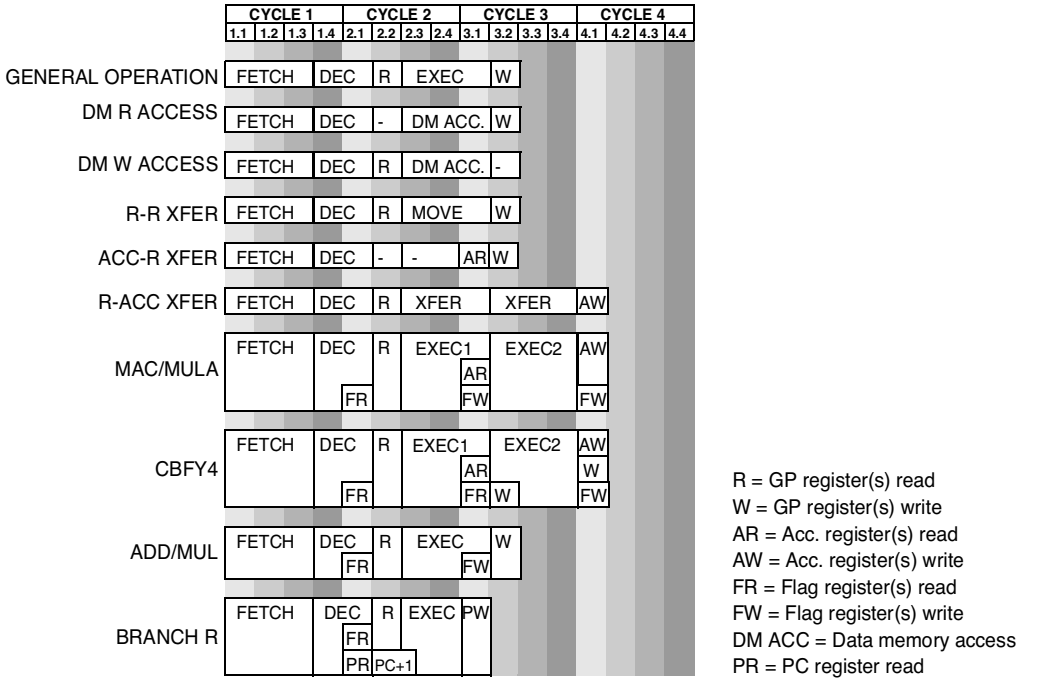


FIGURE 4.4 Macgic DSP pipeline (Audio-I DMU, Audio-I DPU, GP-I HDU).

In the Macgic DSP, the PC, the flag registers, and the accumulators are updated during phase 1 of each clock cycle, and the GP registers during clock phase 2. The program memory is accessed during phases 2 and 3 of each clock cycle, and the data memory during phases 4 and 1, while the hardware stack memory is accessed during the clock phase 3.

The delay between the reading and the writing-back of a register is typically of one clock cycle. This makes the pipeline transparent to the programmer, which greatly eases assembly-language programming. Only branches and a few instructions that are executed in four clock cycles need a special attention by the programmer. Branches necessitate a one-cycle delay slot (i.e., the instruction immediately following a branch is always executed). The instructions that write a result in a GP register with an additional latency cannot be immediately followed by instructions exploiting such a result. Unrelated operations or NOPs need to be inserted for the duration of the latency, before the result can actually be exploited.

One instruction is fetched per clock cycle, except if the pipeline has to be stalled by a program or data memory access wait-state, which delays the fetching and execution of subsequent instructions. Customized DPU/DMU or HDU instructions may, if needed, request a stall of the pipeline or a delaying of exception handling. The PSU handles the fetching and partial decoding of instructions. Fetched instructions are first partially decoded in the PSU, and the category of the operation(s) is determined. Then, the operation is dispatched to the appropriate unit for further decoding and execution. The PSU is not actually fully aware of the whole DSP instruction-set and pipeline. Completely independent and arbitrarily long execution pipelines can therefore be implemented in the DMU, DPU, and HDU. The pipeline can therefore vary from one implementation of a given unit to another implementation of the same unit. (e.g., short pipeline fixed-point hardware is implemented in one unit vs. long pipeline floating-point DPU hardware in the other one).

4.2.8 Instruction-Set

Macgic DSP instructions are 32 bits wide. This relatively small instruction size helps keeping the program memory power consumption to acceptable levels. A 32-bit instruction word fits one or two operations.

First operation		Second operation (executed in parallel with first operation)	
PSU-L	PSU Long operations (jumps, subroutine call, loop, etc.)	None	
PSU-M	PSU flag move operations	None	
DMU-L	DMU long operations (move immediate, direct memory)	None	
DPU-L	DPU long operations	None	
HDU-L	HDU long operations	None	
PSU-C	PSU conditional execution operation	**PSU-S**	PSU short operations (register-register transfers)
		PSU-M	PSU flag move operations
		DMU-S	DMU short operations (register-register transfers)
		DPU-P	DPU parallel operations
		DMU-P	DMU parallel operations (two indirect memory accesses)
PSU-P	PSU parallel operations (flag clear/set/invert)	**DMU-S**	DMU short operations
		DPU-P	DPU parallel operations
		DMU-P	DMU parallel operations
PSU-S	PSU short operations (HW stack PUSH/POP, reg. move)	**DMU-P**	DMU parallel operations (registers data move, indirect memory accesses)
DMU-S	DMU short operations (registers data move)		
DPU-P	DPU parallel operations		
HDU-P	HDU parallel operations		

FIGURE 4.5 Macgic instruction word operations categories coding.

Macgic DSP operations are split into several categories. Operations contained in an instruction word are executed in parallel. Figure 4.5 illustrates the available instruction-level parallelism. Additional parallelism can further be encoded within an operation (e.g., extended, SIMD, vectorial, or specialized operations).

PSU operations are "built-in" and cannot be customized nor removed from the instruction-set of the Macgic DSP. Hardware support is provided for nested hardware loops and instruction repeat.

Branches can be either direct or indirect. In case of indirect addressing, either GP registers or the software branch register (SBR) can be used as index. Program memory addressing can be either absolute or PC-relative. Branches can be conditional. The condition is the value of a flag. There are operations for handling and processing flags. Flags can be set, cleared, inverted. A Boolean expression evaluation can take expressions of up to three flags as operands, perform AND/OR/XOR operations on them and save the Boolean result into a flag in the PF flags register.

The Audio-I DMU makes a comprehensive set of data move operations available. These data transfers can move either single or wide data words. Up to two, parallel wide registers data can be moved into two wide GP registers in a single DMU-S or P operation. Up to two, wide data words can be transferred between the two memory spaces and two GP registers in a single DMU-P operation. In addition to single-word or wide data moves, half-wide or word-specific data moves are available. Immediate data moves are also available.

The Audio-I version of the AGU implements three types of indirect data memory access operations basic, predefined, and extended. Basic operations implement a simple set of very common DSP addressing operations. Up to three predefined operations can be configured for each index register through the appropriate programming of a configuration register. Predefined operations allow access to more powerful addressing modes than the ones made available by the basic operations. Extended operations further extend the complexity of addressing modes and operations that the AGU can perform. Up to four extended operations are available per AGU.

The actual operation performed by an extended operation is configured through an extended operation register. Extended operations may help reducing the number of clock cycles necessary to compute a specific address computation, potentially saving precious clock cycles in key parts of time-consuming algorithms.

The DPU is responsible for the processing of the data in Macgic. This unit can be customized to best fit the targeted class of algorithms and application. Two categories of DPU operations are available: DPU-P and DPU-L operations. DPU-P operations can be executed in parallel with PSU-P or DMU-P operations. In the Audio-I DPU, four kinds of DPU-P operations have been implemented. Standard DPU operations, such as the classical MAC, ADD, SUB, MUL, CMP, AND, OR, and XOR, are available. Computations on complex numbers are also supported. Use of SIMD operations, such as MAC4, ADD4, SUB4, and MUL4, may speed-up the computation performance by a factor up to four. The same is true for vector-oriented operations such as MACV, MSUBV, ACCV, MINV, and MAXV, which usually take multiple input values and compute a single result in a single clock cycle. Specialized or customized operations allow for the speed-up of some targeted algorithms. In the Audio-I DPU, special instructions for FFT computations, IIR and FIR filtering, function interpolation, bit-stream creation and decoding, min and max searches, and data clipping have been implemented. As an example, in a baseband-oriented DPU, specialized operations for the implementation of Viterbi or turbo decoders can easily speed up the algorithm's performance from a factor 2 to > 30 over classical software implementation of such algorithms, depending on the additional hardware used to implement such specialized operations. Audio-I DPU-L operations allow performing two independent DPU operations in parallel (e.g., four MUL and four ADD). More than 170 data processing operations have been implemented in the Audio-I DPU. This extensive number of operations can be further completed/customized, if needed, to better match application-specific algorithms needs. As for the AGU, if a high level of parallelism is needed, and heterogeneous data processing operations should be executed in parallel, a limited set of reconfigurable extended DPU operations can be made available.

A few examples of Macgic Audio-I instructions are given next.

```
irepeat 16
   mac4     acc,rx0w,ry3w || movpx2p rx0w,(ax2, pr0) ry3w,(ay1,iy2)
cmacc acc0,acc1,rx0w.l,ry0w.l || movb2p rx0w,(ax0)+ ry0w,(ay0)+
loop ry7, end_radix4_fft_loop
   cbfy4a0 acc,ry0w.l                              || movpxp rx2w,(ax0,pr0)
   cbfy4a1 rx0w,acc,rx1w.l,ry1w.l,rx5w.l,ry4w.l || movpxp rx3w,(ax0,pr1)
   cbfy4a2 rx0w,acc,rx2w.l,ry2w.l,rx5w.u,ry4w.u || movpxp (ax1,pr0),rx0w
   cbfy4a3 rx0w,ry5w.l,rx4w,ry4w,acc,rx3w.l,ry3w.l,rx4w,ry5w.l || movpxp (ax1,pr0),rx0w
end_radix4_fft_loop:
```

The irepeat operation allows repeating the execution of the next instruction the given number of times. The cmacc operation takes the complex conjugate of the second operand, performs a complex multiplication, and accumulates the complex result into the specified accumulators. The loop operation allows repeating a specified sequence of operations for a given number of times. The cbfy4 operations are specialized instructions for FFT/IFFT computation. The various movpxp and movbp operations are data move operations that usually perform data memory accesses, or just index registers updates when no source or destination registers are specified.

4.3 Macgic DSP Reconfiguration Mechanisms

Two reconfiguration mechanisms have been developed for the audio versions of the AGU and DPU. Both mechanisms use a similar principle. Given the relatively small instruction word of the Macgic DSP (32-bit), the degree of ILP available to the DSP programmer may be relatively limited, unless the powerfulness of SIMD, and of specialized operations can be exploited. To provide the programmer with additional programming capabilities, a set of extended, software reconfigurable, DMU-P and DPU-P operations are made available by the AGUs and the DPU.

4.3.1 Address Generation Unit Reconfiguration

Each AGU permits the reconfiguration of four extended operations. An extended operation allows to both perform an address computation, to access the data memory using an indirect addressing, and to

save address computation results into up to three AGU registers. In a DMU-P operation, two 3-bit fields, one per AGU, specify the kind of operation to be performed by the AGU: either a predefined AGU operation or an extended AGU operation. Examples of use are:

```
macv acc,rx0w,ry3w || movpx2p (ax2, pr0),rx0w (ay1,iy2)
clra acc
irepeat 16
    macv acc,rx1w,ry2w || movpx2p rx1w,(ax2, ix0) ry2w,(ay1,iy3)
```

The programmer shall specify the use of an extended operation by typing the extended operation register I$s$$n$ (s = X,Y, n = 0..3) after the index register. Predefined operations are specified in a similar manner, by typing PRm (m = 0..2) after the index register to be used. The mnemonic for predefined and extended operations is movpxp or movpx2p.

As an example, a single extended AGU operation may perform the following computations in parallel:

```
rx2w <= XDM[ax1+ox3], ax2<=(ax0+ox3)%mx1, ox1<= ox2+mx3
```

or

```
ax1 <= ClearLSBs(ax2,ox3), ox2<=ox2>>2, mx2<= mx2>>2
```

Figure 4.6 illustrates the various parts of an AGU datapath and its reconfiguration capability. Direct addressing operations, as well as basic and predefined indirect addressing operations are internally remapped into extended operations.

Figure 4.6(a) illustrates the selection of two address registers, two offset registers, and two modulo registers that will be read. The first address, offset and modulo register can either be selected by the value specified in the instruction's opcode, or by a value specified in the extended operation' configuration register. The second address offset and modulo registers can independently be selected from the configuration register. The six values read from the AGU registers are made available to two ALUs. The first one (Figure 4.6(b)) is responsible for computing the actual address to be accessed in the data memory, while the second one (Figure 4.6(c)), which allows more complex operations to be performed is rather used to compute post-modified addresses that will be used the next time a memory access will be performed. The results coming from these two ALUs can be input to a third ALU (Figure 4.6(d)) for further post-processing. Table 4.1 lists the various operations available for each of the three ALUs present in an AGU. Up to three results can be saved into the AGU registers: one in an address register, one in an offset register, and one in a modulo register (Figure 4.6(e)). Figure 4.6(f) gives the format of an extended AGU operation configuration register. Four of such registers are available in each AGU, one per extended operation.

4.3.2 Data Processing Unit Reconfiguration

Eight extended DPU operations are made available by the DPU. An extended DPU operation configuration word is 128-bit wide. An extended operation is invoked through a specific DPU mnemonic: dpxop.

```
dpxop n,Value1,Value2
```

The first operand of the extended operation is the operation number (0..7). Two 3-bit operand values can then be specified. The meaning of these additional operands depends on the configuration of the extended operation. Figure 4.7 presents the configuration possibilities.

Up to four wide GP registers can be read from the DPU. Figure 4.7a illustrates how the GP registers are selected. The values read from the registers are made available as four values XR1, XR2, YR1, and YR2. Figure 4.7b illustrates the four virtual processing units. The operations that these virtual units can perform are given in Table 4.2. The source operands for each operation can be freely chosen from any of the four wide values XR1, XR2, YR1, and YR2. Each virtual operation can be conditionally executed, using a flag value as condition. The result of each virtual operation can be saved into any of the two wide

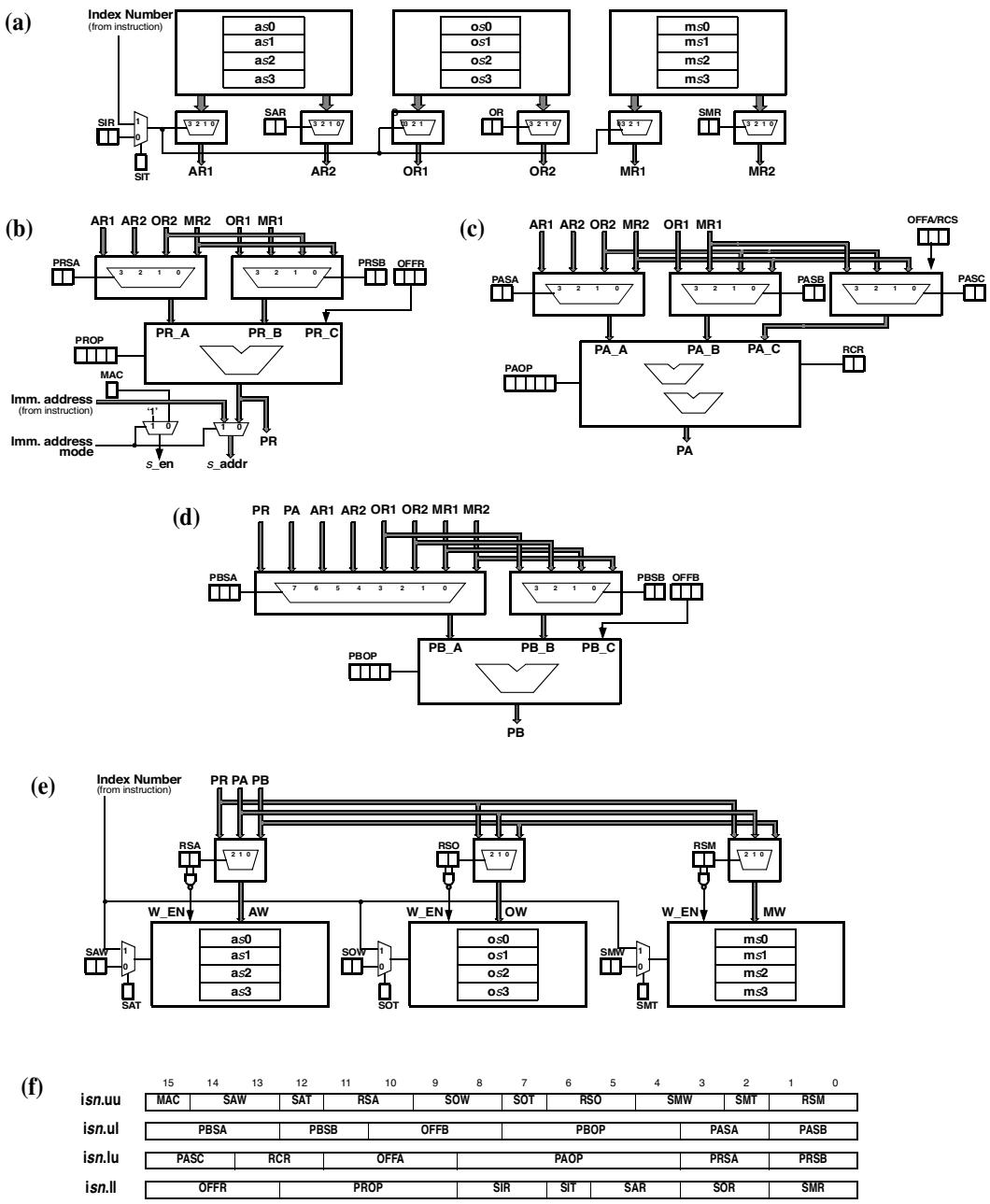


FIGURE 4.6 AGU datapath: (a) AGU registers read ports; (b) premodified address generation; (c) post-modified A address computation; (d) post-modified B address computation; (e) AGU registers write ports; (f) extended AGU operation configuration.

output values XW, YW. The two wide output results can be saved back into GP registers. Figure 4.7c illustrates the GP registers' write-selection mechanism. Sixteen 64-bit extended operation registers are made available to the programmer, two per extended operation. Figure 4.7d is an extended DPU operation configuration register. Eight of such registers are available, one per extended operation.

As an example, a single extended DPU operation can perform the following computations in parallel, in a single clock cycle:

TABLE 4.1 AGU Datapath, Example of Address Generation Operations

Value	PROP	PAOP	PBOP
0	A	C	C
1	B	-C	-C
2	A+B	A+C	A+C
3	A-B	A-C	A-C
4	B-A	B+C	B+C
5	A+OFFR	B-C	B-C
6	B+OFFR	A+B	A+B
7	A+(B>>1)	A-B	A-B
8	A-(B>>1)	B-A	B-A
9	A+(B>>2)	B<<(C AND 7)	A<<(C AND 7)
A	A-(B>>2)	B>>(C AND 7)	A>>(C AND 7)
B	A+(B<<1)	ClearLSBs(A,B)	ClearLSBs(A,B)
C	A-(B<<1)	RevCarryInc(A,RCR,C AND 7)	Reserved
D	A+(B<<2)	RevCarryInc(B,RCR,C AND 7)	Reserved
E	A-(B<<2)	(A+B)%C	Reserved
F	OFFR	(A-B)%C	Reserved
10	-	(A+C)%B	-
11	-	(A-C)%B	-
12	-	A+(B>>(C AND 7))	-
13	-	A-(B>>(C AND 7))	-
14	-	B+(A>>(C AND 7))	-
15	-	B-(A>>(C AND 7))	-
16	-	A+(B<<(C AND 7))	-
17	-	A-(B<<(C AND 7))	-
18	-	B+(A<<(C AND 7))	-
19	-	B-(A<<(C AND 7))	-

```
if (pf7) {//execute if flag 7 of PSU flag register PF is set
    Rx2w.uu <= rx7w.uu * rx6w.uu; Rx2w.µl <= rx7w.µl * rx6w.µl;
    Rx2w.lu <= rx7w.lu * rx6w.lu; Rx2w.ll <= rx7w.ll * rx6w.ll;
}
if (npf3) {//execute if flag 3 of PSU flag register PF is cleared
    Ry5w.uu <= ry3w.uu + ry2w.uu; Ry5w.µl <= ry3w.µl + ry2w.µl;
}
if (pa1) {//execute if flag 1 of DPU flag register PA is set
    Ry5w.lu <= rx7w.lu>>2; Ry5w.ll <= rx7w.ll>>2;
}
if (npa1) {//execute if flag 1 of DPU flag register PA is cleared
    Ry5w.lu <= rx7w.lu; Ry5w.ll <= rx7w.ll;
}
```

4.4 Performance Results

Some computation performance results of the Audio-I version of the DMU and DPU are given in Table 4.3. The results are expressed in clock cycles and are given for the algorithms' kernels. Additional clock cycles overhead may exist if these algorithms are to be placed into subroutines.

The complexity of the Audio-I version of the DSP in number of transistors count is about 750 k for a dw = 24-bit implementation, and about 550 k for a dw = 16-bit implementation.

The simulated power consumption of the Macgic Audio-I DSP (dw = 24-bit) is about 1.2 mW/MHz at 1.8 V in the TSMC 0.18-µ technology, and only about 0.25 mW/MHz at 0.9 V (e.g., CSEM's CSELIB 6 standard cells library and Synopsys' power compiler).

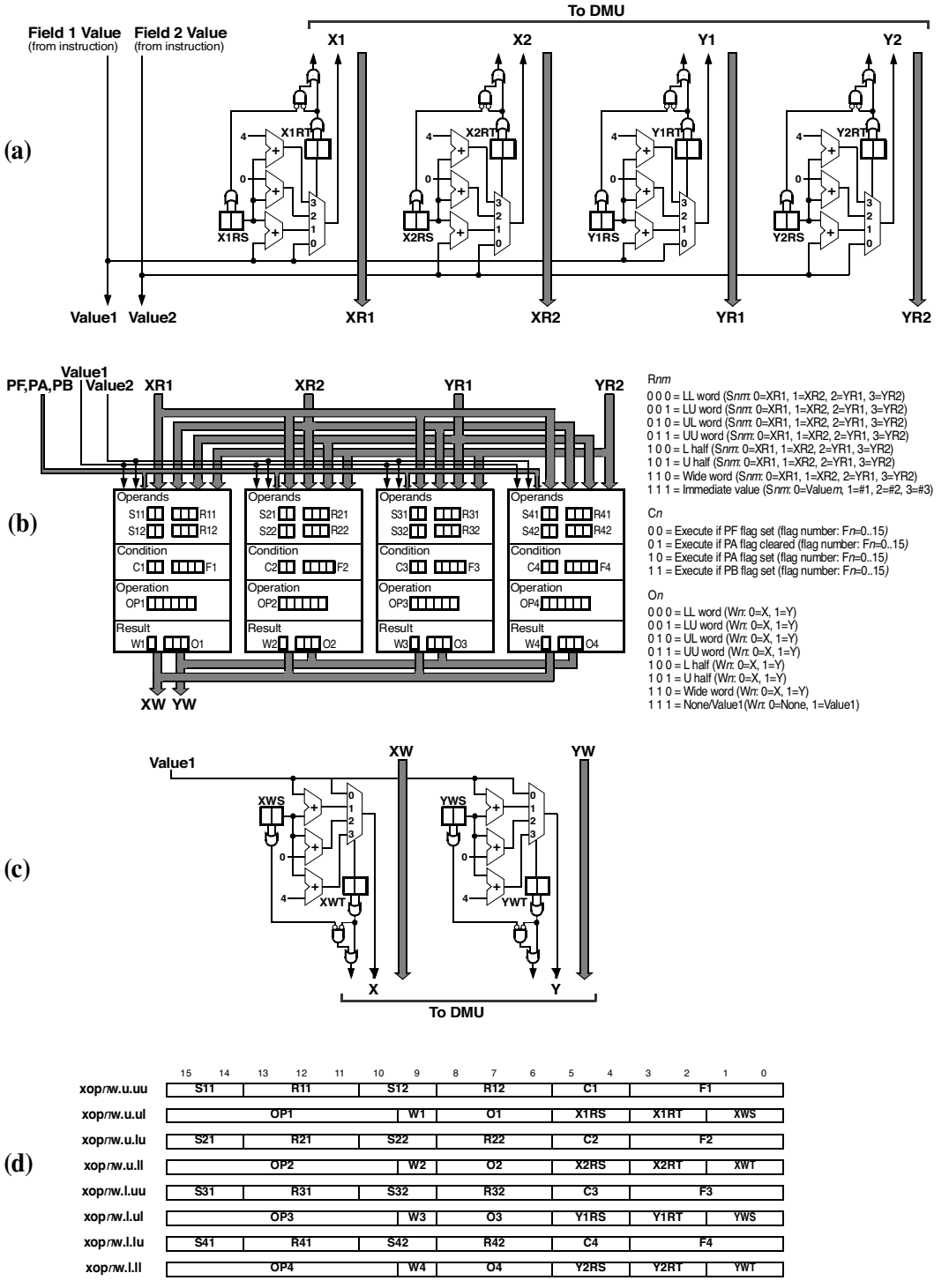


FIGURE 4.7 Extended DPU operations virtual datapath: (a) GP registers read ports; (b) extended DPU operations; (c) GP registers write ports; (d) extended DPU operation configuration.

TABLE 4.2 Example of Extended DPU Operations

Value	OP1	OP2	OP3	OP4	Value	OP1	OP2	OP3	OP4
00	NOP	NOP	NOP	NOP	20	SHR	SHR	SHR	SHR
01	ABS	ABS	ABS	ABS	21	SHRU	SHRU	SHRU	SHRU
02	ACCA	ACCA	ACCA	ACCA	22	SUB	SUB	SUB	SUB
03	ADD	ADD	ADD	ADD	23	SUBA	SUBA	SUBA	SUBA
04	ADDA	ADDA	ADDA	ADDA	24	SUBC	SUBC	SUBC	SUBC
05	ADDC	ADDC	ADDC	ADDC	25	SUBN	SUBN	SUBN	SUBN
06	ADDN	ADDN	ADDN	ADDN	26	ADD2	ADD2	SHL2	SHL2
07	AND	AND	AND	AND	27	MUL2	MUL2	MOV2	MOV2
08	CHKB	CHKB	CHKB	CHKB	28	SUB2	SUB2	SHR2	SHR2
09	CLRA	CLRA	CLRA	CLRA	29	MULS2	MULS2	SWAP2	SWAP2
0A	CLRB	CLRB	CLRB	CLRB	2A	ADDC2	ADDC2	SHRU2	SHRU2
0B	CMP	CMP	CMP	CMP	2B	MULU2	MULU2	SEL2	SEL2
0C	CMPA	CMPA	CMPA	CMPA	2C	SUBC2	SUBC2	SHLW	WCONC1
0D	CMPM	CMPM	CMPM	CMPM	2D	MULF2	MULF2	SHRW	WCONC2
0E	CMPU	CMPU	CMPU	CMPU	2E	MAX	MAX	BSEX	WCONC3
0F	DSQA	DSQA	DSQA	DSQA	2F	MAXU	MAXU	BSEXF	WCONCU
10	EXOR	EXOR	EXOR	EXOR	30	MAXM	MAXM	BSEXU	WCONCL
11	INVB	INVB	INVB	INVB	31	MIN	MIN	BSINSF	WCONCUS
12	MAC	MAC	MAC	MAC	32	MINU	MINU	BSINSF	WCONCLS
13	MOV	MOV	MOV	MOV	33	MINM	MINM	BSRD	WINS
14	MSUB	MSUB	MSUB	MSUB	34	CMUL	CMUL	BSRDF	WEXTR
15	MUL	MUL	MUL	MUL	35	CCONJ	CCONJ	BSRDU	TRSPL
16	MULA	MULA	MULA	MULA	36	CNORM	CNORM	EXTRAR	TRSPU
17	MULF	MULF	MULF	MULF	37	ACCV	MACV	EXTRA	CNTRMSB
18	MULS	MULS	MULS	MULS	38	CMP2	CMP2	CNTRMSB	CNT0MSB
19	MULU	MULU	MULU	MULU	39	CMPM2	CMPM2	CNT0MSB	CNT0LSB
1A	NEG	NEG	NEG	NEG	3A	CMPU2	CMPU2	CNT0LSB	MOV4
1B	NOT	NOT	NOT	NOT	3B	ADD4	MUL4	SHL4	SWAPHL
1C	OR	OR	OR	OR	3C	SUB4	MULS4	SHR4	SWAPIHL
1D	SEL	SEL	SEL	SEL	3D	ADDC4	MULU4	SHRU4	SWAPE
1E	SETB	SETB	SETB	SETB	3E	SUBC4	MULF4	CLR2	CLR4
1F	SHL	SHL	SHL	SHL	3F	Reserved	Reserved	Reserved	Reserved

TABLE 4.3 Macgic DSP Performance Results (Audio-I DMU, DPU, GP-I HDU)

Algorithm	Number of Clock Cycles (kernel)
Vector SUM (vector size = N)	~N/4
Vector addition	~N/2
Vector multiplication	~N/2
Dot product	~N/2
Vector normalization	~N/4
Minimum/maximum of a vector	~N/4
Matrix multiplication (matrix size = N × M)	~(N*M)*(N/4+2)
Matrix transportation	~N*M*(5/16)
Convolution (or FIR filter)	~1/4 per tap
Complex convolution	~1 per tap
Complex convolution, with conjugate of second operand	~1 per tap
IIR (biquad)	1 per stage
Complex FFT/1FFT radix 4 (size 64, optimized)	~220
Complex mixed-radix FFT/1FFT (size 2048, not optimized)	~14.6 k
Complex mixed-radix FFT/1FFT (size 4096, not optimized)	~29.1 k
Complex mixed-radix FFT/1FFT (size 8192, not optimized)	~66.2 k
MP3 decoder (not optimized)	~340 per stereo sample

4.5 Conclusions

Macgic has been developed in VHDL language and can therefore be synthesized to virtually any CMOS technology. It can provide a huge computational power and can be tailored to best suit an application's needs. Power consumption has been minimized by extensive use of clock gating. Design robustness and power consumption reduction has been obtained through use of level-sensitive storage elements (latches), which helps reducing constraints on clock distribution trees.

A set of software development tools has been specifically developed for the Macgic DSP to fully support its parameterization capabilities. These tools include an integrated development environment, a powerful macro-assembler as well as a source-level software debugger. A C++ phase-accurate pipelined model of the DSP is available and can be integrated in various cosimulations environments (e.g., Matlab/Simulink, System C, COSSAP/CoCentrics, and ModelSim). An FPGA implementation of the DSP also permits a quick prototyping of systems using the Macgic DSP.

A new DPU specialized for baseband data processing, as well as a simplified DMU and DPU for less computation-intensive applications, are in development.

References

[1] Õ. Paker, J. Sparso, N. Haandbaek, M. Isager, and L.S. Nielsen, A heterogeneous multi-core platform for low-power signal processing in systems-on-chip, *ESSCIRC 2002,* pp. 73–76, Firenze, Italy.

[2] C. Arm, J.-M. Masgonty, and C. Piguet, Double-latch clocking scheme for low-power IP cores, *PATMOS 2000,* Goettingen, Germany, pp. 217–224, September 13–15, 2000.

[3] J.M. Rabaey Reconfigurable computing: the solution to low-power programmable DSP, *ICASSP '97,* Munich, Germany, pp. 275–278, April 21–24, 1997.

[4] A. Abnous and J. Rabaey, Ultra-low-power domain-specific multimedia processors, in *VLSI Signal Processing IX,* IEEE Signal Processing Society 1996, Piscataway, NJ, pp. 461–470. http://www.ieee.org/organizations/society/sp.

[5] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, and J. Rabaey, Evaluation of a low-power reconfigurable DSP architecture, in *IPPS/SPDP '98 Workshops, Vol. 1388 of Lecture Notes in Computer Science,* pp. 55–60, Springer-Verlag, Heidelberg, 1998.

[6] P. Kievits, E. Lambers, C. Moerman, and R. Woudsma, REAL DSP technology for telecom baseband processing, *ICSPAT '98,* 1998.

[7] P. Mosch, G.V. Oerle, S. Menzl, N. Rougnon-Glasson, K.V. Nieuwenhove, and M. Wezelenburg, A 720-μW, 50-MOPs, 1V DSP for a hearing aid chip set, *ISSCC 2000,* pp. 238–239, February 2000.

[8] M.J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design,* Jones & Bartlett, Boston, 1995.

[9] *TMS320C1x User's Guide (Rev. C),* Texas Instruments, SPRU013C, 1991. http://focus.ti.com/lit/ug/spru013c/spru013c.pdf

[10] *TMS320C3x User's Guide (Rev. E),* Texas Instruments, SPRU031E, 1997. http://focus.ti.com/lit/ug/spru031e/spru031e.pdf

[11] *TMS320C5x User's Guide (Rev. D),* Texas Instruments, SPRU056D, 1998. http://focus.ti.com/lit/ug/spru056d/spru056d.pdf

[12] *TMS320C64x CPU and Instruction Set Reference Guide,* Texas Instruments, SPRU189F, 2000. http://focus.ti.com/lit/ug/spru189f/spru189f.pdf

[13] *ADSP-2100 Family User's Manual, Third Edition,* Analog Devices, 1995. http://www.analog.com/Processors/Processors/ADSP/technicalLibrary/manuals/16BitIndex.html

[14] *Blackfin Processor Instruction Set Reference,* Analog Devices, P/N 82-001991-01, 2003. http://www.analog.com/Processors/Processors/blackfin/technicalLibrary/manuals/blackfinIndex.html#Processor%20Manuals

[15] *ADSP-BF535 Blackfin Processor Hardware Reference,* Analog Devices, P/N 82-000410-13, 2003. http://www.analog.com/Processors/Processors/blackfin/technicalLibrary/manuals/blackfinIndex.html#Processor%20Manuals

[16] *ADSP-TS101 TigerSHARC Processor Programming Reference,* Analog Devices, Part No. 82-001991-01, 2003. http://www.analog.com/Processors/Processors/tigersharc/technicalLibrary/manuals/tigersharcIndex.html#Processor%20Manuals

[17] *DSP56000/DSP56001 Digital Signal Processor. User's Manual,* Motorola, DSP56000UM/AD Rev. 1. http://e-www.motorola.com/files/dsp/doc/inactive/DSP56000UM.pdf

[18] *Starcore SC140 DSP Core Reference Manual,* Motorola/Agere, MNSC140CORE/D Rev. 3, 2001. http://e-www.motorola.com/files/dsp/doc/ref_manual/MNSC140CORE.pdf

[19] *DSP1611 DIGITAL SIGNAL PROCESSOR Information Manual,* Agere, MN02-016AUTO, 2001. http://www.agere.com/client/docs/MN02016.pdf

[20] I. Verbauwhede and M. Touriguian. A low-power DSP engine for wireless communications, *J. VLSI Process.,* 18, pp. 177–186, 1998.

5

Low-Power Asynchronous Processors

Kamel Slimani
Joao Fragoso
Mohammed Es Sahliene
Laurent Fesquet
Marc Renaudin
TIMA Laboratory

5.1 Introduction

This chapter gives an overview of the techniques based on the asynchronous technology to design low-power circuits, particularly low-power processors. The asynchronous microprocessors designed and fabricated during the last two decades are reviewed. As it is well known, reducing the power consumption of integrated systems requires applying many different dedicated techniques, at different levels. This chapter is focused on the micro-architecture or structural level of asynchronous circuits. It also discusses how asynchronous processors can run dedicated power-aware operating systems, leading to a significant reduction of the power consumed by embedded systems — reduction that cannot be achieved using synchronous processors.

The first section covers fundamental power-reduction techniques that can be applied at the structural/micro-architecture level of a complex asynchronous circuit. The presentation is considering the design of low-power datapaths, pipelines, and control structures. In the second section, adequate design methodologies are presented, which take advantage of asynchronous circuits' properties for low power and target the power-reduction techniques presented in the previous section. The third section reviews fabricated asynchronous processors and highlights their performances with respect to low power. In this research domain, impressive prototypes were designed and fabricated, from 8-bit complex instruction set computer (CISC) microcontrollers to 32-bit reduced instruction set computer (RISC) machines. Finally, the last section demonstrates that asynchronous processors surpass synchronous processors when

designing low-power embedded systems using an operating system based on dynamic voltage scheduling algorithms. This original work illustrates how the software can exploit the key properties of asynchronous hardware to achieve a significant reduction of the power consumption.

5.2 Power Reduction Techniques in Asynchronous Circuits

5.2.1 Datapaths

Asynchronous datapaths have potentially low-power operation due to two main reasons [1]:

1. No global clock exists. In fact, clock is a major source of power consumption.
2. Asynchronous systems do nothing when their inputs have no data, so inactive asynchronous circuits shut themselves off.

The asynchronous circuit style (i.e., protocol and data coding) has a strong impact on the power consumption, however. A two-cycle protocol (i.e., nonreturn-to-zero [NRZ]) allows to transmit and to process data in each handshaking cycle. In another way, a four-cycle protocol (or return-to-zero [RZ]), which implies a more simple circuit, inserts a bubble (invalid data) into the protocol to finish the handshaking. This RZ phase consumes energy and no data is processed or transmitted.

The asynchronous bundled-data datapaths are similar to synchronous datapaths consequently all synchronous techniques to reduce power-consumption can be used in the asynchronous design.

On the other hand, delay-insensitive (DI) data encoding circuits do not require a request line. Requests are generated using a transition in all digits to be sent. Therefore, delay-insensitive circuits must be hazard-free because transitions are interpreted as requests. In this way, delay-insensitive circuits do not waste energy in spurious transition. Additionally, the number of transitions on the circuit input lines is no longer data-dependent. This feature simplifies the energy estimation [2] because the switching activity can be evaluated with no care about data being processed, provided that implemented algorithms are not data-dependent.

Thus, reducing DI datapath energy consumption mainly means reducing the number of gate switching to process data. The widely used dual-rail data encoding codes a binary value on two wires. An efficient way to reduce energy consumption is to increase the radix. 1-of-M data encodings hold more information in a single transition and more information can be processed/transmitted with less gate transitions [3]. For example, an n-bit adder power consumption can be reduced by a factor two replacing the dual-rail encoding by the 1-of-4 data encoding [4].

In addition, busses energy is saved when using large radix together with one-hot encoding. It also protects the busses from cross-talk effects. Indeed, when a transition occurs on a bus wire, the neighbor wires are by definition quiet [5,6]. Finally, asynchronous circuits are delay insensitive, thus repeaters can be efficiently replaced by asynchronous buffers. Obviously, n-of-m data encoding could be used and the designer has to exploit the solution space to choose the appropriate power/area/speed trade-off [7].

At the architectural level, there are other techniques to save energy-complexity in datapaths structures as presented in Manohar [8] and Park et al. [9]. Although synchronous datapaths power consumption is generally data-dependent, DI asynchronous circuits may lose this ability. For example, in a simple synchronous 32-bit array multiplier, the energy consumption depends on the operand wideness, but a given asynchronous multiplier consumes almost the same power for all operations.

Table 5.1 lists the results of benchmark programs for the 32-bit multiplier of microprocessors [10,11]. It is easy to see that an asynchronous implementation would waste a lot of energy using a constant 32-bit data width. In Manohar [8], a new class of number representation is presented. The number representation uses a self-delimited data encoding that allows dynamically adapting the operation width while keeping delay insensitivity.

Additionally, in specific applications such as finite impulse response (FIR) Filters, the knowledge about the data dependency can be used to improve the architecture power efficiency [12]. Finally, when

TABLE 5.1 Benchmark Programs Data Pattern

Architecture	Benchmark Program	Number of Multiplication	Average Bit Length	
			Multiplicand	Multiplier
SPARC v8.0	lisp	2 K	5.32	8.00
	gcc	240 K	5.81	3.67
	ijpeg	82 M	7.69	11.24
Simplescalar	lisp	15 K	1.54	0.43
	gcc	1 M	6.69	9.35
	perl	42 M	0.01	4.01
	go	13 M	3.70	4.41
	ksim	5 M	11.43	12.88

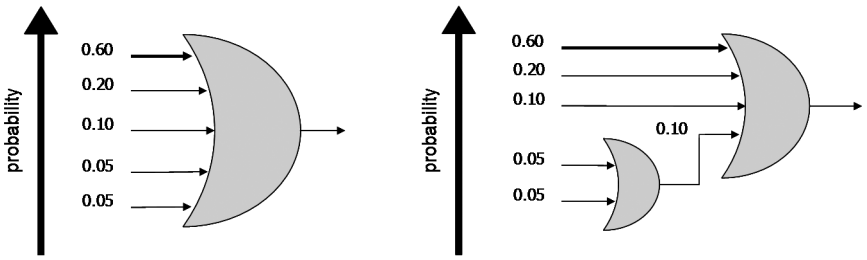


FIGURE 5.1 An OR-5 gate decomposition example.

decomposing gates to limit maximal fan-in, the input transition probability knowledge could be taken into account to reduce the average number of transitions. Figure 5.1 is an example of OR-5 decomposition with maximal fan-in equal to four and all input mutually exclusive. The three most probable inputs (i.e., 60%, 20%, 10%) are connected to the output gate, resulting in a unique gate switching in 90% of the cases.

The use of all these techniques improves energy-aware asynchronous systems potential.

5.2.2 Pipelines

Several techniques have been proposed and introduced to improve the performance of circuits such as processors. Pipelining is a common and popular way to increase the throughput of a circuit.

In synchronous designs, pipelining is achieved by inserting registers within combinational stages and by adjusting the global clock accordingly. Whereas in asynchronous circuits, pipelining is performed only by inserting latches/buffers between stages, no global synthesis has to be done. It looks more complicated in synchronous than in asynchronous because all the circuit has to be resynthesized in synchronous to fulfill the new global synchronization for a functional correctness.

The ease of performing a deeper pipeline in asynchronous must be carried out with care. Indeed, inserting latches to improve the throughput of a circuit induces additional hardware affecting consequently the power consumption of a circuit. It is the eternal trade-off between speed and power consumption. Nevertheless, an upper bound of buffers can be added into a design to reach the maximum throughput. Beyond this number of buffers, adding more buffers increases the power consumption overhead without improving the performance of the circuit. More dramatically, adding an excessive amount of buffers leads to a decrease of performance because the delays of buffers affect the throughput [13].

Some models have been proposed to determine the optimum number of buffers to be added in an asynchronous pipeline to get the maximum throughput. An interesting experiment was proposed by [13]. It has demonstrated that performances of an asynchronous ring can vary between a blocked-state when the number of buffers is lower than the minimum to a maximum throughput at the optimum number of buffers. Beyond this optimum number of buffers, the performance decreases implying an increase of the energy dissipation.

Other models such as $E\tau^2$ [14] energy-time metric is used to determine the upper bound of buffers to be added in an asynchronous pipeline to get the highest throughput [15]. Fixing the optimal number of buffers in a pipeline is also called slack matching [16]. Slack matching can be applied without affecting the correctness of the circuit wherever no arbitration structure exists. These parts of the circuit, which do not include arbitration, are called slack elastic.

Another source of energy waste is observed when branch instructions are executed and taken. As a rule, the deeper the pipeline the higher the energy waste when a branch is taken. This is due to the presence of as many instructions in the pipeline as pipeline stages, so the energy dissipation is directly proportional to the number of instructions that must be discarded in the pipeline. To reduce this waste of energy, solutions based on branch prediction can be used. When the prediction is close to the reality, the energy overhead is significantly reduced. The branch prediction hardware consumes energy for every instruction, however, so it is questionable whether the total energy consumption of the whole execution is reduced. An elegant solution to this problem, which exploits the elasticity of asynchronous pipelines, is to control the pipeline depth by collapsing pipeline stages together. This is performed dynamically, so the processor can switch from a fully pipelined configuration to a low depth pipeline configuration very quickly during an instruction execution [17]. Dynamic configuration of pipeline stages has the potential to significantly save energy. Decreasing the pipeline depth slows down the processor but also saves energy because fewer "speculative" instructions are fetched and decoded. It is based on selectively making some latches transparent, which join pipeline stages together. In micropipeline asynchronous circuits, dynamic configuration of pipeline stages is achieved by the mean of reconfigurable latch controllers [18] that can be either "permanently transparent" (collapsed) or "normal." These new latch controllers were used in AMULET3 [19].

5.2.3 Control Structures

When the datapath is nonlinear (i.e., in the presence of control structures such as multiplexers), the complexity of the circuit is higher because several inputs can be propagated to one output. These architectures are expensive in terms of power consumption if no care is taken in their implementations. Strategies are proposed to reduce the power consumption of these components by unbalancing the structure of choice in favor of the cases with the highest probabilities [20]. The probabilities are established by simulation and are application-dependent. This optimization is a step further toward the power consumption reduction of a circuit. Indeed, the power dissipation overhead lies on the enormous forks that connect several inputs to one output. The idea for reducing the power consumption of these structures is to decompose these huge forks into a dissymmetric tree of small forks. This decomposition allows the signals with the highest probabilities to cross a reduced amount of smaller gates, thus reducing the loads switched. This decomposition is similar to the one proposed in Figure 5.1.

The same reasoning can be applied to any control structure, such as demultiplexers, for which one input is sent to one among several possible outputs.

5.3 Design Methodologies for Low Power

Several universities develop their own design flows and use them in order to demonstrate the efficiency of their design methodologies. Research in asynchronous logic has lead to interesting circuits that demonstrate the effectiveness of asynchronous logic. The results obtained stimulate the research of asynchronous logic. Among some design flows developed in some universities, we can point out Tangram [21], which was developed at Eindhoven University in collaboration with Philips Research Lab in Eindhoven, the Netherlands. This design flow was used to implement a low-power asynchronous 80C51 [22]. Tangram was a good inspiration source for the AMULET group at the University of Manchester, U.K., to develop Balsa [23]. Balsa is a tool that can generate bundled data circuits. A DMA controller [24] integrated in AMULET3i was a good example to demonstrate the potential of the Balsa synthesis tool.

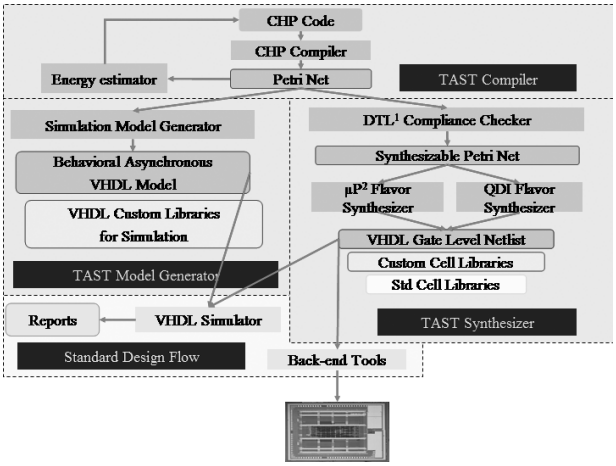


FIGURE 5.2 Design flow in TAST.

Recent works have been accomplished in Balsa to generate quasi-delay insensitive (QDI) architecture for secure applications. An asynchronous implementation of the commercial advanced RISC machine (ARM) microprocessor named SPA [25] was achieved to illustrate the efficiency of this synthesis tool. Finally, we introduce TIMA asynchronous synthesis tools (TAST) [26] (Figure 5.2). It was developed at TIMA laboratory in Grenoble, France, within the concurrent integrated systems (CIS) group. TAST is a complex and complete tool for designing asynchronous circuits. It takes a high-level language, which is based on VHDL and CHP developed by Alain Martin at the California Institute of Technology (Caltech, Pasadena, CA). TAST enables the designer to perform functional simulation in VHDL or C, and to synthesize the communicating hardware processes (CHP) specification into an asynchronous micropipeline or QDI circuit.

One of the interesting features of the TAST design flow is the trace estimator. It gets information on both the circuit activity and the energy consumption during a specific simulation [27,27a]. The trace estimator, which is an interesting option that can guide the designer to achieve a low-power asynchronous circuit, performs it. By following a rigorous strategy of design (it is possible to add labels in the CHP specification to monitor one or several particular instructions), the designer can get information on power consumption and can nicely exploit them to design low-power circuits.

The activity estimation tool allows the designer to achieve a profiling of the code to get code coverage information of a CHP description. The aim of this code profiling is to identify parts of the description that are never used or seldom used.

The activity estimation of a circuit allows the designer to identify parts of a circuit that are the most frequently used. For an asynchronous microprocessor for example, it is interesting to know which units are often used: the register bank, the ALU, the load/store unit, and so on. It is possible to get information at a fine grain with a higher accuracy. For instance, it is interesting to know which registers in a register bank are the most frequently accessed and, therefore, optimize the hardware accordingly. For an ALU, this information on activity can guide the designer to choose the best operator architecture (the operator that is solicited most often will have a low-power architecture).

Information on channels is useful to choose the best encoding. The activity of digits in a channel is useful to see which digits of a channel are rarely, if ever, used. According to this analysis, the channels can be sliced into a set of subchannels that are conditionally active (channel width adaptation).

Finally, statistics on choice structures are essential to privilege cases with the highest probabilities. The statistics obtained can guide the designer to unbalance the structure of choice in favor of the cases with the highest probabilities [20]. For example, if the guard G0 has the highest probability, the structure of choice could be unbalanced as follows:

```
[ G0 => I0                [ G0 => I0
@ G1 => I1 =>             not G0 =>[ G1 => I1
@G2 => I2                            @ G2 => I2
]                                          ]
                                 ]
```

The mutually exclusive structure of choices (written in pseudo CHP code) on the left hand side is a regular structure of three guards G0, G1, and G2 with their respective instructions I0, I1, and I2. On the right-hand side, it is an unbalanced structure to privilege the case G0. The effect on the final circuit by decomposing the structure in that way is to reduce the power consumption of the execution of the guard G0. Indeed, every time the guard G0 is executed, it costs the energy of a two-input multiplexer instead of a structure with three inputs. A complex circuit including complex choice structures having unequal probabilities results in large energy savings.

The activity estimation tool is an efficient tool to guide the designer for power reduction. Moreover, TAST goes beyond by offering to the designer the possibility to get an estimation of the energy in terms of the number of transitions. Commercial tools exist, but the estimation of the energy is done at the gate level [28] or at the transistor level [29], thereby occurring too late in the design flow and taking too much time to be simulated. This is the reason why the trace estimator tool in TAST allows the designer to get power estimation at the synthesizable CHP specification level. This estimation is done without performing the synthesis of the circuit, but it is based on how TAST does the synthesis of asynchronous circuits. Furthermore, this is a C-simulation, thus the simulation and the estimation are very fast, contrary to a gate simulation, which is generally a VHDL gate simulation involving gate delays (using VITAL libraries, for example).

This power consumption estimator is a good way to check whether an optimization is judicious, and indicates the relative gain between two different implementations of the same circuit.

5.4 Asynchronous Processors: A Review

Important studies have been conducted on asynchronous logic to prove the efficiency in terms of power reduction. Thanks to several design methodologies that have emerged from these studies, some designs around the world have been achieved and are very encouraging for the future. Among some circuits, a short list of asynchronous microprocessors, which have marked the last 15 years of intensive research, is presented.

5.4.1 CAP

Alain Martin's group at Caltech designed the first microprocessor entirely implemented in asynchronous logic at the end of the 1980s. This microprocessor is known as the Caltech asynchronous processor (CAP) [30]. This simple processor was designed to prove the feasibility of asynchronous circuits by using the methods of program transformations developed at Caltech [31].

CAP is a 3-stage pipeline, 16-bit RISC processor with 16 registers of 16 bits; it is implemented in dual-rail QDI architecture using a 4-phase communication protocol. The circuit is synthesized and mapped on static complex gates. It contains 20,000 transistors. The circuit is physically manufactured in MOSIS SCMOS 1.6-μm technology.

Performances of the processor at different supply voltages have been recorded: 26 MIPS for a power consumption of 1.5 W at 10 V; 18 MIPS for a power consumption of 225 mW at 5 V; and 5 MIPS for a power consumption of 10.4 mW at 2 V.

5.4.2 MiniMIPS

More recently, the work done at the Caltech on asynchronous design methodologies has lead to the design and fabrication of a powerful asynchronous MIPS R3000 prototype: MiniMIPS [16]. One of the goals of this project was to apply the theory based on the $E\tau^2$ metric [14,32], devoted to the estimation of time and energy efficiency of computation.

The synchronous MIPS architecture is based on three pipeline stages: PC address computation, instruction fetch, and instruction execution; however, the asynchronous MiniMIPS is very finely pipelined inside these stages in order to increase the performances without sacrificing the power consumption. A slack matching was achieved in many parts of the circuit to obtain the maximum throughput (this slack matching was done without affecting the correctness of the circuit because it was achieved on slack elastic parts).

In addition, improvement in completion mechanism was performed on the datapath. Indeed the datapath was decomposed into small units (4 bits wide) that generate their own completion signals. As a result, the completion tree is smaller, thereby reducing the delay and the power consumption of the circuit.

MiniMIPS is very innovative for the year of its invention (1997) because it includes a bypass mechanism and exceptions management. The processor was manufactured in 0.6-μm SCMOS technology, and the performances are 150 MIPS for a power consumption of 1 W at 2 V and 280 MIPS for a power consumption of 7 W at 3.3 V.

5.4.3 AMULET1, 2, 3

AMULET (asynchronous microprocessor using low energy and technology) is a project born in late 1990 at the University of Manchester. The purpose of this project was to demonstrate the potential of the asynchronous technology to reduce the energy consumption.

The advanced RISC machine microprocessor (ARM) was chosen to prove the feasibility of designing a fully asynchronous commercial processor.

The first version was AMULET1 [33] released in early 1993. The realization of this processor required 5 men-year. The processor has been implemented using Sutherland micropipeline [34], and using a 2-phase communication protocol (nonreturn-to-zero). The result obtained for this circuit was not very successful — the performances measured were lower than those of the original ARM6. These unexpected results are partly due to an unsuitable deep pipeline. The power consumption of AMULET1 is 83 mW for a performance of 9 K dhrystones. For the original ARM6, the power consumption is 75 mW for a performance of 14 K dhrystones at 10 MHz.

The experience acquired with the realization of AMULET1 lead to a second version, AMULET2 [35], in 1996, which was higher performance than the former. The architecture is still based on Sutherland micropipeline. The 2-phase communication protocol was replaced by the 4-phase communication protocol, however, which has the advantage of being faster and less consuming despite the multiplication by two of the number of transitions. The 2-phase communication protocol consumes more than the 4-phase because of the use of a complex double edge sensitive logic. Improvement was also achieved by reducing the unsuited deep pipeline of AMULET1, which was partly responsible for the extra power dissipation.

AMULET2 integrates a "branch target cache" for branch prediction, a mechanism of register blocking in the form of a FIFO and a mechanism of forwarding. Logic blocks, such as the shifter and the multiplier, are bypassed when they are not necessary to reduce energy waste. Furthermore, AMULET2 includes a "halt" mechanism to stop all activity in the circuit. The circuit was implemented in a CMOS 0.5-μm technology. The performances of AMULET2 are 40 MIPS for a power consumption of 150 mW, which is better than the ARM710 but less than the ARM810. AMULET2 was incorporated in AMULET2e with flexible RAM memory.

A third version, AMULET3 [19], was designed in 2000. It is an asynchronous 32-bit RISC processor that is competitive with the synchronous ARM9TDMI core.

The datapath is a full custom design, and the standard library was provided by ARM Limited. The control part was synthesized using Petrify [36]; Petrify takes a circuit in form of a STG and transforms it into a speed independent (SI) circuit. AMULET3 uses latch controllers [18] to reduce the pipeline when branch instructions are executed. The technology used in AMULET3 is 0.35 μm and the power supply voltage is 3.3 V. The performances recorded for AMULET3 are 120 MIPS with a power con-

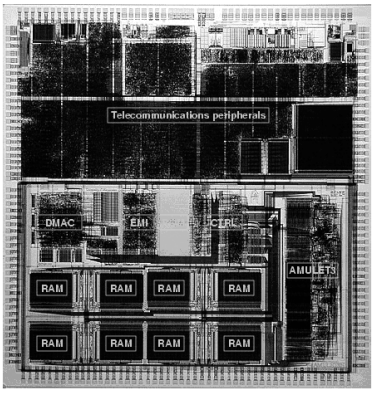


FIGURE 5.3 DRACO layout.

sumption of approximately 155 mW. These performances are very similar to the synchronous ARM9TDMI core.

AMULET3 demonstrated that asynchronous technology is competitive in terms of power consumption. It was integrated in AMULET3i [37] with an asynchronous bus named MARBLE [38], a DMA controller (synthesized using Balsa [23]) and RAM and ROM memories. AMULET3i is used for commercial applications; the first application is for a DRACO control system (Dect [digital enhanced cordless telecommunications] RAdio COmmunication controller) depicted in Figure 5.3.

5.4.4 Asynchronous 80C51

The CISC asynchronous 80C51 microcontroller [22] was designed at Eindhoven University of Technology in collaboration with Philips Research Laboratories in 1995. The circuit was implemented in a relatively short time of 6 months.

The asynchronous version is completely compatible with the original processor. The goal of this duplication was to demonstrate the benefits of an asynchronous implementation by comparing directly the original synchronous processor with the asynchronous implementation.

The circuit was specified in high-level language using Tangram [21], and was compiled in a netlist using the Tangram compiler. The compilation of a Tangram program is done in two stages in which the handshake circuits represent the intermediate form. The handshake circuits are implemented using the 4-phase protocol and single-rail architecture.

The transparency of the Tangram compiler allowed the designers to bring optimizations at the Tangram program specification. In addition, a suitable combination of bus-structure and point-to-point communication contributed to save extra energy dissipation.

The processor was mapped onto a standard cell library and fabricated in CMOS 0.5-μm technology. It integrates a DC-DC converter for voltage scaling (see Section 5.5).

The performances obtained at a supply voltage of 3.3 V are 4 MIPS for a power consumption of 9 mW. It is noted that the energy consumption was reduced by a factor of 4 compared with the original synchronous version. These results prove the benefits brought by the asynchronous logic in terms of power consumption and low electromagnetic emission. This processor has been used in a pager; the layout of the circuit is depicted Figure 5.4.

5.4.5 Lutonium

Lutonium [38a] is a recent asynchronous 8051 microcontroller developed by the California Institute of Technology. Although the 8051 is an irregular CISC processor, which is not really well suited for low-power, it is very popular and is often found in applications where minimizing the energy is important.

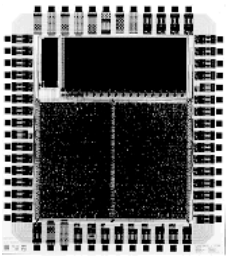


FIGURE 5.4 Asynchronous Philips 80C51.

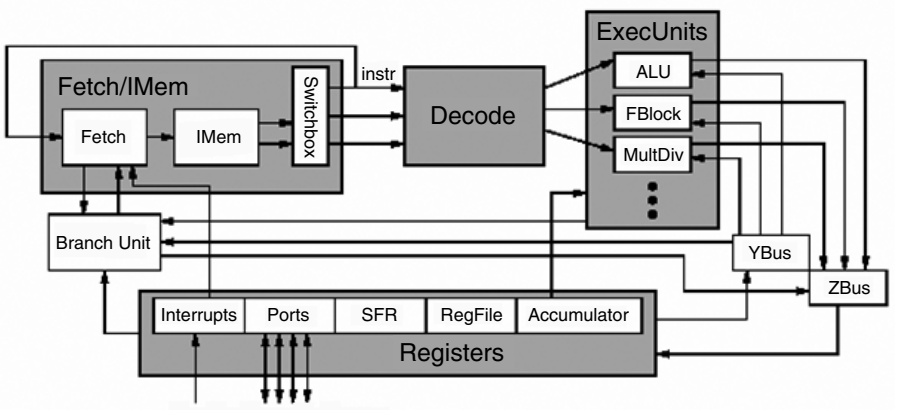


FIGURE 5.5 Lutonium architecture.

TABLE 5.2 Lutonium Performances

1.8 V	200 MIPS	100.0 mW	500 pJ/inst	1800 MIPS/W
1.1 V	100 MIPS	20.7 mW	207 pJ/inst	4830 MIPS/W
0.9 V	66 MIPS	9.2 mW	139 pJ/inst	7200 MIPS/W
0.8 V	48 MIPS	4.4 mW	92 pJ/inst	10900 MIPS/W
0.5 V	4 MIPS	170.0 mW	43 pJ/inst	23000 MIPS/W

Lutonium was implemented for an optimal Et^2 [14] parameter. It is highly pipelined to increase the speed. The instruction decoder is more complex than the MiniMIPS [16] and is centralized in one component to decrease the size and the number of communication channels in the circuit. Moreover, the lutonium performs a deep-sleep mode in which almost all switching activities are stopped. The advantage of asynchronous implementation is that Lutonium can wake up instantly from the deep-sleep mode. Moreover, Lutonium introduces a segmented bus-control protocol to save energy: useful digits only are sent (especially for control structures as presented in Section 5.2.2).

The QDI logic is used for its robustness property at low voltages. Lutonium has been inplemented in SCN018 (a 0.18-μm CMOS library) by Taiwan Semiconductor Manufacturing Company (TSMC) supplied by MOSIS; the nominal supply voltage is 1.8 V. Figure 5.5 depicts the architecture of Lutonium.

The performances of Lutonium are shown in Table 5.2.

5.4.6 MICA

Microcontrôleur asynchrone (MICA) is a quasi-delay insensitive asynchronous 8-bit microcontroller CISC machine, designed by the CIS group of TIMA in collaboration with France Telecom research and development in Grenoble, France, and STMicroelectronics in Grenoble in 2000 [39,40]. It integrates two

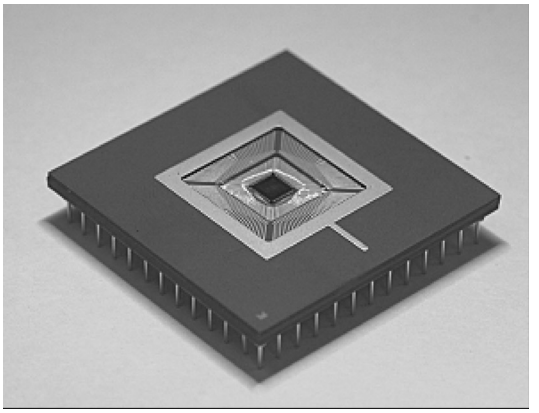


FIGURE 5.6 MICA microcontroller and measured performances.

different register-files: eight 8-bit registers devoted to data, and eight 16-bit registers devoted to pointers (including the program counter and the stack pointer). Specific arithmetic units are associated with each register file enabling concurrent computations of data and addresses. A peripheral unit is also included, supporting six 8-bit parallel ports, and four serial links (using a two-phase delay insensitive protocol compatible with the RISC asynchronous ASPRO processor). Moreover, the microcontroller integrates 16 Kbytes of RAM and 2 Kbytes of ROM. The latter includes a built-in-self-test, which is executed at reset according to the boot mode selected (eight modes are available).

The MICA processor was tested functional from 3 V down to 0.6 V (2.5 V is the nominal voltage of the 0.25 μm CMOS technology used). It is noticeable that the chip only consumes 800 μW at 1 V, still delivering a computational power of 4.3 MIPS. At 0.8 V, the chip consumes less than 400 μW (Figure 5.6).

The microcontroller core was designed using the so-called quasi-delay insensitive (QDI) logic. A four-phase protocol was used in conjunction with an n-rail encoding. This chip was a vector for developing new skills in the design of standard-cell based QDI asynchronous circuits. The design of MICA was focused on two correlated concerns: designing distributed asynchronous finite state machine and designing for low power.

To reduce the power consumption of the microcontroller, the number and the energy-cost of communication actions occurring during the execution of each instruction, as well as the number of sequential steps to perform each instruction were minimized. In other words, instead of designing the architecture around a big central sequencer, the sequencer implementation was distributed all over the architecture and as much as possible. The asynchronous logic is particularly well suited to satisfy such a design approach because, by nature, the sequencing of an asynchronous circuit is performed by multiple local sequencers implementing handshaking communications and local treatments.

Thus, the architecture of MICA was designed as a distributed system, each part providing specific services. For example, the two register files, the status register, and the memory integrate local units, which manage the memory resources. These modules implement functions such as "read," "write," "read then write back" or even more complex functions such as: read a byte, increment/decrement the pointer/address, and read another byte (Copy and Push & Load instructions, for example, use these features). Adopting such an approach significantly simplifies the design of the main sequencer of a CISC microprocessor like MICA. It then minimizes the power consumed by the main sequencer, where the consumption associated with each instruction is the direct image of its complexity. In fact, complex instruction implementation does not penalize simple instruction implementation at the main sequencer level. Moreover, such a distributed approach minimizes the power consumed by communications because the minimum number of transactions occurs through busses (memory accesses for example).

Because of the low-power constraint and because computational power was not a priority for the targeted applications, a minimum number of pipeline stage was introduced. This does not avoid parallel

execution of instruction subparts, but simply means that parallel execution of instructions is not supported. In some cases, however, successive instructions may partially overlap.

Finally, at the signal level, communication channels are using a low-power data encoding. Instead of using dual-rail coding, N-rail coding (also called "one hot"), is used (i.e., one out of the N wires is active during a transaction). Therefore, the different parts of the architecture are controlled by the means of channels using 5-rail to 12-rail data encoding, which minimizes the number of transitions per communication action, and thus minimizes the dynamic power consumption. The datapaths (8-bit and 16-bit) are entirely designed with 4-rail encoded data, requiring radix-4 logic/arithmetic processing units. The register files are also designed with 4-rail encoded data. Instead of bit-registers, they are built of digit-registers, each digit representing four values.

MICA was successfully used for the design of a contact-less smart-card chip, called MICABI [39], which integrates an on-chip coil connected to a power reception system and an emitter/receiver module compatible with the ISO 14443 standard. The benefit brought by an asynchronous processor in such a system on chip (SoC) is that design constraints are relaxed [39,41,42], especially concerning the software, the analog, and the radio-frequency parts.

5.4.7 ASPRO

ASynchronous PROcessor (ASPRO) [43] is a 3-stage pipeline 16-bit RISC asynchronous microprocessor. Its design started at the Ecole Nationale Supérieure de Télécommunications de Bretagne antenne de Grenoble, in Grenoble, France, in collaboration with France Telecom R&D and STMicroelectronics in 1998. The team then joined TIMA [40] in 1999. ASPRO is a regular RISC microprocessor, which decodes the instructions in order and completes them out of order; it contains 16 registers of 16 bits.

The design flow and the circuit style are an original application of the method of Alain Martin. The motivations were to get experience in the design of complex asynchronous QDI circuits using standard cells and to demonstrate that asynchronous design techniques could improve very large-scale integration (VLSI) systems in term of speed and power consumption.

ASPRO was implemented using multi-rail encoding and a 4-phase communication protocol. The synthesis was partially performed by hand and it targeted the standard cell library provided by the founder STMicroelectronics.

The processor integrates two distinct on-chip memories for instructions and data. It was manufactured in 0.25-µm CMOS technology by STM. It can deliver 24 MIPS for a power consumption of 20 mW at 1 V and 140 MIPS for a power consumption of 350 mW at 2.5 V (including consumption of the memories). The CPU integrates three independent sources of supply voltages (from 0.9 to 3.0 V) for dynamic voltage scaling purposes (see Section 5.5). The final physical circuit is pictured in Figure 5.7.

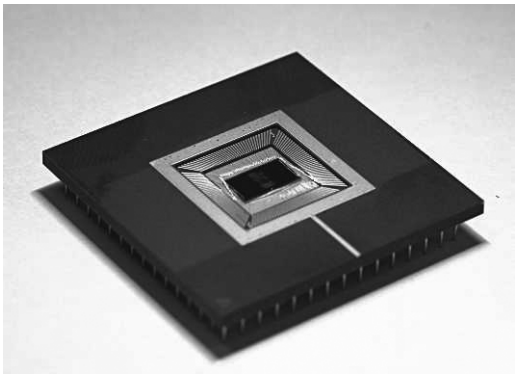


FIGURE 5.7 ASPRO.

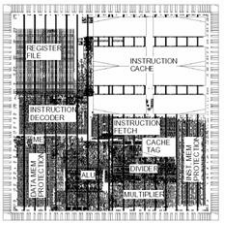


FIGURE 5.8 TITAC-2 layout.

5.4.8 TITAC-2

The realization of an asynchronous processor TITAC-2 [44] was invented by the Tokyo Institute of Technology in Tokyo, Japan. It is an asynchronous version of the MIPS R2000 processor (some instructions are missing or are modified). It is a 5-stage pipeline 32-bit RISC processor.

TITAC-2 is implemented using a scalable-delay insensitive (SDI) architecture [44]. SDI is based on QDI architecture with an unbounded delay model (i.e., no upper bound is assumed on the gate and wire delays). The benefits of this model are visible in terms of speed. The SDI circuits can effectively run faster than the equivalent QDI circuits.

To get the maximum performance without affecting the consumption of the processor, pipelining in TITAC-2 was fixed by simulation.

The data are coded into dual-rail and the communication is done using a 4-phase communication protocol. TITAC-2 was manufactured using a 0.5-μm CMOS technology and synthesized using a standard cell library.

The performances obtained are 52.3 VAX MIPS for a V2.1 test dhrystone and consumes 2.11 W at 3.3 V and 20°C. Figure 5.8 illustrates the layout of the circuit.

5.4.9 Conclusion

The results obtained for these microprocessors are in reward of all the work accomplished in asynchronous logic over the last 15 years of research. The asynchronous 80C51 and the AMULET3i processors are used in commercial applications. These two asynchronous processors, which are a reliable duplication of synchronous commercial processors, have demonstrated the advantages of asynchronous logic in comparison with the synchronous logic in terms of power reduction. The MiniMIPS prototype demonstrated that high performance is achievable using the asynchronous technology, while keeping the relative power consumed rather low. Finally, the MICA microcontroller and the MICABI contact-less system-on-chip demonstrated the benefits brought by the asynchronous technology to design mixed-signal circuits requiring low voltage, low power, and low noise.

All these prototypes have demonstrated that asynchronous logic has attained a maturity level that is commercially viable today. It is, however, clear that an ever greater reduction of asynchronous processors power consumption is possible, by combining the different techniques introduced in Section 5.2 and by applying design methodologies devoted to low power as suggested in Section 5.3.

5.5 Power Reduction Techniques at the System Level

5.5.1 Introduction

To go a step further in reducing the power consumption let us now jointly consider the hardware and software parts of an integrated system. Several hardware and software techniques have been developed over the last years to manage the electrical consumption of a system. Nevertheless, as devices become much more powerful and sophisticated, power requirements increase continuously [45]. Therefore, new power management techniques have been investigated at hardware and software levels [46,47]. In this section, a new method is considered that combines asynchronous processors and an operating system

for low-power management. Although the literature on power management expounds extensive research on what to do at the hardware or software level, this approach investigates both levels simultaneously. In cooperation with a power management policy adapted to an asynchronous processor, the operating system (OS) adjusts the speed of the processor to the task requirements at runtime by controlling the processor operating voltage. This scheme exploits the ability of asynchronous processors to self-regulate their processing speed with respect to the supply voltage only [48].

5.5.2 Principles of Power Reduction with Operating Systems

For current CMOS integrated circuits, power dissipation is dominated for the moment by the switching power, even if the static leakage current induces more and more power losses in deep submicron technologies. This arises from the charging and discharging of the loading capacitance and is expressed for a synchronous processor as:

$$P = CV^2 f \tag{5.1}$$

where C is the load capacitance, V is the supply voltage, and f is the clock frequency [49–51]. In the case of an asynchronous processor, the relation reads:

$$P = CV^2 s \tag{5.2}$$

where s is the instantaneous speed of the processor. These equations suggest that minimizing the load capacitance, reducing supply voltage (or speed), or slowing down the clock can reduce power consumption.

Although the load capacitance can be affected during chip design by minimizing on chip routing capacitance [52], voltage scalable processor and power controllable peripheral devices make it possible to reduce power at the operating system level. In cooperation with scheduling policies, operating systems can vary the processor's speed and voltage (dynamic voltage scaling) and put devices in low-power sleep states (dynamic power management) [53]. Indeed, a typical embedded system consists of processors, memories, communication links, and other peripheral devices that are not always used. Thus, an OS can control the power states of peripheral devices in the system according to the workload. When a device is idle, it can be put in a low-power sleeping state after a long enough idle period to compensate time and energy overhead of shutting down and waking up [54,55]. To determine whether a device can sleep, time-out, predictive, and stochastic policies have been developed [56,57]. In the time-out based policy, after a device is idle for a time-out value, it remains idle for at least a certain time. Predictive policy uses the past idle periods or both the current and the past idle period to predict the length of future idle periods eliminating the time-out period wasted energy. Choosing a policy for a given application depends on prediction accuracy, power savings, and resources requirements such as memory and computation [54,56,57]. A shutdown mechanism can be applied to put the processor into idle mode when not in use [54]; however, a more fruitful way to save power is to run slower at reduced voltage according to computational load demands [58].

A technique called dynamic voltage scaling (DVS) allows processors to dynamically alter their voltage and speed at runtime under the control of voltage scheduling algorithms [59,60]. These algorithms predict future workload and set the processor voltage and speed based on this prediction. The interval based voltage scheduler called PAST (bounded-delay limited PAST scheduling algorithm) [61] assumes that the processor utilization of the next interval will be like the previous one and updates the processor speed accordingly. If the last interval was busier than idle, speed is increased. Similarly, if it was mostly idle, speed is decreased. A comparative simulation of PAST and other proposed algorithms points out that a simple smoothing algorithm can be more effective than sophisticated prediction techniques [62]. Recent studies take the real-time constraints [63,64] into account. The processor speed is estimated considering workload prediction and task deadlines. The goal is to complete tasks before or on deadlines. Because the processor is often running at a reduced speed, however, all these studies assume missed deadlines as a trade-off between power saved and deadlines missed.

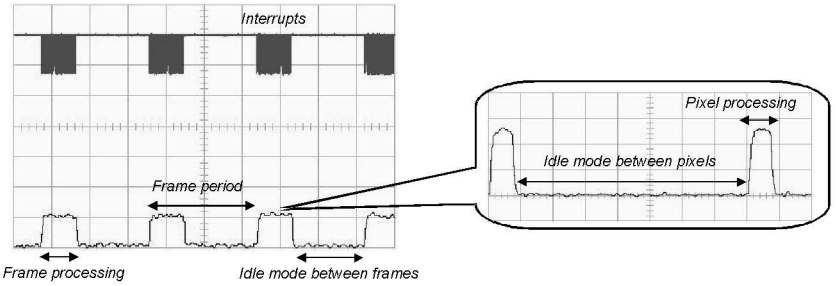


FIGURE 5.9 Fine grain idle mode with the ASPRO processor.

5.5.3 Low-Power Sleeping States

5.5.3.1 Synchronous Processors Idle Mode

Synchronous processors need a huge amount of time and energy to enter the idle mode or to wake up from a sleeping state. This constitutes a severe drawback to swap rapidly with low energy costs between the normal and the idle mode. Indeed, the transition time to a sleeping state currently takes several tenths of microseconds (for the lpARM [60] or the Crusoe [65,66]) to several tenths of milliseconds for the wake-up (with a StrongARM processor [64]). The time and energy overheads do not allow continuous starting and stopping of the processor. For minimizing these drawbacks, the synchronous processors own different levels of sleeping states.

5.5.3.2 Asynchronous Processors Idle Mode

Contrary to the synchronous processors, the asynchronous processors are well suited to exploit fine grain idle mode. Indeed, they only consume energy if the processor has data to process. If no data is processed, the processor is immediately set in idle mode until a request or interrupt wakes it up. The wake-up time overhead is equivalent to the time overhead of an interrupt. These specific properties of the asynchronous circuits allow the use of a fine-grain idle mode. This point is illustrated in Figure 5.9, which presents real measurements performed with a video application running on the ASPRO processor (see Section 5.4) [43,67]. A digital video camera sends images to ASPRO via a high throughput serial link. Between two consecutive frames, the processor is idle because it has no data to process. More astonishing, if the frame is zoomed, we can see that between two consecutive pixels the processor is idle too (see the balloon in Figure 5.9). The activity duty cycle is 0.333 for the frames and 0.125 for the pixels. This leads to a 95% energy reduction compared with a system without idle capabilities. This is not possible for a synchronous processor.

5.5.4 DVS for Synchronous Processors vs. DVS for Asynchronous Processors

5.5.4.1 Timing Model for Asynchronous Processor Speed Variation

The power supply voltage only drives the speed variation of asynchronous processors. The variation time t_v only depends on the DC-DC converter and the load capacitance of the processor. Time t_v could be modeled as a function of V_{dd1} and V_{dd2}, the supply voltages before and after the variation. A simple way to express t_v is to consider a linear expression:

$$t_v = k \cdot \left| V_{dd1} - V_{dd2} \right| \tag{5.3}$$

where k is a fitting parameter depending on the DC-DC converter and the load capacitance of the processor.

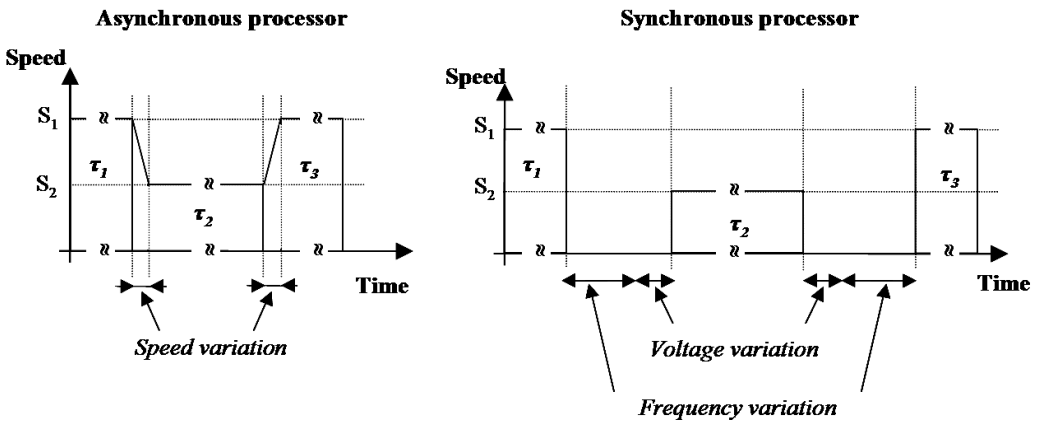


FIGURE 5.10 Task execution and speed variation for asynchronous and synchronous processors.

In the asynchronous case, we can notice that task processing is not stopped during the speed variation. Moreover, the speed can vary continuously and can be finely adjusted during the task execution. Figure 5.9 illustrates that the execution of tasks τ_2, and τ_3 is not stopped during the speed variation.

5.5.4.1.1 Timing Model for Synchronous Processor Speed Variation

In the synchronous case, the speed variation is controlled by the supply voltage and by the clock frequency. This operation needs two steps. In the case of speed reduction, the first step is used to slow down the frequency and the second step reduces the power supply voltage. In the case of speed increase, the first step increases the voltage and the second speeds up the frequency. The transition time, t_v, depends on the DC-DC converter and the phase-locked loop (PLL) that controls the clock frequency; t_v can be expressed as:

$$t_v = t_{DC\text{-}DC} + t_{PLL} \tag{5.4}$$

where $t_{DC\text{-}DC}$ is the transition time from the initial voltage to the final voltage, and t_{PLL} is the transition time to change the frequency. As modeled in Equation 5.3 t_v can be expressed as:

$$t_v = k_1 \left| V_{dd2} - V_{dd1} \right| + k_2 \left| f_2 - f_1 \right| \tag{5.5}$$

where k_1 and k_2 are, respectively, the fitting parameter for the DC-DC converter and for the PLL. Frequencies f_1 and f_2 are the initial and the final frequency of the processor.

The frequency transition time t_{PLL} is constant in the case of the StrongARM processor [64] or variable if a Transmeta Crusoe is considered [65,66]. The frequency of synchronous processors does not vary continuously but by steps. For instance, the Crusoe processor frequency varies by 33 MHz steps. Moreover, a critical point (except for the lpARM), which drastically increases the inefficiency of synchronous processors, is that task processing is stopped during the speed variation. Figure 5.10 presents the timing model for the synchronous processor speed variation. Because the processor stops the processing during the transition time, the speed is set to zero for this duration.

5.5.4.1.2 DVS Additional Energy Costs for Synchronous Processors

Contrary to the asynchronous processors, the speed variation of synchronous processors has a cost in terms of energy and time. Indeed, synchronous processors are stopped, and they continue to spend energy during the speed variation, while asynchronous processors process their tasks and vary their speeds at the same time. During the speed variation, the energy spent could be expressed as the sum of the energetic cost of the power supply variation plus the energetic cost of the clock frequency variation.

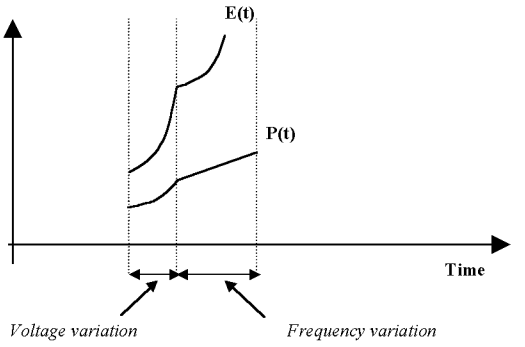


FIGURE 5.11 Energy and power evolution during the speed variation of a synchronous processor.

$$E = E_v + E_f \tag{5.6}$$

Referring to Equation 5.1 and Equation 5.6, the DVS energy cost (see Figure 5.11) is expressed as follows:

$$E = \alpha Cf \int_{t0}^{t1} V^2(t)\,dt + \alpha CV^2 \int_{t1}^{t2} f(t)\,dt \tag{5.7}$$

where α is a scaling factor and the intervals $[t0, t1]$ and $[t1, t2]$ are, respectively, the voltage variation time at fixed frequency and the frequency variation time at fixed voltage.

5.5.6 DVS Algorithms for Asynchronous Processors

A real-time system has often to manage periodic and sporadic tasks. Although periodic tasks are commonly used to process data from sensors and update the current state of the system, sporadic tasks are required to process asynchronous events; however, most of the voltage scheduling schemes presented in the literature consider systems with periodic tasks only. Few attentions have been dedicated to a system with sporadic tasks [63]. The following paragraphs consider both periodic and sporadic tasks.

5.5.6.1 Task Model Definition

Each task can be characterized by a triplet <NIi, Di, Ti>, where NIi is the number of instructions of the task, Di its deadline and Ti its period or its minimum inter-arrival time. We assume that:

- Tasks are independent and their parameters become known when arriving.
- Periodic tasks have deadlines equal to their periods and tasks periods are different.
- At the maximum supply voltage (at highest processor speed), all considered periodic and sporadic tasks can be processed.
- The overhead due to the context switching is negligible.

Because the computation can continue during the voltage switching, we assume that the overhead associated with the voltage scheduling is also negligible.

5.5.6.2 Sporadic Task Voltage Scheduling Algorithm

This subsection considers a case when only sporadic tasks arrive to the system. We assume that the ready time of each task is the instant of its arrival. When a new task τ_i arrives, an acceptance test is performed to determine whether the task can meet its deadline without causing any prior guaranteed tasks to miss their deadlines:

$$\left(\sum_{j\le i}\frac{\overline{NI}_j}{td_j - t}\right) \le S_{MAX} \tag{5.8}$$

where $\overline{NI}_j$ denotes the number of instructions still to be executed for the task τ_j, td_j denotes its deadline, t denotes the current time, and S_{MAX} denotes the highest processor speed in MIPS at the maximum supply voltage. If τ_i is accepted, it is inserted into a priority task queue according to the earliest-deadline-first (EDF) order,* and the voltage-scheduling algorithm updates the processor speed to complete all tasks in the task queue before or at their deadlines. This speed is given by:

$$S = \underset{l\in Q}{MAX}\left[\frac{\overline{NI}_j \sum_{j\ne l / Pl\le Pj} \overline{NI}_j}{td_l - t}\right] \tag{5.9}$$

where Q denotes the stream of the sporadic tasks existing in the task queue, and P_k denotes the priority of task τ_k according to EDF policy. For each task τ_i in the task queue, including the new task, the voltage scheduler computes the required speed S_i to finish the task τ_i considering all priority tasks. Then, it sets the processor speed to the maximum value of S_i. The processor speed is updated whenever a task is added or removed from the task queue. Compared with the voltage scheduling proposed in Pering et al. [63] our algorithm avoids an overestimation of the processing requirements and leads to a higher power saving. This is because our approach takes only runnable and ready tasks into account to compute the operating voltage.

To illustrate the effectiveness of the proposed voltage scheduler, we consider three tasks as presented in Figure 5.12(a), with arrival time and deadlines assigned to each of them. Therefore, when no power reduction technique is applied, the processor runs at S_{MAX} and consumes P_{MAX}. All the speed and power figures reported are based on real measurements performed with an ASPRO motherboard supplied with different voltages.

Using shutdown techniques, the processor can be stopped on completion of task τ_2, woken up at task τ_3 arrival time, and stopped when this task completes. The consumed power is then 42% P_{MAX}. Because an asynchronous processor can instantly be stopped and woken up without any time overhead, all tasks meet their deadlines. In a synchronous system, this technique is ineffective. Because sporadic tasks have random arrival times, it is difficult to predict the future idle times. Thus, tasks can miss their deadlines. Furthermore, shutting down and waking up synchronous processors cause a time and energy overhead.

Voltage scheduling policy is more effective to reduce power consumption in comparison with the shutdown technique. When it is applied, the processor runs at variable voltage and speed as presented in Figure 5.12(b). In this example, the processor runs at 50% S_{MAX} until task τ_1 completes. Then the processor

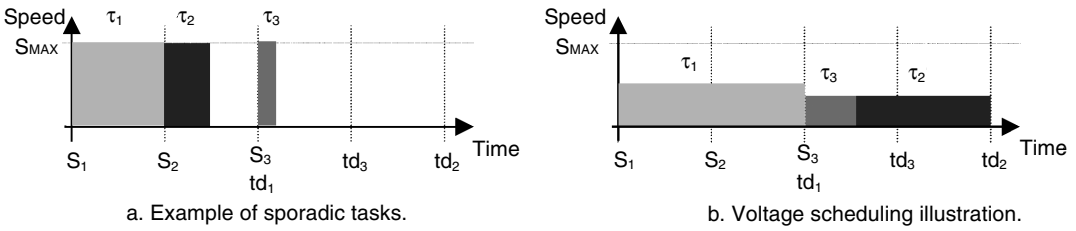


FIGURE 5.12 How sporadic task voltage scheduling reduces power consumption.

*Any scheduling policy is applicable.

TABLE 5.3 Example Periodic Tasks Set

Tasks	NI_j	D_j	T_j	Ready Time
τ_1	0.25×10^6	2	2	0
τ_2	1.0×10^6	5	5	0
τ_3	0.5×10^6	3	3	4

speed is reevaluated and set at 35% S_{MAX} for tasks τ_2 and τ_3. The consumed power is then 14% P_{MAX}. When no task is running, the processor enters a sleeping state in which it consumes no power.

5.5.5.3 Periodic Task Voltage Scheduling Algorithm

This subsection assumes that all tasks are periodic (i.e., no sporadic tasks arrive to the system). We also assume that at $t = 0$, a set of n periodic tasks is ready and sorted into *a priori*ty task queue according to the EDF order. The processor speed is set to:

$$S = \sum_{j=1}^{n} \frac{NI_j}{D_j} \tag{5.10}$$

When a new periodic task τ_i is added to the system, it is inserted into the priority task queue according to the EDF order, and the voltage-scheduling algorithm updates the processor speed to complete all tasks in the tasks queue on their deadlines. The new speed is given by:

$$S = \frac{NI_i}{D_i} + \sum_{j=1}^{n} \frac{\overline{NI}_j}{td_j - t} \tag{5.11}$$

where $\overline{NI}_j$ denotes the number of instructions still to be executed for the task τ_j, td_j denotes its deadline, and t denotes the current time. Similarly, if a periodic task is removed from the system, the processor speed is updated:

$$S = \sum_{j=1}^{n-1} \frac{\overline{NI}_j}{td_j - t} \tag{5.12}$$

Consider the tasks set in Table 5.3, with ready time and deadlines assigned to each task. Speed and power are normalized to their values at the maximum supply voltage: S_{MAX} and P_{MAX}.

In Figure 5.13(a), when the tasks are scheduled running at the highest processor speed, some waiting states exist between the end of a task and the arrival time of the next task. The system then wastes power waiting for the next task. Therefore reducing the supply voltage, such that the tasks make full use of the CPU time, can lower the processor speed.

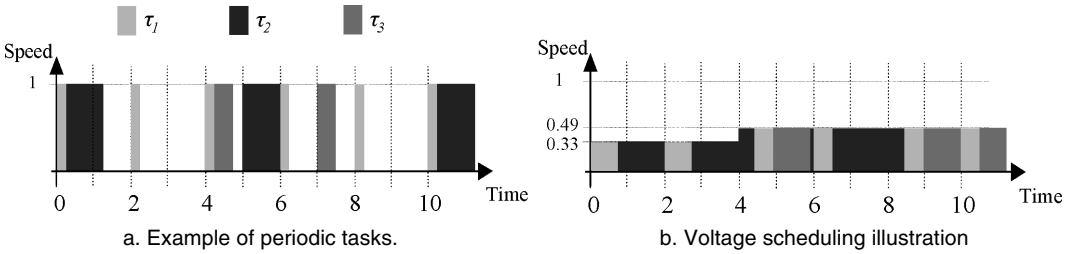


FIGURE 5.13 How periodic tasks voltage scheduling reduces power consumption.

In Figure 5.13(b), the processor speed is reduced to achieve power reduction. When starting, it is set to 33% S_{MAX} according to Equation 5.4. At $t = 4$, task $\tau 3$ becomes ready and the processor speed is updated, according to Equation 5.5, to 49% SMAX. The consumed mean power is then 14% P_{MAX}.

5.5.7 Conclusion

This section demonstrated the ability of asynchronous systems to jointly manage power at the hardware and software levels. Compared with their synchronous competitors, the asynchronous processors are easy to idle and to stop without time and energy overheads. Moreover, a scheduling policy adapted to asynchronous processors has been explained to take full advantage of their properties.

5.6 Conclusion

The goal of this chapter was to give to the reader a better and more global view of the properties and potentials of the asynchronous technology to reduce the power consumption of integrated systems. Starting from the power reduction techniques that can be applied at the structural level of asynchronous circuits, we moved to the software (OS) layers of an embedded system. At the hardware level, reducing the energy consists in reducing the number of transitions required by the computation. This assertion relies on asynchronous circuit properties: they are hazard-free (no energy wasted in spurious transitions) and event-driven (only operating parts consume energy). To minimize the number of transitions, hence the energy, we reviewed a set of techniques based on data encoding, logical and micro-architecture structures. At the system level, it was demonstrated that a relevant exploitation of the asynchronous hardware potentials by the software enables significant power savings that cannot be achieved by a synchronous/clocked implementation. As reported in this chapter, many asynchronous processors have been designed and manufactured. All these prototypes, even though quite complex, have involved low manpower. Regarding their performances, we want to point out that asynchronous processors operate at wide voltage range. Therefore their consumption can be adjusted within a two orders of magnitude range. Moreover, another key feature of asynchronous processors is their ability to promptly shutdown and wake-up.

This suggests adopting a very different point of view for the design of the hardware, the software as well as the running applications of an embedded system. In fact, instead of designing synchronous clock-driven systems, the asynchronous technology gives us the opportunity to design information-driven or event-driven systems. As for the power-aware OS presented in Section 5.5, it was demonstrated that the design of asynchronous algorithms implemented with asynchronous circuits has led to the design of ultra low-power systems, unattainable by clocked systems [68,69]. This extra power reduction originates from the exploitation of the knowledge available on the information to be processed, jointly by the hardware and by the software. This approach can be applied to the design of all subparts of SoCs, processors, peripherals, analog and RF parts, communication network, etc. Removing the clock and moving to asynchronous circuits enables to focus on information processing and design ultra low-power information-specific hardware and software.

References

[1] G. Birtwistle and A. Davis, Eds., *Asynchronous Digital Circuit Design,* Springer-Verlag, New York, 1995.

[2] P. Pénzes and A. Martin, An energy estimation method for asynchronous circuits with application to an asynchronous microprocessor, *Proc. of Design, Automation, and Test in Europe Conf.,* Paris, France, Mar. 2002.

[3] K. Stevens, Energy and performance models for clocked and asynchronous communication, *Proc. of 9th IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp. 56–66, Vancouver, B.C., Canada, May 2003.

[4] J. Fragoso, G. Sicard, and M. Renaudin, Power/area tradeoffs in 1-of-M parallel-prefix asynchronous adders, *Proc. of 13th Int. Workshop on Power and Timing Modelling, Optimization, and Simulation,* Turin, Italy, Sep. 2003.

[5] W.J. Bainbridge and S.B. Furber, Delay-insensitive system-on-chip interconnect using 1-of-4 data encoding, *Proc. of 7th IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Salt Lake City, Utah, pp. 118–126, Mar. 2001.

[6] C. Piguet, M. Renaudin, and T.J.-F. Omnès, Special session on low-power systems on chip (SoC), *Proc. of Design, Automation, and Test in Europe Conf.,* Munich, Germany, Mar. 2001.

[7] W.J. Bainbridge, W.B. Toms, D.A. Edwards, and S.B. Furber, Delay-insensitive, point-to-point interconnect using M-of-N codes, *Proc. 9th Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Vancouver, B.C., Canada, pp. 132–140, May 2003.

[8] R. Manohar, Width-adaptive data word architectures, *Proc. of 19th Conf. on Advanced Res. in VLSI,* Salt Lake City, Utah, pp. 112–129, March 2001.

[9] C.-H. Park, B.-S. Choi, D.-I. Lee, and H.-Y. Choi, Asynchronous array multiplier with an asymmetrical parallel array structure, *Proc. of 19th Conf. on Advanced Res. in VLSI,* pp. 202–212, 2001.

[10] D. Burger, T.M. Austin, and S. Benett, Evaluating future microprocessors: the simplescalar tool set, Tech. Rep. CS-TR-96-1308, University of Wisconsin–Madison, Jul. 1995.

[11] SUN Microsystem Laboratories, Introduction to shade, TR 415-960-1300, Revision A of 1, SUN Microsystem Laboratories, Mountain View, CA, Apr. 1992.

[12] V. A. Bartlett and E. Grass, A low-power asynchronous VLSI FIR filter, *Proc. 19th Conf. on Advanced Res. in VLSI,* pp. 29–39, 2001.

[13] T.E. Williams, Performance of iterative computation in self-timed rings, *J. VLSI Signal Proc.,* 7:17–31, Feb. 1994.

[14] A. Martin, M. Nyström, and P. Penzes, ET2: a metric for time and energy efficiency of computation, in *Power-Aware Computing,* R. Melhem and R. Graybill, Eds., Kluwer Academic Publishers, New York, 2002.

[15] J. Teifel, D. Fang, D. Biermann, C. Kelly, and R. Manohar, Energy-efficient pipelines, *Proc. 8th IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp. 23–33, Apr. 2002.

[16] A. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, The design of an asynchronous MIPS R3000 microprocessor, *Proc. on Advanced Res. in VLSI,* pp. 164–181, Sep. 1997.

[17] A. Efthymiou and J.D. Garside, Adaptive pipeline structures for speculation control. *Proc. 9th IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Vancouver, B.C., Canada, pp. 46–55, May 2003.

[18] M. Lewis, J.D. Garside, and L.E.M. Brackenbury, Reconfigurable latch controllers for low-power asynchronous circuits, *Proc. ASYNC '99,* pp. 27–35, Apr. 1999.

[19] S.B. Furber, D.A. Edwards, and J.D. Garside, AMULET3: a 100-MIPS asynchronous embedded processor, *Proc. of Int. Conf. Computer Design (ICCD),* Sep. 2000.

[20] J. Tierno, R. Manohar, and A. Martin, The energy and entropy of VLSI computations, *Proc. 2nd IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Mar. 1996.

[21] K. Van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, The VLSI-programming language tangram and its translation into handshake circuits, *Proc. European Conf. on Design Automation (EDAC),* pp. 384–389, 1991.

[22] H. Van Gageldonk, D. Baumann, K. Van Berkel, D. Gloor, A. Peeters, and G. Stegmann, An asynchronous low-power 80c51 microcontroller, *4th Proc. Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* San Diego, CA, pp. 96–107, 1998.

[23] D. Edwards and A. Bardsley, Balsa: an asynchronous hardware synthesis language, *The Computer J.,* 45(1):12–18, 2002.

[24] A. Bardsley and D.A. Edwards, Synthesising an asynchronous DMA controller with Balsa, *J. Syst. Architecture,* 46:1309–1319, 2000.

[25] L.A. Plana, P.A. Riocreux, W.J. Bainbridge, A. Bardsley, J.D. Garside, and S. Temple, SPA — a synthesizable AMULET core for SMARTCARD applications, *Proc. of Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp 201–210, Apr. 2002.

[26] A.V. Dinh Duc, J.B. Rigaud, A. Rezzag, A. Sirianni, J. Fragoso, L. Fesquet, and M. Renaudin, TAST CAD tools: tutorial. Tutorial given at the *8th IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Apr. 2002, Manchester, UK, TIMA internal report ISRN:TIMA-RR-02/04/01-FR, http://tima.imag.fr/cis.

[27] K. Slimani, Y. Remond, A. Sirianni, G. Sicard, and M. Renaudin, Estimation et optimisation de la consommation d'énergie des circuits asynchrones, *4ème journées francophones d'étude Faible Tension Faible Consommation (FTFC '2003),* Paris, France, May 2003.

[27a] K. Slimani, Y. Remond, G. Sicard, and M. Renaudin, Test profiles and low energy asynchronous design methodology, PATHOS '04, Santorini, Greece, Sep. 2004.

[28] http://www.synopsys.com/products/etg/powermill_ds.html.

[29] http://www.cadence.com/datasheets/spectre_cir_sim.html.

[30] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus, The design of an asynchronous microprocessor, in *Decennial Caltech Conf. on VLSI,* C.L. Seitz, Ed., MIT Press, pp. 351–373, Cambridge, MA, 1989.

[31] S.M. Burns and A.J. Martin, Synthesis of self-timed circuits by program transformation, in *The Fusion of Hardware Design and Verification,* G.J. Milne, Ed., North Holland, Amsterdam, pp. 99–116, 1988.

[32] A.J. Martin, An asynchronous approach to energy-efficient computing and communication, *Proc. SSGRR 2000, Int. Conf. on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet,* Aug. 2000.

[33] S.B. Furber, P. Day, J.D. Garside, N.C. Paver, and J.V. Woods, AMULET1: a micropipelined ARM, *Proc. IEEE Computer Conf. (COMPCON),* pp. 476–485, Mar. 1994.

[34] I.E. Sutherland, Micropipelines, *Commn. ACM,* 32(6):720–738, Jun. 1989.

[35] S.B. Furber, J.D. Garside, and S. Temple, Power-saving features in AMULET2e, *Power-Driven Microarchitecture Workshop,* Jun. 1998.

[36] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers, Technical report, Universitat Politècnica de Catalunya, Barcelona, Spain, 1996.

[37] J. Garside, W. Bainbridge, A. Bardsley, D. Edwards, S. Furber, J. Liu, D. Lloyd, S. Mohammadi, J. Pepper, O. Petlin, S. Temple, and J. Woods, AMULET3i: an asynchronous system-on-chip. *Proc. Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp. 162–175, Apr. 2000.

[38] W. Bainbridge and S. Furber, An asynchronous macrocell interconnect using MARBLE, *Proc. Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp. 122–132, 1998.

[38a] A.J. Martin, M. Nyström, K. Papadantonakis, P.I. Penzes, P. Prakash, C.G. Wong, J. Chang, K.S. Ko, B. Lee, E. Ou, J. Pugh, E.-V. Talvala, J.T. Tong, and A. Tura, The lutonium: A sub-nanojoule asynchronous 8051 microcontroller, *9th IEEE International Symposium on Asynchronous Systems & Circuits,* Vancouver, Canada, May 12–16, 2003.

[39] A. Abrial, J. Bouvier, M. Renaudin, P. Senn, and P. Vivet, A new contact-less smart card IC using on-chip antenna and asynchronous microcontroller, *J. Solid-State Circuits,* Vol. 36, pp. 1101–1107, 2001.

[40] TIMA laboratory, CIS group Web site: http://tima.imag.fr/cis/.

[41] M. Renaudin, Asynchronous circuits and systems: a promising design alternative, in microelectronics for telecommunications: managing high complexity and mobility (MIGAS 2000), special issue of the *Microelectron.-Eng. J.,* P. Senn, M. Renaudin, and J. Boussey, Guest Eds., Vol. 54, No. 1–2, pp. 133–149, Dec. 2000.

[42] J. Kessels, G. den Besten, T. Kramer, and V. Timm, Applying asynchronous circuits in contactless Smart Cards, *Proc. Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Eilat, Israel, April 2000.

[43] M. Renaudin, P. Vivet, and F. Robin, ASPRO-216: A standard-cell QDI 16-bit RISC asynchronous microprocessor, *Proc. Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp. 22–31, 1998.

[44] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya, TITAC-2: an asynchronous 32-bit microprocessor based on scalable-delay-insensitive model, *Proc. Int. Conf. on Comput. Design (ICCD),* pp. 288–294, Oct. 1997.

[45] M. Gowan, L. Biro, and D. Jackson, Power considerations in the design of the alpha 21264 microprocessor, *Proc. 35th IEEE Design Automation Conf.,* San Francisco, CA, pp. 726–731, Jun. 1998.

[46] M. Es Salhiene, L. Fesquet, and M. Renaudin, Dynamic voltage scheduling for real-time asynchronous Systems, *12th Int. Workshop on Power and Timing Modeling, Optimization, and Simulation (PATMOS),* Sevilla, Spain, Sept. 2002.

[47] Y. Li, G. Patounakis, A. Jose, K. Shepard, and S. Nowick, Asynchronous datapath with software controlled on-chip adaptative voltage scaling for multirate signal processing application, *Proc. IEEE Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* pp. 216–225, May 2003.

[48] L. Nielsen, C. Niessen, J. Sparso, and J. Van Berkel, Low-power operation using self-timed circuits and adaptative scaling of the supply voltage, *IEEE Trans. on Very Large-Scale Integration (VLSI) Syst.,* 2(4):391–397, Dec. 1994.

[49] M. Pedram, Design technologies for low-power VLSI, in *Encyclopedia of Comput. Science and Technol., Vol. 36.* Marcel Dekker, New York, pp. 73–96, 1997.

[50] A. Chandrakasan, S. Sheng, and R. Brodersen, Low-power CMOS digital design, *IEEE J. Solid-State Circuits,* 27(4):473–484, Apr. 1992.

[51] T. Burd and R. Brodersen, Energy-efficient CMOS microprocessor design, *Proc. 28th IEEE Hawaii Int. Conf. on System Sciences,* Vol. 1, pp. 288–297, Jan. 1995.

[52] G. Smit and P. Havinga, A survey of energy saving techniques for mobile computers, Moby Dick technical report, University of Twente, The Netherlands, 1997.

[53] T. Simunic, L. Benini, A. Acquaviva, G. Glynn, and G. De Micheli, Dynamic voltage scaling and power managements for portable systems, *Proc. 38th IEEE Design Automation Conf.,* Las Vegas, NV, pp. 524–529, June 2001.

[54] L. Benini, A. Bogliolo, and G. De Micheli, A survey of design techniques for system-level dynamic power management, *IEEE Trans. on Very Large-Scale Integration (VLSI) Syst.,* 8(3):299–316, Jun. 2000.

[55] Y. Lu, L. Benini, and G. De Micheli, Operating-system-directed power reduction, *Proc. Int. Symp. on Low-power Electronic Design,* Rapallo, Italy, pp. 37–42, July 2000.

[56] Y. Lu and G. De Micheli, Comparing system-level power management policies, *IEEE Design Test of Comput.,* pp. 10–19, 2001.

[57] M. Srivasta, A. Chandrakasan, and R. Brodersen, Predictive system shutdown and other architectural techniques for energy efficient programmable computation, *IEEE Trans. on Very Large-Scale Integration (VLSI) Syst.,* 4(1):42–55, March 1996.

[58] K. Flautner, Automatic monitoring for interactive performance and power reduction, Ph.D. Dissertation, University of Michigan, Ann Arbor, 2001.

[59] T. Pering, T. Burd, and R. Broderesen, Dynamic voltage scaling and the design of a low-power microprocessor system, Power-Driven Microarchitecture Workshop, in conjunction with *Int. Symp. on Comput. Architecture,* Barcelona, Spain, June 1998.

[60] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, A dynamic voltage scaled microprocessor system, *IEEE J. Solid-State Circuits,* 35:1571–1580, Nov. 2000.

[61] M. Weiser, B. Welch, A. Demers, and S. Shenker, Scheduling for reduced CPU energy, *USENIX Symp. on Operating Syst. Design and Implementation,* Monterey, CA, pp. 13–25, Nov. 1994.

[62] K. Govil, E. Chan, and H. Wassermann, Comparing algorithms for dynamic speed-setting of a low-power CPU, *ACM Int. Conf. on Mobile Computing and Networking,* Berkeley, CA, pp. 13–25, Nov. 1995.

[63] T. Pering, T. Burd, and R. Brodersen, Voltage scheduling in the lpARM microprocessor system, *Proc. Int. Symp. on Low-Power Electronic Design,* Rapallo, Italy, pp. 96–101, July 2000.

[64] P. Kumar and M. Srivastava, Predictive strategies for low-power RTOS scheduling, *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors,* Austin, TX, pp. 343–348, Sep. 2000.
[65] M. Fleischmann, Crusoe LongRun power management, Transmeta Corporation, Santa Clara, CA, Jan. 2001.
[66] Transmeta Corporation, *Crusoe Processor System Design Guide,* http://www.transmeta.com/.
[67] M. Renaudin, P. Vivet, and F. Robin, ASPRO: an asynchronous 16-bit RISC microprocessor with DSP capabilities, *ESSCIRC '99,* Duisburg, Germany, pp. 28–31, Sep. 1999.
[68] E. Allier, G. Sicard, L. Fesquet, and M. Renaudin, A new class of asynchronous A/D converters based on time quantization, *Proc. 9th Int. Symp. on Advanced Res. in Asynchronous Circuits and Syst.,* Vancouver, B.C., Canada, pp. 196–205, May 2003.
[69] B. Galilée, F. Mamalet, M. Renaudin, and P.Y. Coulon, Watershed parallel algorithm for asynchronous processor array, *IEEE Int. Conf. on Multimedia and Expo (ICME),* Lausanne, Switzerland, August 26–29, 2002.

6

Low-Power Baseband Processors for Communications

Dake Liu
Eric Tell
Linköping University

6.1 Introduction

Three processors usually exist in a communication terminal system: the digital signal processing (DSP) baseband processor (BBP), the DSP application processor (APP), and the microcontroller (MCU).

Baseband signals are all signals in a radio system, which are not modulated onto the carrier wave. In a cellular phone, this means all signals except those in the radio frequency (RF) part of the phone. This chapter discusses processors for digital baseband signal processing, known as digital baseband DSP processors (DBBP). Figure 6.1 defines the basic partitioning of a radio communication system from both functional and hardware points of view. A DBBP plays an important role in both the transmitter and the receiver. In a transmitter, the DBBP converts the data from application sources to a format adapted to the radio channel. In a receiver, a DBBP recovers symbols from the distorted analog baseband signal and translates them to a bit stream with acceptable bit error rate (BER) for applications. Figure 6.1(b)

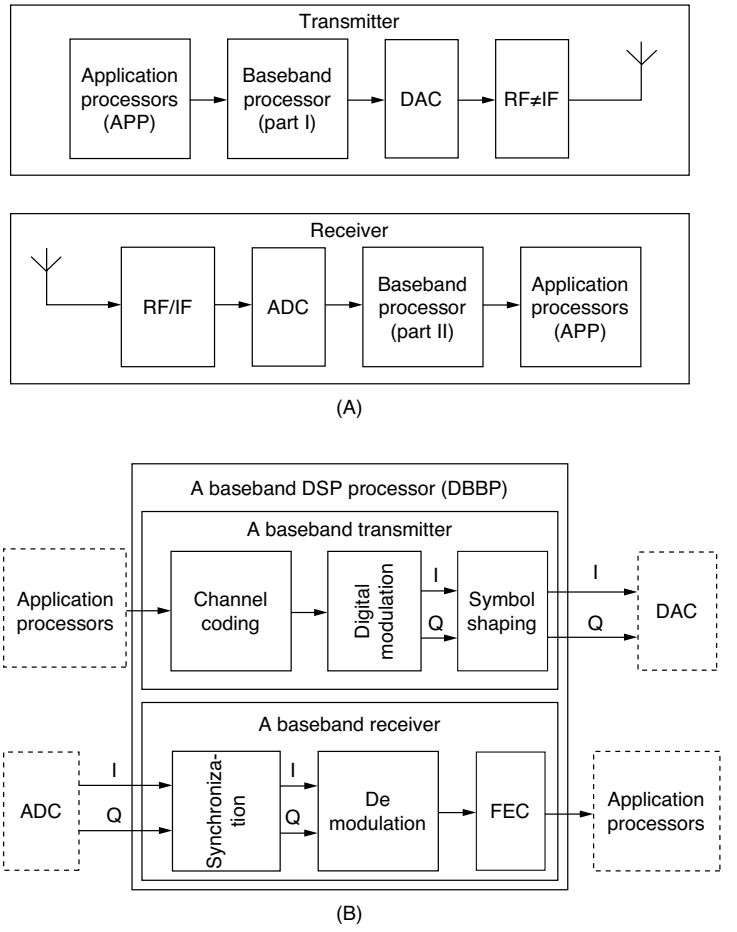


FIGURE 6.1 A radio communication transceiver.

describes functions in a DBBP. The most power-consuming parts are in the receiver, with functions such as synchronization, demodulation, and forward error correction. This chapter introduces the implementation of low-power DBBP. Detailed theory and knowledge of DSP for digital communications may be found elsewhere [2].

6.2 Digital Baseband DSP Processors (DBBP)

6.2.1 Function Coverage

This section gives an overview of the functions needed in a DBBP, using wireless local area network (LAN) as an application example. In principle, what we want to handle in the DBBP is the digital part of the physical layer, which includes all functions between analog to digital converter (ADC)/digital to analog converter (DAC) and the MAC (medium access control) layer interface. Figure 6.1(a) depicts the physical partitioning of a radio transceiver, and Figure 6.1(b) depicts the main functions handled by the DBBP.

6.2.2 The Transmitter

A transmitter performs three major functions: channel coding, digital modulation, and symbol shaping. Channel coding covers different methods for error correction (e.g., convolutional coding) and error detection (e.g., cyclic redundancy check (CRC)). Interleaving is used to minimize the effect of burst errors.

Digital modulation is the process of mapping a bit stream to a stream of complex samples. The first (and sometimes the only) step in the digital modulation is to map groups of bits to a specific signal constellation, such as binary phase shift keying (BPSK), quadrature phase shift keying (QPSK), or quadrature amplitude modulation (QAM). These are different ways of mapping groups of bits to the amplitude and phase of a radio signal). In most cases, a second step, domain translation, is applied. In an orthogonal frequency division multiplexing (OFDM) system (i.e., a modulation method where information is sent over a large number of adjacent frequencies simultaneously), an inverse fast fourier transform (IFFT) is used for this step. In a direct sequence spread spectrum (DSSS) system (e.g., direct sequence-code division multiple access [DS-CDMA], a "spread spectrum" method of allowing multiple users to share the RF spectrum by assigning each active user an individual "code"), each symbol is multiplied with a spreading sequence of ones and minus ones. The final step is symbol shaping, which transforms the square wave to a band-limited signal using a finite impulse response (FIR) band-pass filter. This is necessary to make sure no parts of the transmitted signal are outside the permitted frequency band.

Channel coding and mapping functions operate on bit level (not on word level) and are therefore not suitable for implementation in a programmable processor. In a low-bandwidth transmitter, the symbol-shaping filter can be implemented in firmware. In a high-bandwidth baseband processor, however, a dedicated low-power, low-cost FIR filter circuit is needed.

6.2.3 Synchronization and Channel Equalization

Synchronization in the receiver can be divided into several steps. The first step includes detecting an incoming signal or frame, so called energy detection. In connection with this, operations, such as antenna selection and gain control, are also carried out. The next step is symbol synchronization, which aims to find the exact timing of the incoming symbols. All the preceding operations are typically based on complex auto- or cross-correlations.

In most cases, it is necessary that a receiver performs some kind of compensation for imperfections in the radio channel. This is known as channel equalization. In OFDM systems, this involves a simple scaling and rotation of each subcarrier after the FFT. In a CDMA system, a so-called rake receiver is often used to combine incoming signals from multiple signal paths with different path delays. In some systems, least mean square (LMS) adaptive filters are used. Similar to synchronization, most operations involved in channel estimation and equalization employ convolution-based algorithms. These algorithms are not similar enough to share the same fixed hardware, but they can be implemented efficiently on a programmable DSP processor. If bandwidth and mobility is relatively low, the whole synchronization and equalization flow can be implemented in a programmable DSP processor. For higher bandwidth, high-speed processors with complex multiply and accumulate (MAC) units may be needed.

6.2.4 Demodulation and Forward Error Correction

Demodulation is the opposite operation of modulation. It involves an FFT in OFDM systems and a correlation with spreading sequence (so called despread) in DSSS systems. The last step of demodulation is to convert the complex symbol to bits according to the signal constellation.

Similar to channel coding, deinterleaving and channel decoding are not suitable for firmware implementation. In particular, Viterbi or turbo decoding, which are used for convolutional codes, are very demanding functions.

6.2.5 Comparison with a General DSP Processor

Different kinds of DSP processors are designed for different applications. A DBBP is designed especially for radio baseband signal processing, focusing on synchronization, modulation–demodulation, coding, and forward error correction. A DBBP can be implemented as an application-specific integrated circuit (ASIC) or as an application-specific instruction-set processor (ASIP). An architecture based on a programmable processor with surrounding accelerators is sometimes called a "centralized architecture" as

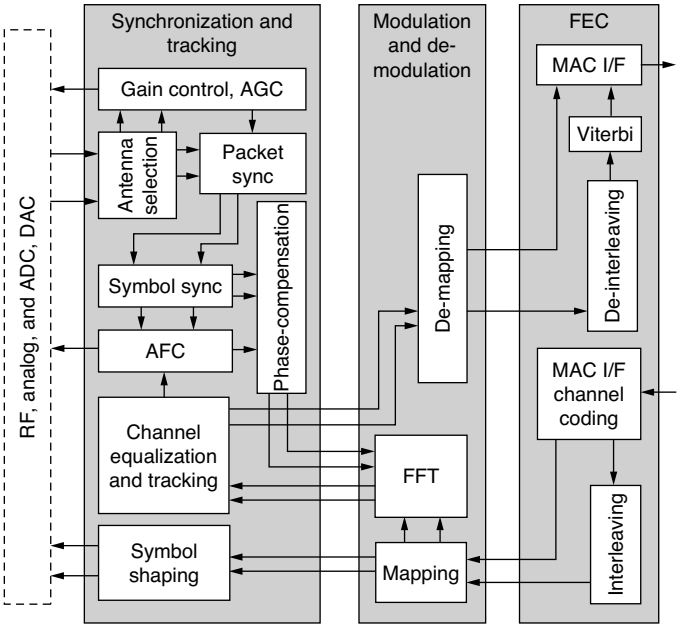


FIGURE 6.2 A baseband processor for IEEE802.11a.

opposed to a "distributed architecture," based on integration of custom circuits. Most current high-bandwidth baseband processors (for third-generation [3G] or wireless local area networks [WLAN]) are implemented as ASICs. One example of a DBBP for IEEE802.11a [4,13] is given in Figure 6.2.

To fulfill a certain level of flexibility and to cover multiple standards, modern low-power DBBPs are approaching a mix of ASIC and centralized ASIP architectures. Recently, the interest in software defined radio (SDR) solutions, which can cover as many standards as possible, has grown. Increased programmability in baseband processors has been the trend in both academia and industry; however, programmability in a DBBP is significantly different from the flexibility required in a general DSP processor. A general DSP processor [3] has to be flexible enough to cater to a large variety of DSP applications on the arithmetic level. Because of this, it has no support for acceleration of specific algorithms (except convolutions). A DBBP, on the other hand, gives dedicated algorithm acceleration for radio baseband DSP processing. For example, to increase performance and decrease control overhead, a baseband processor could have one dedicated instruction to carry out a complex vector operation that would require a subroutine of 20 instructions in a general DSP processor. Table 6.1 and Table 6.2 give the five most often used instructions for general DSP processors and DBBPs, respectively.

6.2.6 Classification of Baseband Processors

Different radio systems require different baseband DSP processors. Some systems, for example global system for mobile communications (GSM, which is the pan-European digital cellular radio standard) and wideband CDMA (WCDMA, which is a 3G mobile phone system) are full duplex (i.e., sending and receiving at the same time), and some, for example, WLAN, is half duplex (i.e., sending and receiving at different times). In some systems (e.g., digital audio broadcasting [DAB] and digital video broadcasting [DVB]), most devices are only receivers.

Requirements on DBBPs can be characterized by two variables, as presented in Figure 6.3. The first variable is the bandwidth. Computing power for synchronization, demodulation, and forward error correction (FEC) increases linearly or faster with the bandwidth. Therefore, the bandwidth has the largest impact on the power consumption of a baseband processor. Another important variable is the mobility. Higher mobility results in faster channel fading, which requires more processing to recover from channel

TABLE 6.1 Five Most Often Used Instructions of a General DSP Processor

Instructions	Functional Specification
Multiplication and accumulation	Accumulator <= accumulator + register 1 * register 2
Multiplication and round arithmetic in accumulator	Register 3 <= round (register 1 * register 2) Accumulator <= arithmetic operation (accumulator)
Memory to/from register	Memory [address pointer] <= (or =>) register
Register to/from accumulator	Accumulator <= (register 1, register 2) or Register <= round (accumulator)

TABLE 6.2 Most Often Used Instructions of a Baseband DSP Processor

Instructions	Functional Specifications
Conjugate complex convolution (auto correlation)	For I = 1 to N do (Complex Reg <=) Complex REG + V1[i] * Conjugate of V1(or 2)[i]
Complex convolutions	For I = 1 to N do {Complex REG <= Complex REG + V[I] * V2[i]}
Conjugate complex vector product	For I = 1 to N do (V3[i] <= V1[i] * Conjugate of V2[i])
Complex vector product	For I = 1 to N do (V3[i] <= V1[i] * V2[i])
Lookup table	REG2 <= Memory [Segment + REG1]

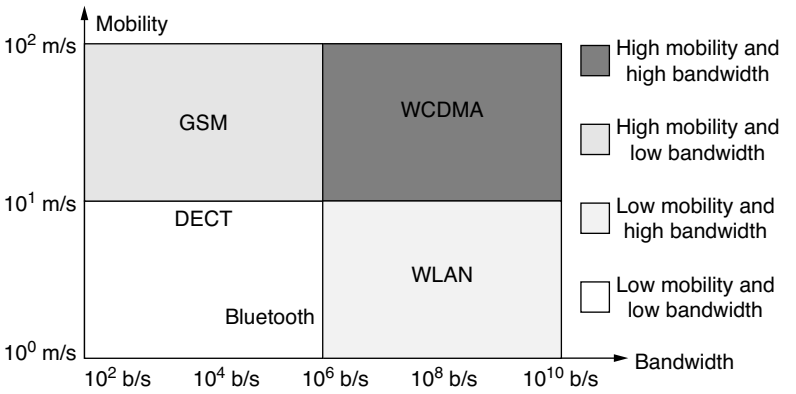


FIGURE 6.3 Classification of baseband DSP processors.

distortion. For a fast fading channel, the receiver may have to be updated for every received symbol; whereas with a slow fading channel, it may be enough to update the channel equalizer for every new packet. Because channel estimation and equalization often are among the most demanding operations in the receiver, the mobility requirements have a large impact on power consumption. Another parameter that should be considered is the dynamic range. This depends on the ratio of bit rate over symbol rate and on the distance over which transmissions may take place. With larger dynamic range, higher data precision is required in the receiver, which leads to higher power consumption.

6.3 Design of Low-Power Radio Baseband DSP Processors

6.3.1 Basic Principles for Low-Power Design

Dynamic power consumption of digital complementary metal oxide semiconductor (CMOS) circuits is $P = aCfV^2$, where a is the activity, C is the total load capacitance, f is the toggling frequency, and V is the supply voltage and the swing of the digital signal. Based on this formula, the principle of low-power design is to eliminate extra toggling, to decrease power supply voltage, and to minimize the overall capacitance. Elimination of extra toggling includes selecting a low-power datapath (data flow) architec-

ture, selecting suitable precision for data processing units, eliminating control overhead, minimizing memory access, using operand stopping techniques, and designing for clock gating. Parallelization and pipelining are the major ways to reach low supply voltage. If the extra latency introduced by pipelining is not acceptable, extra hardware acceleration may be needed. If we disregard physical scaling, capacitances on silicon can be eliminated by shrinking/optimizing data and computing precision, by hardware acceleration, and by smart hardware multiplexing (sharing the same hardware by multiple functions in different time slots).

6.3.2 Trade-Off between Programmability and Fixed Function Hardware

At least three major advantages come from programmability: flexibility throughout the product lifetime, shorter debugging time, and a certain level of design error tolerance through the possibility of modifying firmware. It is also easier to share or multiplex a programmable device between multiple functions. On the other hand, extra power consumption is introduced by program memory and extra control circuits compared with an architecture where functions are direct mapped to fixed hardware. Due to the lack of hardware multiplexing, much more silicon is required by the direct mapped architecture. A trade-off between a programmable device and fixed function hardware is configurable devices that can be reused for similar functions. We know from previous sections that a programmable device is suitable for jobs related to synchronization and equalization. We also know that some other jobs, including modulation/demodulation and coding/decoding, are not suitable for firmware implementation. Instead, dedicated configurable circuits are a better solution for these jobs. We therefore conclude in this section that it is suitable to use a programmable device for synchronization and equalization and for miscellaneous jobs.

6.3.3 Nonprogrammable Low-Power Baseband Processor Architecture

Different kinds of architectures are used for different baseband applications, as well as for different trade-offs between power consumption and flexibility. FEC, for example, Viterbi decoders or turbo decoders, are always executed by dedicated hardware. The rest of the functions could be located in either a programmable or a nonprogrammable processor, according to product requirements. In this section, we discuss nonprogrammable DBBP for WLAN applications. One example is the baseband processor for HiperLAN2 and IEEE802.11a from interuniversitair micro-elektronica centrum (IMEC) Belgium [7]. Another example is the baseband and MAC processor from Atheros in Sunnyvale, California [6]. Nonprogrammable DBBPs are implemented by mapping algorithms to dedicated hardware. Because WLAN systems are a half-duplex, sending and receiving jobs are not processed simultaneously, so sharing hardware between the transmitter and the receiver is possible. Furthermore, because of their different execution times, sharing hardware between preamble and payload functions is also feasible. As another example of a nonprogrammable DBBP, Figure 6.4 gives an example of a converged DBBP for IEEE802.11a and IEEE802.11b [5].

The upper part of Figure 6.4 describes the IEEE802.11a functions divided into four data flows. The upper flow, which is the first to be activated, takes care of the so-called short pilot signals (10 equal symbols of 16 samples each). These are used mainly for energy detection, antenna selection, gain control, and synchronization tasks. The next flow operates on the so-called long pilots (2 equal symbols of 64 samples each), which are used for channel estimation and for fine-tuning the synchronization. Next, the main receiver flow takes care of the payload data symbols. The important steps here are I-Q-phase compensation (carried out by a rotor), FFT, channel equalization, demapping, deinterleaving, and forward error correction. After theses steps the resulting bit stream is handed over to the MAC layer. The final flow is the transmitter flow, which consists of forward error coding, interleaving, constellation mapping, IFFT, and the symbol-shaping filter. By multiplexing hardware of the four flows, the receiver filter is used by all receiver flows, and the FFT engine is used by the long pilot, payload reception, and transmitter flows. The same hardware is also used for both interleaving and deinterleaving.

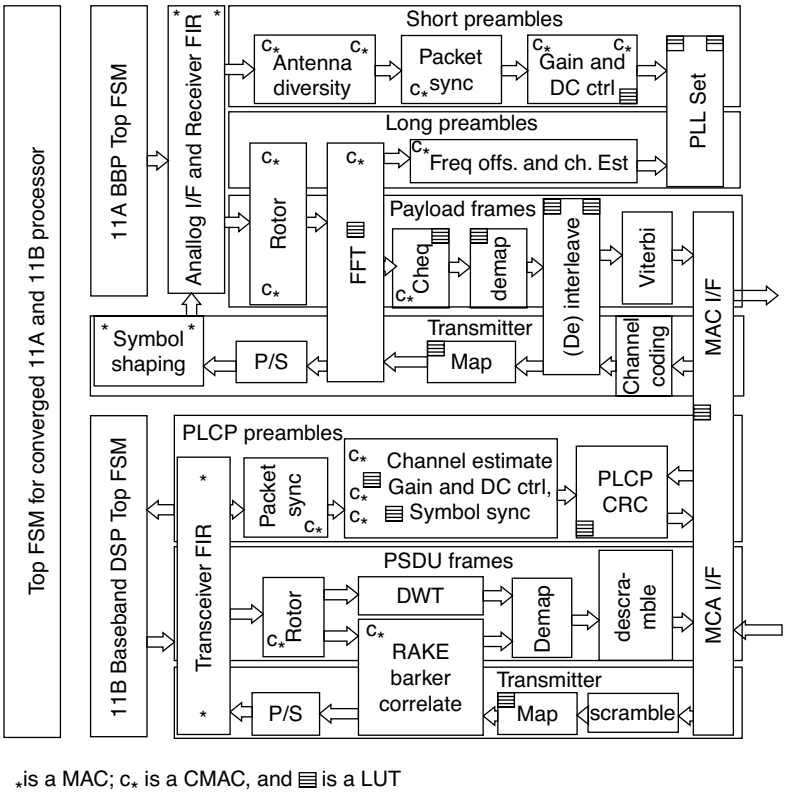


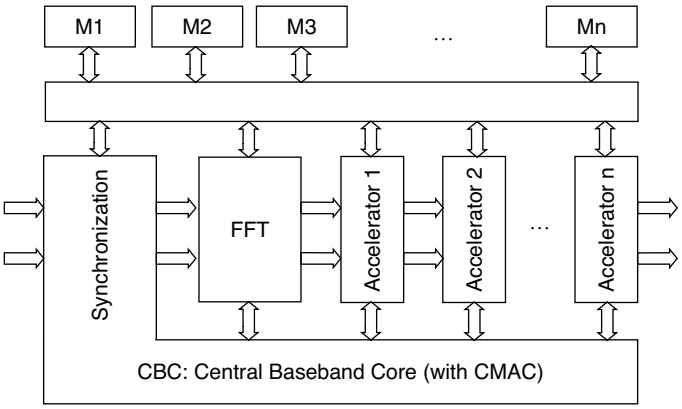
FIGURE 6.4 Mapping functions to nonprogrammable BBP architecture.

The lower part of Figure 6.4 depicts the IEEE802.11b functions, which are divided into three flows. The first flow operates on the preamble and takes care of energy detection, gain control, and synchronization. The second flow handles the payload data and has functions such as I-Q-compensation, channel equalization, despreading (which is either a correlation or a Walsh transform depending on the transmission mode), demapping, and descrambling. The third is the transmitter flow, which does scrambling, mapping, spreading, and symbol shaping. Possible candidates for hardware multiplexing are the receive filter and the scrambler/descrambler. The MAC layer protocol is identical for the two standards, so all MAC functions are also multiplexed.

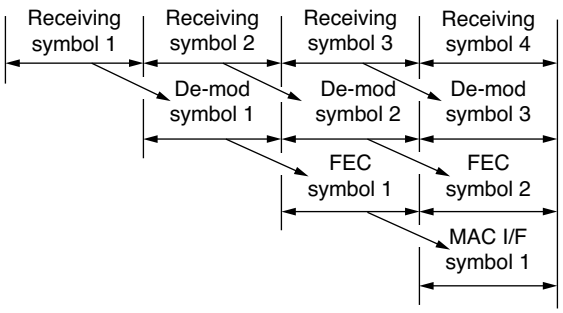
The high degree of parallelization in this architecture enables low clock frequency (~50 MHz) and thereby makes it possible to lower the supply voltage. This architecture will give minimum power consumption at the price of no flexibility and high silicon cost. Architectures, such as the one in Figure 6.4, need approximately 17 complex multipliers, 15 lookup tables, and many other complex hardware blocks. Obviously, the total silicon cost will be high.

6.3.4 Programmable Baseband Processor (PBP) Architectures

In a converged baseband processor (e.g., a converged solution for GSM/GPRS/3G [1] or IEEE802.11a/b/g [8]), a custom circuit implementation will require more silicon area than a centralized ASIP solution because the hardware (HW) multiplexing does not cross between standards. To implement different standards with multiple modes of operation (i.e., preamble reception, payload reception, and transmission) and different data rates, a high degree of dynamic reconfigurability is required. The control will be



(A) The PBP architecture



(B) The pipeline scheduling example of 11A payload process

FIGURE 6.5 A centralized PBP.

very complex. To reduce the complexity of the control path and reach the required flexibility, a programmable solution will be necessary.

Programmable solutions were investigated in Sengupta et al. [1] and Tell and Liu [8]. To support high millions of instructions per second (MIPS) capacity and low computing induced latency, centralized and scalable architectures are proposed by both academia and industry. By "centralized," we mean that a central programmable processor manages the DSP flow and some of the DSP functions. By "scalable," we mean that multiple accelerators and memories can be added. Figure 6.5(a) gives an example of such a solution. CBC in Figure 6.5(a) stands for "central baseband core." The CBC functions as the master of the system controlling the slave accelerator blocks. Accelerator and memory blocks are connected by a connection network, which is also configured by the CBC.

By analyzing the essential baseband algorithms listed in Figure 6.2 and the most often used instructions in Table 6.2, we have found that tasks suitable for CBC are the top DSP job flow management, complex vector computing, and lookup table functions. Therefore, a suitable CBC architecture will be a combination of a compact complex MAC and an arithmetic and logic unit (ALU) datapath. The complex MAC, which computes a complex multiplication, $(A_R + jA_I)^*(B_R + jB_I)$ in one clock cycle and complex accumulation in one clock cycle, supports complex vector computing (i.e., complex convolution, conjugate complex convolution, and complex vector dot product) [9]. The ALU will be used to support DSP job flow control. In the programmable baseband processor (PBP) of Figure 6.5, the CBC manages the following functions:

1. Running the top- and sublevel program flows, configuring the connection bus and the memory partitioning, initializing the accelerators, and implementing other miscellaneous controls.
2. Data quality control including scaling, antenna control, frequency offset estimation, and AGC/AFC to analog baseband.

TABLE 6.3 Kernel Instructions of the CBC

Operation Functions	Operand A	Operand B	Results
	Complex Arithmetic		
Normal MAC	Vector MEM A	Vector MEM B	Accumulator
Conjugate MAC	Vector MEM A	Vector MEM B	Accumulator
Vector energy	Vector MEM A	Vector MEM A	Accumulator
Vector product	Vector MEM A	Vector B	V-MEM C (A)
	Real Arithmetic		
LUT			Register file
±, compare	Register file	Register file	Register file
++, --	Register file	Register file	Register file
Shift/logic	Register file	Register file	Register file
	Flow and Control		
Configuration	Register/constant		Control register
Move			
Call/return			
Conditional jump			

3. Packet and symbol synchronization, channel estimation, and equalization — This requires complex vector computations including complex (conjugate) convolution, dot products, vector square-sum, and lookup tables (LUTs) for sine, arctangent, and 1/x.

The CBC runs two threads in parallel:

1. Program flow and miscellaneous jobs
2. Complex vector computations

The programmability gives several advanced features. The first feature is the excellent opportunity for convergence. For example, by adding an FFT accelerator and a rake receiver accelerator, both OFDM and CDMA standards are supported. This freedom of convergence decreases the time required for adapting to new market requirements. The second important feature is the hardware multiplexing. Most baseband algorithms can be implemented by the instructions in Table 6.2 on a complex multiply and accumulate (CMAC) in the CBC. The CBC can support most baseband functions with low silicon area cost if the firmware in the program memory can be synthesized into logic gates during the silicon backend design. For example, in a converged IEEE 802.11a/b/g receiver, approximately 2 complex MAC units will be sufficient, while the nonprogrammable solution may need approximately 17. The third advanced feature, and possibly the most important feature when it comes to implementation, is the relatively short time needed for system verification. Firmware verification is much easier than HW verification. Changing firmware can solve most small problems. The firmware design iteration time can be measured in minutes, compared with the long hardware iteration time, which could be about a day or more.

The instruction set of the CBC is given in Table 6.3. Because it is an ASIP, the instruction set of the CBC should be as simple as possible. Most complex mode arithmetic instructions are vector instructions. Real arithmetic instructions are used for table-based arithmetic acceleration and miscellaneous functions.

Figure 6.6 demonstrates how the functionality of an IEEE802.11a/b transceiver is allocated to a PBP. The fill patterns illustrate the functions allocated to different accelerators. For example, functions marked with the white fill pattern are allocated to the CBC and functions marked with horizontal lines are allocated to the FFT accelerator. To relax the load in the CBC, an extra CMAC is added as a configurable accelerator for vector processing. Also for example, part of the channel equalization could be allocated to this extra CMAC. The system clock frequency is relaxed by using a radix-4 FFT accelerator [12] and by off-loading CBC jobs to the extra CMAC. In the OFDM flow, accelerators could be used for FIR and analog/MAC interface, FFT, interleaver/deinterleaver, and FEC including Viterbi decoder, CRC, and

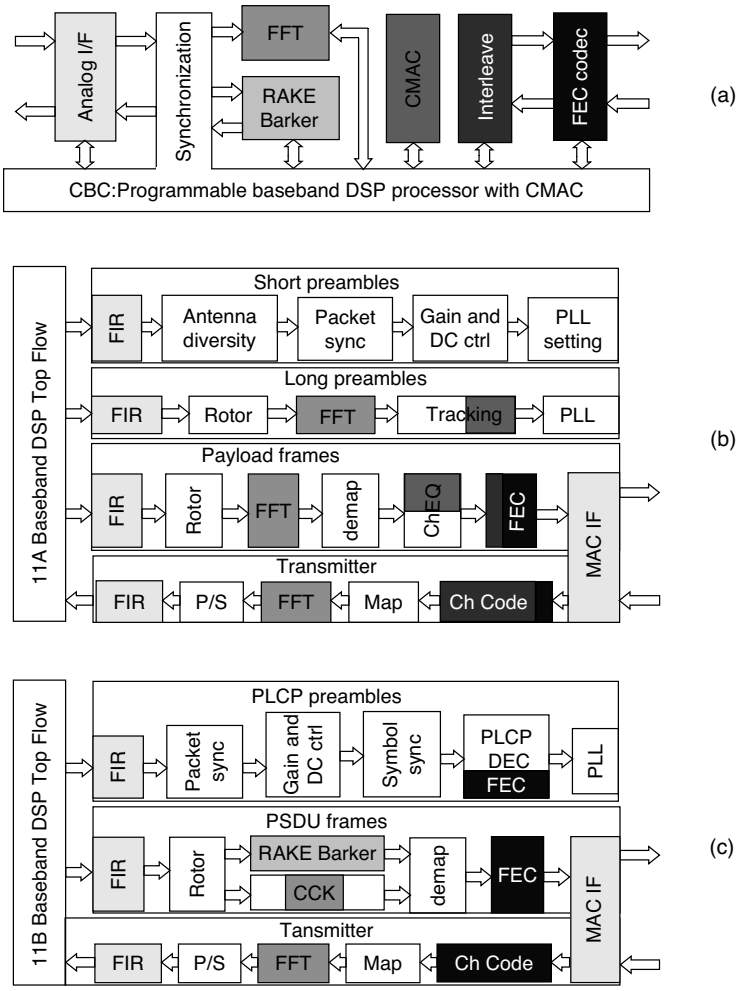


FIGURE 6.6 Mapping functions to PBP architecture.

scrambling. The DSSS transmission of IEEE std. 802.11b shares the FIR/interface and CRC/scrambling accelerators with the OFDM flow and uses an extra accelerator for spread/despread.

6.3.5 PBP Design Challenges

The main problem during the HW partitioning and integration is the memory access cost while delivering computing buffers from one processor/accelerator to another processor/accelerator. These extra memory accesses consume both extra power and execution time. Latency is especially a problem when jobs are divided and run in parallel at a lower clock frequency. Memory size and memory usage optimizations are important [8,17]. The most important memory size optimization in a baseband processor is the minimization of variable lifetime. General variable reuse techniques are very important when using cache as the computing buffer. When using only scratch pad memory for data stream processing in a baseband processor, most variable lifetimes are explicit and fixed according to the baseband data flow. What we need here is to minimize the memory access operations to minimize both the memory cost and run time. This optimization should be done during task level scheduling. In Abnous [15] and Tell and Liu [8], memory operations for delivering a computing buffer can be eliminated by reconfiguration of the memory connection network. When an algorithm is assigned to a computing unit (a processor or an

accelerator) and a group of memories, one memory must be assigned as the result buffer. In the next stage, the memory connection network is reconfigured so that the result buffer of the previous unit becomes the input buffer for the new unit. In this way, the memory cost and the memory access time are minimized. This procedure, which we call "computing buffer delivery technique" [8], decreases both power consumption and latency. To implement this technique, a comprehensive methodology for system reconfiguration and integration is necessary. The research project Pleiades at the Department of Electrical Engineering and Computer Science (EECS) in the University of California-Berkeley, gives an architecture and methodology for reconfiguring and reconnecting computing devices and memories according to the dataflow at hand. In the Pleiades architecture, system reconfiguration and reconnection is performed via a system on chip connection network that includes a data network and a control network. The control network is used to configure the data network. In our architecture [8], the CBC controls the connections by writing a configuration vector to bus connection registers, which are addressable control and configuration registers.

Another major challenge is the design of the CBC-accelerator interface. The CBC is made for relatively general baseband jobs and is designed before the accelerators are specified. The interface must allow easy plug-in of accelerators of varying type without modification of the hardware. We need to send control and data from CBC to accelerators and results from accelerators to the CBC.

A bus architecture gives the required scalability and device address space. When we plug in an accelerator to the bus extension, the added hardware is seen as an ordinary execution unit of the processor. The design idea is to define a simple and robust protocol for accelerator integration. Giacalone [16] gives a good acceleration methodology. In the TIC55, scalability is given by instruction set architecture extension. This is also known as tightly coupled HW acceleration. In TIC55 and in our CBC, the extension is defined by a protocol accepted by both the processor and the accelerators. A set of special accelerator instructions is needed. Each accelerator instruction is divided into a common part and a custom part. The common part gives address and bus control, and must be decoded by both the processor and the accelerator. The custom part carries control codes that are only decoded by the accelerator. Because the CBC does not decode the custom control code, the processor can easily be upgraded by adding a new accelerator. In addition, no restrictions are made regarding the kind of accelerators that can be added.

6.3.6 Decreasing Supply Voltage

Lowering the supply voltage is the most effective way to decrease power because the reduction of power consumption follows the square rule; however, three constraints (i.e., the throughput required by the baseband specification, the limitations on the computing latency, and the limited choices of supply voltage on system level) have to be met while reducing the supply voltage. In this section, we aim for lowering the supply voltage thus satisfying throughput and computing latency requirements.

To decrease supply voltage, we should start by analyzing the system scheduling and finding the timing critical path. By accelerating the timing critical path, we get a chance of lowering the supply voltage. The following paragraph gives an example based on the system in Figure 6.6. We first allocate jobs to execution time slots following the constraints from the MAC layer specification. After job allocation according to the task pipeline definition in Figure 6.5(b), we find that the timing critical path is the demodulation step, which includes rotor, FFT, channel equalization, and demapping. We modify the schedule by moving the rotor to the receiving time slot (the receiving time slot now includes sampling, buffering, receiver FIR filter, and rotor). Next, we speed up the remaining operations in the demodulation step. By replacing the radix-2 FFT by a radix-4 FFT, the computing time is decreased from about 200 cycles to 52 cycles [12]. The channel equalization takes about 200 cycles to run in the CBC. Adding one more CMAC and allocating half of the channel equalization job to it decreases the cost for channel equalization to about 100 cycles. The demapping for 64-QAM takes about 1000 cycles if there is no instruction level acceleration. By specifying a special demapping instruction for 16-QAM and 64-QAM, the cycle cost for 64-QAM becomes about 144 cycles. Including about 200 cycles for miscellaneous jobs,

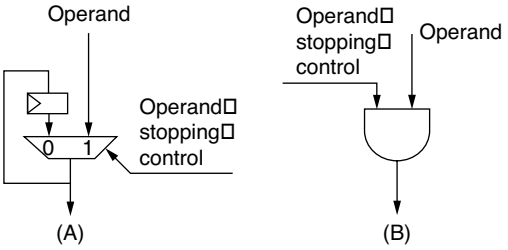


FIGURE 6.7 Operand-stopping technique.

the cycle cost of the payload processing has decreased from about 1644 cycles to about 498 cycles. This means that the system clock can be decreased from 416 MHz to 125 MHz. If the critical path at circuit level is given by the 10-bit CMAC, about 250 MHz can be reached using a 0.15-μm digital CMOS technology running on the nominal supply voltage. The fact that we only need 125 MHz means that we have a chance to lower the supply voltage by about 40%. The corresponding power reduction factor is 2.7. The actual power reduction factor is approximately two because of the extra power cost from the added hardware.

6.3.7 Eliminating Unnecessary Switching

Three possibilities are available for reducing logic switching: minimizing the memory access, minimizing the computing power, and minimizing the control power. Minimizing memory accesses has already been discussed in this chapter. About 10% power and execution time can be saved by eliminating the passing of computing buffers [8]. Operand stopping is a technique where operands are stopped from propagating through a bus to nonactive logic blocks. Two basic operand-stopping techniques are given in Figure 6.7. The method, given in Figure 6.7(a), stops the operand by keeping it in a register. It is used for stopping the logic toggling involved in single step instructions. The method given by Figure 6.7(b) stops the operand by masking it with AND or OR gates. It is used for stopping the logic toggling involved in vector mode instructions. By using operand-stopping technique, a datapath can be divided into several active regions. The operand stopping control signal could come from the decoded instruction.

Clock gating is another technique for achieving low-power consumption. During architecture and function level design, the hardware should be partitioned into blocks suitable for clock gating. The instruction decoding can then generate clock gating control signals to different blocks. Although a part of the circuit is not active, the clock to flip-flops in this part will be gated or shut down. It should be noted that the reset signal should be valid to every flip-flop even when the clock is gated. Therefore, asynchronous reset is preferred.

6.3.8 System-Level Power Management

If it is possible to use a controllable power switch or DC-DC converter for the WLAN subsystem, three levels of power management will be possible: power supply on-off control, system clock on-off control, and circuit clock on-off control. The supply on-off control and the system clock on-off control are managed on system level beyond the the programmable core of the baseband processor. Clock on-off control for circuits in a DBBP can be managed inside the DBBP or inside the accelerators. The control could be handled implicitly or explicitly. Explicit control is managed by running instructions such as no-operation (NOP) and sleep and wait for external interrupt (SLEEP). The instruction decoder manages implicit control. Because every instruction activates only certain parts of the processor, the instruction decoder can control clock gating and operand stopping inside the CBC. Power management is a general issue in low-power design, and detailed implementation techniques can be found in other chapters in the book.

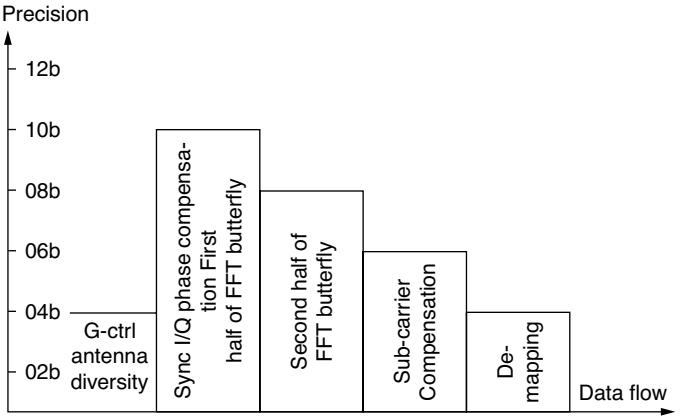


FIGURE 6.8 Variable precision in an IEEE 802.11a baseband processor.

6.4 Case Study One: Variable Data Length and Computing Precision

The power consumption increases with the data precision. In a nonprogrammable architecture, each step in an algorithm is assigned to a specific computing or storage unit. To save power, each unit is designed with the minimum acceptable data width. In a programmable BBP, however, the computing unit must be designed for the largest required precision.

To decrease the power consumption in the datapath of a programmable processor, dynamic precision firmware is introduced in the CBC [10]. In the algorithm design phase for the WLAN baseband firmware, we need to optimize precision for every step of all algorithms. When reduced precision can be used for a step of an algorithm, we will mask operands to a limited precision. Figure 6.8 demonstrates how different precision can be assigned for different algorithms in a receiver. (Data and computing precision is not specified after demapping because the remaining operations are bit level operations.) This chapter focuses on symbol processing because the precision of symbol processing algorithms has significant impact on the receiver quality.

We have estimated the switching power in a MAC unit with different masks applied and with different datapath precision. The results are given in Table 6.4. The right part of the table gives power consumption from 6 MAC units with different data width. The left part of the table gives the difference of the power consumption by masking the same MAC units, which is a 16-bit MAC with 32-bit accumulator. Note, for example, that replacing a 16-bit MAC operation with a 12-bit operation saves more than 50% power, although the same hardware is used. Furthermore, no major additional power saving can be made by replacing the 16-bit MAC unit by 12-bit hardware. We conclude that reducing the precision of computations can save a significant amount of power, even if a high-precision processor is used.

TABLE 6.4 Relative Power Consumption Measured from Masked MAC

Masked Operands on a 16-bit MAC		MAC Units with Different Precisions	
Mask Precision	Relative Power	Datapath Precision	Relative Power
16-bit (no mask)	1.00	16-bit (32-bit accumulator)	1.00
12-bit (mask 4-LSB)	0.47	12-bit (24-bit accumulator)	0.41
10-bit (mask 6-LSB)	0.31	10-bit (20-bit accumulator)	0.26
8-bit (mask 8-LSB)	0.18	8-bit (16-bit accumulator)	0.12
6-bit (mask 10-LSB)	0.09	6-bit (12-bit accumulator)	0.06
4-bit (mask 12-LSB)	0.04	4-bit (8-bit accumulator)	0.02

6.5 Case Study Two: Hardware Architecture for a Block Interleaver

6.5.1 Introduction

This case study introduces a low-power interleaver/deinterleaver architecture and compares it to the conventional hardware implementation. Excellent power saving was reached through significantly reduced toggling and power supply voltage reduction.

Interleaving is an operation often carried out as part of the channel coding in a radio system. The purpose of interleaving is to distribute transmitted bits in time, frequency, or both. Consecutive bits on the input stream should not be transmitted consecutively in time (or on the same frequency in an OFDM system). Interleaving reduces the effect of burst errors caused by fast fading. The requirements for interleaving depend both on the modulation and error correction schemes used, and on the channel characteristics.

One common method for interleaving is to use a block interleaver. A block interleaver operates on one block of data at a time, and no interleaving occurs between blocks. A block interleaver is typically implemented by writing the bits into a matrix row by row and then reading them column by column. Deinterleaving is simply the reversed operation — writing column by column and reading row by row. However, a real interleaver implementation could be more complicated. The following steps can define a more general block-interleaving algorithm:

1. Write bits to matrix row by row
2. Perform intra-row permutations
3. Perform intra-column permutations
4. Read bits columns by column

6.5.2 Traditional Interleaver Implementation

Figure 6.9(a) is a traditional interleaver implementation based on LUTs. This implementation can support any interleaving scheme. A sequential memory is used and a ROM LUT stores the specified interleaving sequence. For interleaving operation, the input bits are first written sequentially to the memory and then read in the order defined by the LUT. For deinterleaving, the bits are written according to the LUT and then read sequentially. The number of bits in the data memory is equal to the largest block size, and the number of words in the LUT is at least equal to the sum of all block sizes the interleaver should handle.

The main advantages of the traditional implementation are that it is simple, straightforward, and general. The main disadvantage of this implementation is that the bits have to be written and read one at a time, resulting in a high cycle cost. Another disadvantage is that the complete interleaving sequence of every interleaving scheme has to be stored explicitly in the LUT, making it relatively large if many standards have to be supported.

6.5.3 A New Block Interleaver Implementation

This section introduces a new implementation of a multi-standard block interleaver [14]. The purpose of the new implementation is to avoid the need for addressing individual bits in the memory and enable read and write of several bits in parallel. Generally, this scheme executes an interleaving of R rows and C columns in R + C clock cycles, while the old scheme needs two RC clock cycles. Figure 6.9(b) presents the main idea of the new architecture. It is based on a special matrix memory block where words are written to rows but read from columns.

A complete row can be written and a complete column can be read in one clock cycle. Intra-row permutations are carried out before the bits are stored to memory by simply reordering the bits on the input data bus. In the same way, inter-column permutations are carried out by reordering the bits on the output data bus after the data has been read.

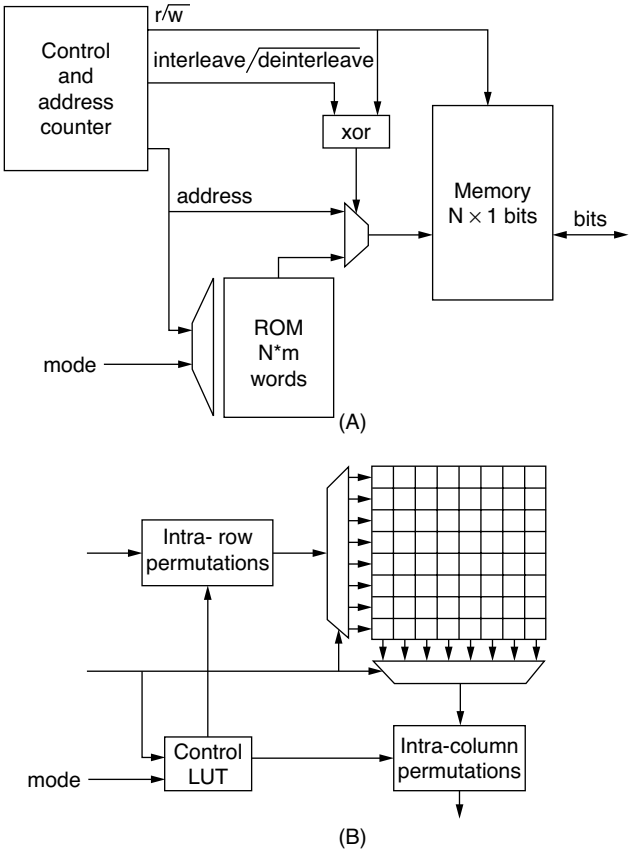


FIGURE 6.9 Two interleaver implementations.

If a small number of different permutation schemes are needed, the permutation blocks are a simple set of multiplexers. A more general interleaver may require a more intricate permutator.

A control block may be needed to set up the permutation block according to the current mode (and address). This could be implemented as a small LUT.

6.5.4 Hardware Implementation

An interleaver for the WLAN standard IEEE 802.11a has been implemented using the two previously discussed schemes. The IEEE 802.11a standard uses four different interleaving schemes with block sizes up to 16×18 bits and uses three different intra-column permutation schemes. The two implementations of the interleaver were both implemented in VHDL and synthesized to a 130-nm process. The result in terms of area and speed can be found in Table 6.5.

6.5.5 Power Issues

The new implementation is superior to the traditional one not only in performance, but also in power consumption for several reasons:

1. Most of the energy will be consumed in the data memory. Because we need to handle both 16×18 and 18×16 matrices (for interleaving and deinterleaving respectively), the number of physical memory cells is larger in the new scheme (324 vs. 288 in the old scheme). However, the number of active cells is the same in both schemes. Furthermore, because the new scheme uses much

TABLE 6.5 Interleaving implementation results

Feature	Old Architecture	New Architecture
Interleaving for 6–9 Mb/s transmission (BPSK, block size 3×16)	96 cycles	19 cycles
Interleaving for 36-54 Mb/s transmission (64-QAM, block size 18×16)	576 cycles	34 cycles
Approximated max frequency	250 MHz	500 MHz
Data memory area	0.0197 mm^2	0.0189 mm^2
Other area	0.0053 mm^2	0.0018 mm^2
Total area	0.0250 mm^2	0.0207 mm^2

smaller address decoders (18 addresses instead of 288) the power needed for addressing each bit is smaller in the new scheme.

2. The sequence ROM in the traditional scheme consumes significant power. For the 802.11a interleaver, the ROM size is 624×9 bits. (If the LUT in the new scheme was implemented in a ROM in the most naive way, it would need 128×4 bits, but it is possible to make it much smaller because the permutation does not depend on the address in three of the four modes.)
3. Because the new scheme needs fewer clock cycles, it can run at a much lower frequency, thereby the supply voltage can also be lower.
4. More power is also consumed in, for example, control logic in the traditional scheme, simply because it has to run for many more clock cycles.

6.6 Conclusion

A baseband DSP processor is an application specific processor dedicated for baseband signal processing. Requirements on a baseband processor vary a lot according to differences in dynamic range, channel fading, bandwidth, and applications (voice or data). Low-power design is essential in baseband processors for radio terminals. In addition to conventional low-power design rules, low-power design for baseband processors especially focuses on optimizing task level partitioning, scheduling, and parallelization. Minimizing memory size and memory accesses is essential. Functional acceleration is the key factor to achieve the balance between low-power consumption and flexibility. Variable precision, operand-stopping, and clock-gating techniques are also important ways of achieving low power.

References

[1] Sengupta, C. et al., The role of programmable DSPs in dual mode (2G and 3G) handset, in *The Application of Programmable DSPs in Mobile Communications,* Gatherer, A. and Luslander, E., Eds., John Wiley & Sons, New York, chap. 3, pp. 23–40, 2002.

[2] Gibson, J.D., *The Mobile Communication Handbook, 2nd ed.,* CRC Press, Boca Raton, FL, and IEEE Press, 1999.

[3] Lapsley, P., Bier, J., Shoham, A., and Lee, E.A., *DSP Processor Fundamentals, Architectures, and Features,* IEEE Press, 1997.

[4] IEEE Std. 802.11a-1999, High-speed physical layer in the 5-GHz band.

[5] IEEE Std. 802.11b-1999, High-speed physical layer extension in the 2.4-GHz band.

[6] Thomson, J. et al., An integrated 802.11a baseband and MAC processor, *Proc. ISSCC 2002,* San Francisco, CA, pp. 451–453.

[7] Eberle, W. et al., 80-Mb/s QPSK and 72-Mb/s 64-QAM flexible and scalable digital OFDM transceiver ASICs for wireless local area network in the 5-Ghz band, *IEEE J. Solid-State Circuits,* Vol. 36, pp. 1829–1838, 2001.

[8] Tell, E., Nilsson, A. (J), and Liu, D., A vector processor for converged baseband DSP, to be submitted to *WASP 2004,* Stockholm, Sweden.

[9] Huang, Y. and Chiueh, T., A sub-word parallel digital signal processor for wireless communication systems, *Proc. Asian-Pacific Conf. on ASIC 2002,* Taipei, pp. 287–290.
[10] Tell, E., Seger, O., and Liu, D., Operand masking for low-power baseband DSP firmware, submitted to IEEE NORCHIP, 2004.
[11] Gunn, J.E., Barron, K.S., and Ruczczyk, W., A low-power DSP core-based software radio architecture, *IEEE, J. Selected Areas in Communications,* Vol. 17, pp. 574–590, April 1999.
[12] Tell, E. and Liu, D., A converged hardware solution for FFT, DCT, and Walsh transform, *Proc. ISSPA 2003,* Paris, France, July 2003.
[13] Heiskala, H. and Terry, J.T., *OFDM Wireless LANs: A Theoretical and Practical Guide,* Sams Publishing, Indianapolis, Indiana, 2002.
[14] Tell, E. and Liu, D., A hardware architecture for a multi-standard block interleaver, *Proc. ICCSC 2004,* Moscow, Russia, July 2004.
[15] Abnous, A., Low-power domain specific processor for digital signal processing, Ph.D. dissertation, University of California-Berkeley, 2001.
[16] Giacalone, J., Application-specific instruction-set architecture extensions for DSPs, in *The Application of Programmable DSPs in Mobile Communications,* Gatherer, A. and Luslander, E., Eds., John Wiley & Sons, New York, chap. 18, pp. 361–377.
[17] Catthoor, F. et al., *Custom Memory Management Methodology,* Kluwer Academic Publishers, Dordrecht, 1998.

7

Stand-By Power Reduction for SRAM Memories

Stefan Cserveny
Jean-Marc Masgonty
Christian Piguet
CSEM

7.1 Introduction

In processor-based systems on chip (SoCs), the memories limit most of the time the speed and are the main part of the power consumption. Much work has been done to improve their performances [1], however, new approaches are required to take into account the trend in scaled down deep submicron technologies toward an increased contribution of the static consumption in the total power consumption [2]. The main reason for this increase is the reduction of the transistor threshold voltages [3–7].

In a previous article [8], proposals were made for low-power SRAM and ROM memories working in a large range of supply voltages. With all bit-lines kept precharged at the supply value, however, the proposed techniques are not favorable for the static leakage. Simple solutions are available for a ROM: it can be switched off in the standby mode, or only the selected bit-lines are precharged before a read (with the corresponding speed penalty). For the six-transistor SRAM cell, however, enough supply voltage has to be present all the time to keep the stored information.

Negative body biasing increases the NMOS transistor threshold voltage and, therefore, reduces the main leakage component — the cutoff transistor subthreshold current. A positive source-body bias has the same effect and can be applied to the devices that are processed without a separate well, however, it reduces the available voltage swing and degrades the noise margin of the SRAM cell. Another important feature to be considered is the speed reduction resulting from the increased threshold voltage, which can be very severe when a lower than nominal supply voltage is considered.

This chapter presents an approach based on the source-body biasing method for the reduction of the subthreshold leakage, with the aim of limiting the normally associated speed and noise margin degradation by switching it locally. At the same time, this bias is limited at a value guaranteeing enough noise margins for the stored data.

Only the case of a SRAM is considered here, however, the same approach can be applied to any blocks containing storage circuits (e.g., flip-flops and registers).

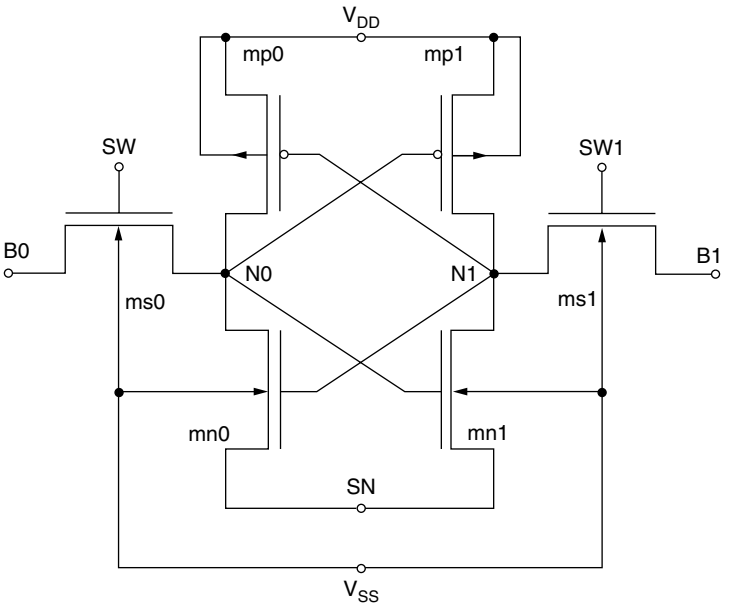


FIGURE 7.1 SRAM cell with separate NMOS source connection SN.

7.2 Leakage Reduction

In the deep submicron processes, as they scale down, there is a tendency toward a static leakage strongly dominated by the subthreshold current of the turned-off metal-oxide semiconductor (MOS) transistors. In even deeper nanometric technologies, an important tunneling gate leakage current exists [9]; however, it can be neglected in the preceding 100-nm technologies. Because this subthreshold leakage is much higher for the NMOS than that of the PMOS, only the NMOS transistors will be considered here for the leakage reduction (if necessary, however, the proposed methods can be applied as well for both). Notice also that the leaky NMOSs usually have no separate well in a standard digital process.

To allow source-body biasing in the six-transistor SRAM cell, the common source of the cross-coupled inverter NMOS (SN in Figure 7.1) is not connected to the body. Body pick-ups can be provided in each cell or for a group of cells, and they are connected to the V_{SS} ground.

In Figure 7.1, the possibility for separate select gate signals SW and SW1 has been illustrated, as for the asymmetrical cell described in Masgonty et al. [8] and Cserveny et al. [10], in which read is performed only on the bit-line B0 when only SW goes high while both are activated for write. Even if a symmetrical cell, which is selected for read and write with the same select word signal SW ≡ SW1, can also be considered for the leakage reduction techniques to be proposed, the asymmetrical six-transistor SRAM cell [10], presents several advantages.

With the asymmetrical cell, the dynamic power consumption at read is reduced because only one of the two bit-lines has a voltage swing. This consumption can be reduced even more by physically splitting these single bit-lines into subbit-lines as illustrated in the Figure 7.2. Because the subbit-lines are connected to a fraction of the cells in the column, the capacitive load to be discharged is significantly reduced.

With the asymmetrical cell, the functional voltage supply range is large, particularly for voltages well below the nominal process values because read on only one bit-line is performed without sense amplifiers, while both bit-lines are used for write. Not only are the problems related to sense amplifiers eliminated, even the power consumption compares favorably despite the full swing discharge of the subbit-lines where a "1" is read. Because only one select transistor is activated at read, the width (W) and length (L)

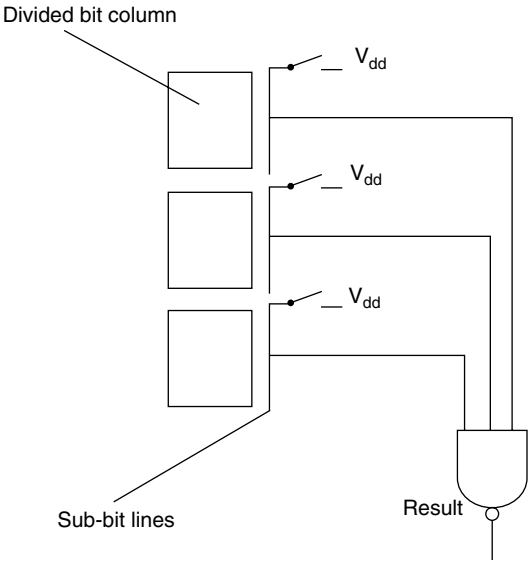


FIGURE 7.2 Physically split bit-lines principle: The subbit-lines are connected directly to a NAND gate, contrary to the speed reducing series transistor used when subbit-lines are connected to a main bit-line by a switch. In this example, three subbit-lines are considered.

of the transistors can be optimized differently on the two sides of the asymmetrical cell by taking advantage of the relaxed read noise margin constraint and the asymmetrical need for high driving capability.

A positive bias between the SN node and the V_{SS} ground (V_{SN}) will reduce the subthreshold leakage of a nonselected cell (SW and SW1 at V_{SS}) as plotted in Figure 7.3.

The result in Figure 7.3 is for a minimum size symmetrical cell in a 0.18-μm process (mn0, mn1, mp0, and mp1 have W = 0.22 μm and L = 0.18 μm only ms0 and ms1 are longer) for the state in which the

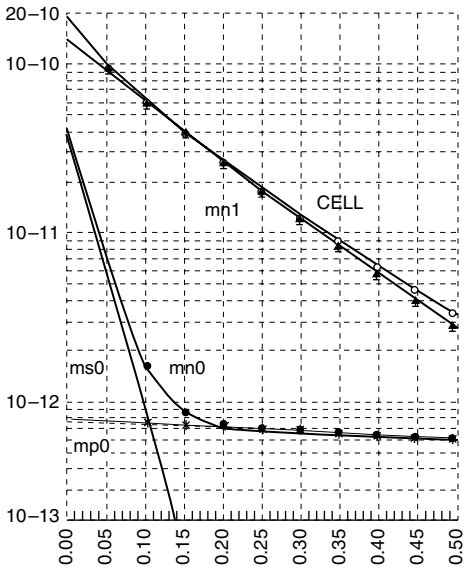


FIGURE 7.3 The simulated cell current (logarithmic scale in A) and its decomposition into the contributions of its transistors as a function of the source-body bias (linear scale in V).

node N0 is low (at V_{SN}) and the node N1 is high (at V_{DD}). In this case the cell leakage is dominated by the transistor mn1, the NMOS which is cut off with $V_{GS} = 0$; its leakage is controlled by the body effect, the source-body voltage V_{BS} effect on its threshold voltage. The other branch contribution due to mp0, a less leaky PMOS, and to ms0, a longer NMOS with a gate-source voltage V_{GS} that becomes negative, is much less.

Furthermore, the ratio between the on (at VD = VG) and off (at $V_{GS} = 0$) currents in these deep submicron processes is worst at the minimum channel length, normally used in the digital designs; therefore, a somewhat longer transistor will be considered whenever possible and effective, without too much area penalty. According to the preceding analysis, such longer transistors are interesting mainly for the two inverter NMOS (mn0 and mn1), which, in the asymmetrical cell can be optimized differently also for low leakage.

The proposed method becomes less effective in the much deeper nanometric technologies; however, because there is less body bias effect to control the threshold voltage and the leakage due to the tunnelling gate, current and the gate induced drain leakage become increasingly important.

7.3 Noise Margin and Speed Requirements

A fixed bias V_{SN} on the source SN can reduce the subthreshold leakage of a nonselected cell as far as the remaining supply is enough to keep the flip-flop state, including the necessary noise margin.

For the selected cell in the active read/write mode with such a fixed V_{SN} bias, a speed loss is associated with the reduced available driving current and the noise margin at read is modified. This speed loss can be partially compensated adapting the V_{SN} value to the process corner because the corner that is worst for leakage is best for speed and vice versa. Nevertheless, this approach is hard to control over a large range of supply voltages because the speed/leakage relationship is a strong function of the supply voltage and temperature. Moreover, as depicted in the simulations of Figure 7.4, the noise margin at read is reduced at low supply voltages, which is a common situation for applications that need a supply value that is low or in a large range, such as low-power portable systems.

The noise margin, represented by the maximum size square that can be nested into the cross-coupled voltage transfer characteristics [11,12], is visibly reduced by the source-body bias $V_{DD} = 0.9$ V, a supply value well below the nominal 1.8 V for this process. Notice also that the crossing of the transfer characteristics changes from 3 to 5 points.

For the nonselected cell, important for the standby leakage, only the inverter transfer characteristics (without select MOS loading) come into account, and even if the V_{SN} increase reduces the noise margin,

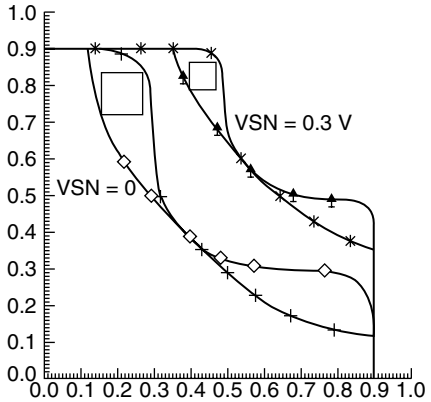


FIGURE 7.4 Worst-case (Nfast Pslow 125°C) read access static noise margin analysis for the minimum size symmetrical cell at $V_{SN} = 0$ and $V_{SN} = 0.3$ V for $V_{DD} = 0.9$ V.

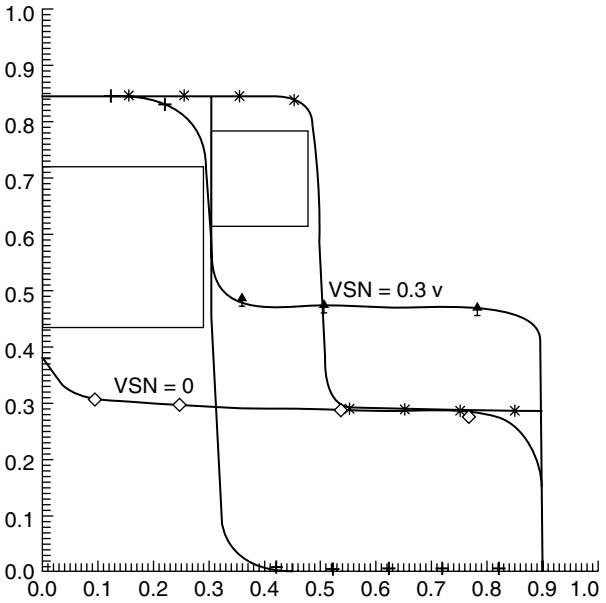


FIGURE 7.5 Worst-case (Nfast Pslow 125°C) standby static noise margin analysis for the minimum size symmetrical cell at $V_{SN} = 0$ and $V_{SN} = 0.3$ V for $V_{DD} = 0.9$ V.

it remains reasonably high if the V_{SN} does not increase too much, as plotted in Figure 7.5 for the same voltages discussed previously.

7.4 Locally Switched Source-Body Bias

The V_{SN} bias, useful for static leakage reduction, is acceptable in standby if its value does not exceed the limit at which the noise margin of the stored information becomes too small, but it degrades the speed and the noise margin at read. Therefore, it will be interesting to switch it off in the active read mode; however, the relatively high capacitance associated to this SN node, about six to eight times larger than the bit-line charge of the same cell, is a challenge for such a switching.

It is proposed here to partition the cell array into a number of groups, with the inverter NMOS sources of all cells inside a group being connected to a common terminal SN belonging to that group, and to locally assign a switch between these SN terminals and the ground, as illustrated in the Figure 7.6. When an active read or write operation takes place, the switch assigned to the group containing the selected word, connects the SN terminal of this group to ground. Therefore, in the active mode the performance of the cell is that of a cell without source bias. In standby, however, or if the group does not contain the selected word, the switch is open. With the switch open, the SN node potential increases reducing the leakage of the cells in that group, as described before, until the leakage of all cells in the group equals that of the open switch, which is slowly increasing with the SN potential (V_{DS} effect). Nevertheless, to guarantee enough noise margins for the stored state, the SN node potential should not become too high; this is avoided with a limiter associated to this node.

The group size and the switch design are optimized, compromising the equilibrium between the leakages of the cells in the group and the switch with the voltage drop in the activated switch and the selected group SN node switching power loss. The switch is an NMOS that has to be strong enough compared with the read current of one word, the selected word (i.e., strong compared with the driving capability of the cells defined by the select and inverter NMOS in series). At the same time, the NMOS switch has to be weak enough to leak without source-body bias as little as the desired leakage for all words in the group with source-body bias at the acceptable V_{SN} potential. On the other hand, to limit

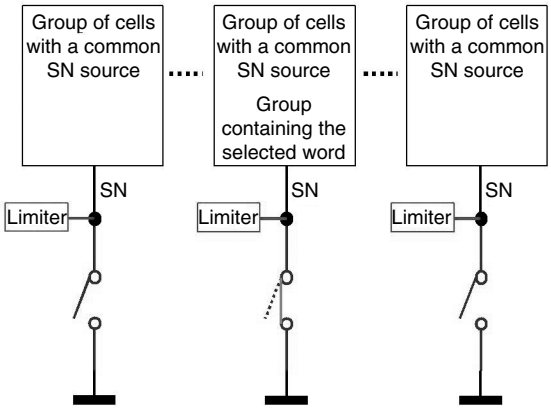


FIGURE 7.6 The limited and locally switched source-body biasing principle applied to a SRAM.

the total capacitive load on the SN node at a value keeping the power loss for switching this node much less than the functional dynamic power consumption, the number of words in the group cannot be increased too much. In particular, this last requirement demonstrates why the local switching is needed contrary to a global SN switching for the whole memory.

Figure 7.7 illustrates the most flexible approach to implement the described principle. Groups are built with a specific number of rows in a one-word column, the selected group of rows SRG containing the selected row SR. The switch size, built with one transistor for each bit, is adapted to the number of bits in a word. As proposed in Masgonty et al. [8], the divided word line blocks have the size of one word and the select column signal SC is activated at read and write for the column containing the selected word SW in the selected row SR.

In this implementation, with the number of rows in a group an integer power of 2 (e.g., 16 or 32), the selection of the group of rows (i.e., the SRG signal) is made by a simple partial row decoding. By combining this SRG signal with the one-word column selection signal SC, the group activation signal SG is generated. This SG signal closes the switch for that group (i.e., the NMOS transistors with the gate at SG). The limitation is obtained by connecting the SN node of the group to a VSN source with a NMOS

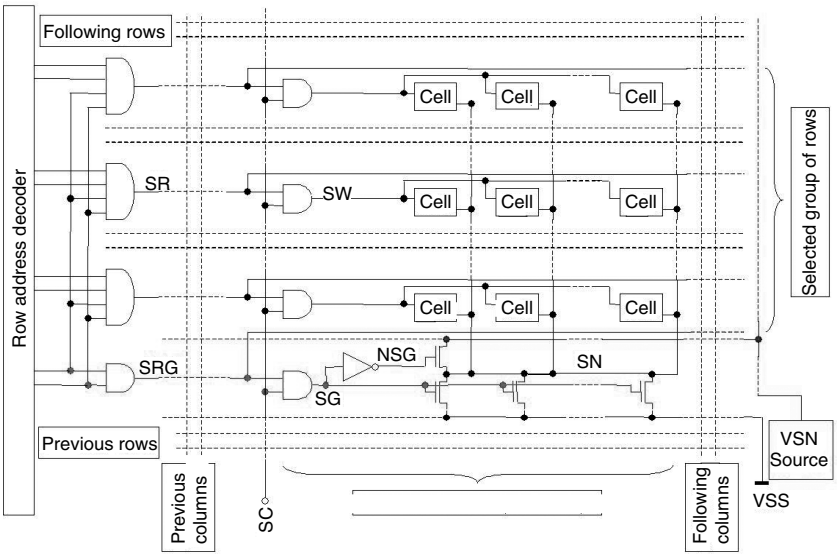


FIGURE 7.7 Example of a SRAM organization implementing the locally switched and limited source-body biasing.

pass transistor activated by the complement of the SG signal, thus connecting all groups except the activated one. Contrary to the fixed bias approach in which the whole active current goes through the biasing source, here the VSN source has to deliver or absorb only a very small current (i.e., the difference between the small leakages of the cells and the switches that are essentially equilibrated by design).

Other implementations were considered depending on the size and specific design constraints of the memory. It is quite interesting to activate the group only by address decoding, reducing the number of groups switching at the expense of a larger leakage in the group that remains selected; in particular, a group containing only all the words in a row can be activated directly by the select row signal SR. It is important to notice that the SN switching power loss depends on the group size and its relevance to the total power dissipation depends on the number of groups in the memory. Even more important is to notice that the limiter should only limit the SN potential increase. Figure 7.7 presents only one of the different implementations that have been considered to fulfill this function, however the VSN source can be replaced by an active or passive limiter circuit. Diode limiters, as considered in Enomoto et al. [13] for the typical case, are very interesting; however, they should be carefully designed and checked over all parameter corners.

7.5 Results

A cell implementing the described leakage reduction techniques together with all the characteristics of the asymmetrical cell described in Section 7.2 has been made in a 0.18-µm process. The SN node can be connected vertically or horizontally, and body pickups are provided in each cell for best source bias noise control. The inverter NMOS mn0 and mn1 use a 0.28-µm transistor length. Despite larger W/L of ms0 and mn0 on the read side used to take advantage of the asymmetrical cell for higher speed and relaxed noise margin constraint on their ratio, a further two to three times leakage reduction has been obtained besides the source-body bias effect plotted in Figure 7.3. The largest reduction is obtained where it is most desired — in the fast-fast process corner, the worst case for the leakage. The cell is a square with 2.8-µm sides — five times the upper level metal pitch allowing easier access blocks layout. It is 68% larger than the minimum size cell designed at the foundry with the special layout rules not accessible for other designs; however, the new cell is only 25% larger than a cell using the minimum size cell transistors if both layouts use the same standard rules. Put in another way, this further two to three times leakage reduction is equivalent with a reduction of 0.1 V to 0.15 V of the source-body bias needed for same leakage values. Overall, with V_{SN} near 0.3 V, the leakage of the cell has been reduced at least 25 times, however, more than 40 times for the important fast-fast worst case.

Figure 7.8 compares the read bit-line discharge delay for the group switched source-body bias with the fixed bias approach.

As expected, the equivalent V_{SN} bias for the same delay, depicted by the crossings in Figure 7.8, corresponds to the voltage drop in the switch. For the worst case considered here (all bits in the read word at one, that is, maximum current in the switch), the voltages at these crossings vary between 19 mV for SS (worst case for delay) and 28 mV for FF (best delay).

When the local group V_{SN} switching is compared with a SRAM using the same cell without source-body bias ($V_{SN} = 0$), the approximate 10% maximum speed loss, as well as the meaningless noise margin reduction due to these few mV, are not important.

On the other hand, the Figure 7.8 results illustrate the important speed improvement obtained with the local group switching as compared to the fixed V_{SN} bias; for example, at 0.3 V, the delays are 68 times shorter for the SS corner, the worst case for speed, and still 7.4 times shorter for the FF corner, the worst case for leakage, illustrating a very important speed increase. At the same time, the important reduction of the noise margin at read at low V_{DD} due to such a fixed V_{SN} bias (see Figure 7.3) is avoided.

The only area increase, compared with the previous design [8], is due to the group switches. If the switches are implemented as in Figure 7.7, the switches for two groups need an area about that of one row. Therefore, the area penalty for 16-word groups is about 3%. If the groups are organized in the rows, the area increase is even less.

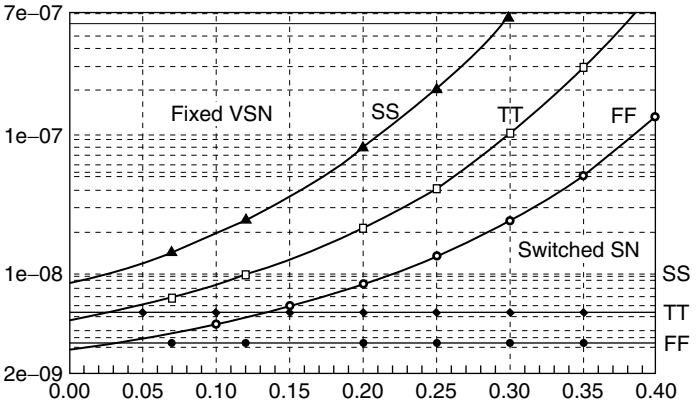


FIGURE 7.8 Simulated bit-line dicharge delay (in s) as a function of the V_{SN} source-body bias (in V) for the new cell and group switch design compared with the fixed V_{SN} bias result for the same cell at the slow-slow (SS), typical (TT), and fast-fast (FF) process corners, considering 256 rows, 25°C and all bits in the word at 1.

Following the strong leakage reduction in the cell array, it was important to design the remaining blocks of the memory to keep the total standby current of the memory still dominated by the cells. The standby leakage of the other blocks has been limited at well under 10% of the total standby total current of the memory by adapting some control signals for better low leakage of the blocks. For the most leaky elements, identified according to their functions, similar approaches have been used, such as source biasing, but without state retention requirements or longer than minimum length transistors. This did not require any extra area. The switches added for the bit-line pull-down transistors could be placed in the very large input signal buffers area already defined by the length of the select row and the height of the read/write blocks.

With the proposed locally switched source-body biasing applied to a SRAM, when compared with a SRAM without source-body bias, an important reduction of the standby leakage is obtained without any significant degradation of its speed or noise margin. Compared with the recently available SRAM with fixed source-body bias, for the same leakage reduction, the proposed SRAM improves significantly the speed and the noise margin at read.

7.6 Conclusion

Standby power reduction for storage circuits, which have to retain data, is obtained through limited locally switched source-body biasing. The standby leakage current is reduced by using a source-body bias not exceeding the value that guarantees safe data retention and less leaking nonminimum length transistors. This bias is short-circuited in active mode to improve the speed and the noise margin, especially for low supply voltages; however, this is made for a fraction of the circuit containing the activated part, allowing a trade-off between switching power and leakage. For a SRAM in a 0.18-μm process, the leakage is reduced more than 25 times without speed or noise margin loss.

References

[1] K. Itoh, Low-voltage memories for power-aware systems, Keynote Speech at *ISLPED '02,* Monterey, CA, August 12–14, 2002.

[2] H. Morimura et al., A shared-bitline SRAM cell architecture for 1-V ultra low-power word-bit configurable macrocells, *Proc. ISLPED 1999,* San Diego, CA.

[3] S. Kosonocky et al., Enhanced multithreshold (MTCMOS) circuits using variable well bias, *Proc. ISLPED '01,* pp. 165–169.

[4] S. Narendra et al., Scaling of stack effect and its applications for leakage reduction, *Proc. ISLPED '01,* pp. 195–200.
[5] C. Kim and K. Roy, Dynamic VT SRAM: a leakage tolerant cache memory for low-voltage microprocessors, *Proc. ISLPED '02,* pp. 251–254.
[6] N. Azizi et al., Low-leakage asymmetric-cell SRAM, *Proc. ISLPED '02,* pp. 48–51.
[7] C. Piguet et al., Techniques de circuits et méthodes de conception pour réduire la consommation statique dans les technologies profondément submicroniques. Invited paper, *Proc. FTFC '03,* pp. 21–29, Paris, May 15–16, 2003.
[8] J.-M. Masgonty, S. Cserveny, and C. Piguet, Low-power SRAM and ROM memories, *Proc. PATMOS 2001,* paper 7.4.
[9] F. Hamzaoglu and M. Stan, Circuit-level techniques to control gate leakage for sub-100-nm CMOS, *Proc. ISLPED '02,* pp. 60–63.
[10] S. Cserveny, J.-M. Masgonty, C. Piguet, and F. Robin, Random access memory, U.S. Patent 6,366,504 B1, April 2, 2002.
[11] E. Seevinck, F.J. List, and J. Lohstroh, Static-noise margin analysis of MOS SRAM cells, *IEEE J. Solid-State Circuits,* vol. 22, pp. 748–754, Oct. 1987.
[12] A.J. Bhavnagarwala, X. Tang, and J.D. Meindl, The impact of intrinsic device fluctuations on CMOS SRAM cell stability, vol. 36, pp. 658–665, April 2001.
[13] T. Enomoto, Y. Oka, H. Shikano, and T. Harada, A self-controllable-voltage-level (SVL) circuit for low-power high-speed CMOS circuits, *Proc. ESSCIRC 2002,* pp. 411–414.

8

Low-Power Cache Design

Vasily G. Moshnyaga
Koji Inoue
Fukuoka University

8.1 Introduction

Cache memories are the most area- and energy-consuming units in today's microprocessors. As the speed disparity between processor and external memory increases, designers try to put large multilevel caches on a chip to reduce the number of external memory accesses and thus boost the system performance. (See Table 8.1 for a survey of the on-die caches for several recent high-end microprocessors.) On-chip data and instruction caches are implemented using arrays of densely packed static RAM cells. The device count for the caches often exceeds the number of transistors devoted to the processor's datapath and controller. For example, the Alpha21364 [3] and PA-RISC Maco [5] microprocessors have over 90% of their transistors in RAM, with most of them dedicated for caches; the Itanium2 [1] has 80% in caches, the IBM G5 [7] has 72%, the PowerPC [8] has 71%, and Strong-ARM110 [9] has 70%. Due to the large load capacitance and high access rate, these caches account for significant portion of the overall power dissipation (e.g., 35% in Itanium2 [1]; 43% in Strong-ARM [9]). Therefore optimizing caches for power is increasingly important. Although much work on energy reduction has taken place in the circuit and technology domains [10,11], interest in cache design for power efficiency at the architectural level continues to increase. Architecture is the entry point in cache design hierarchy, and decisions taken at this level can drastically affect the efficiency of design.

This chapter describes architectural techniques appropriate for reducing power and energy in caches. Because power does not incorporate any notion of completing a given task, we could minimize the power by simply turning cache off or running it slowly. Our focus here is on techniques appropriate for reducing energy. Cache energy reduction is more difficult because it must be achieved under very strict timing requirements (i.e., without affecting performance). The chapter is organized as follows. First, we describe conventional cache design, and define sources of energy dissipation and degrees of freedom in the low-power design space. Then, we present an in-depth survey (and in many cases analyses) of cache energy-reduction techniques. We conclude by summarizing the major low-power design challenges that lie ahead.

TABLE 8.1 Survey of Recent High-Performance Microprocessors

Micro-Processor [reference]	Freq. (GHz)	Tech. (μm)	Die size (mm^1)	Trans. Count ($\times 10^6$)	Power (W)	Cache Organization							
						Cache type	Size (KB)	Assoc. (ways)	Line Size (byte)	Latency (cycles)	Write Update	Bandwidth (GBs)	Transistor Count (% of Total)
Itanium-2 McKinley [1]	1.0	0.18	421	221	130	L1-D	8	4	64	1	WT	16	
						L1-I	12op	4	n/a	1	n/a	32	
						L2	256	8	128	5i/6f	WB	32	80
						L3	3000	8	1024	12	WB	32	
Pentium 4	1.5	0.18	55	217	54	L1-D	12*	4	64	2i/6f	WT	44.8	
						L1-I	256	8	n/a	n/a	n/a	n/a	n/a
						L2	3000	8	128	7i/7f	WB	44.8	
Alpha 21364 [3]	1.0	0.18	397	100	150	L1-D	64	2	64	1	WT	19.2	
						L1-I	64	2	64	5i/6f	WT	19.2	93 (in fRAM)
						L2	1750	6	n/a	12	WB	16	
Ultra Sparc III [4]	1.0	0.18	244	23	80	L1-D	32	4	64	2i/6f	WT	16	
						L1-I	64	4	n/a	n/a	WT16	52	
						L1-pw	4	4	n/a	n/a	n/a	n/a	n/a
						L2-tags	88	n/a	n/a	n/a	n/a	n/a	
PA-RIS C Maco [5]	1.0	0.18	366	325	n/a	L1-D	1500	4	64	1	WT	n/a	
						L1-I	1500	4	64	1	WT	n/a	92 (in fRAM)
						L2-tags	1000	n/a	n/a	n/a	n/a	n/a	
Power4 (2 cores) [6]	13.0	0.18	400	n/a	125	L1-D	2*32	1	128	1	WT	416	
						L1-I	2*64	2	128	1	WT	416	n/a
						L2	1500	8	512	n/a	WB	n/a	
						L3-tags	176est.	8	n/a	n/a	WB	n/a	

Note: L1-D and L1-I denote level-1 data cache and level-1 instruction cache, respectively; L2 and L3 are level-2 and level-3 caches; pw is the prefetch cache; victim is the victim cache; WT is write-through; WB is write-back; 5i/6f defines latency of integer access and floating point access, respectively; n/a means not available.

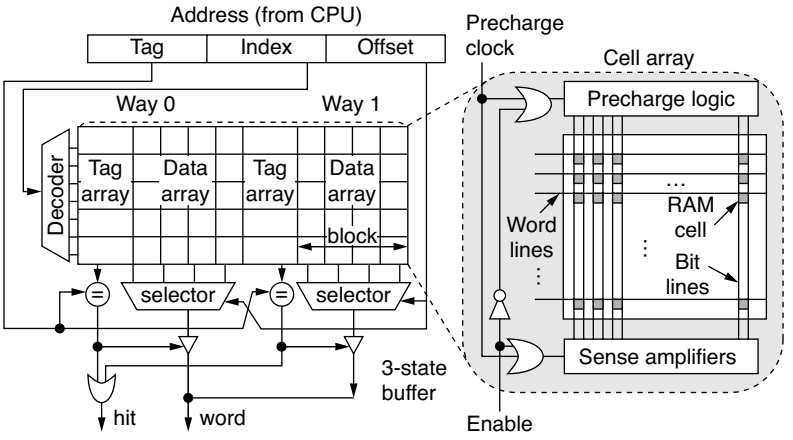


FIGURE 8.1 Organization of a two-way, set-associative cache.

8.2 Cache Organization

The dominant cache organization employed in modern microprocessors is *m*-way set-associative cache. Figure 8.1 exemplifies the organization for $m = 2$. A cache line stores a set of *m* memory blocks (several words each). Unlike traditional static RAM (SRAM), each data block in the cache is tagged with extra bits that indicate which address of main memory is actually being stored in the line, as well as other bits that indicate the validity and status of cached copy. A pair of tag array and data array is called way each consisting of S rows, or sets. If cache has only one way (i.e., $m = 1$), it is called direct-mapped; if the number of ways equals the number of blocks in the cache, the cache is called fully associative.

An *m*-way set-associative cache works as follows. Whenever processor initiates a memory access, it sends the physical address of the target word to the cache. The index bits of the address indicate cache line, while the least significant bits (or block offset) indicate position within the block. To read a datum, the cache activates in parallel all bits in the selected line, simultaneously comparing the upper portion of the address with *m* tags in the set. If a match occurs and data is valid, the data word from the way that hits is supplied to the processor. Otherwise, a cache-miss occurs, and the cache passes the address to a low-level memory to read the datum and replace it with a block within the cache. When the datum arrives, both the processor and the cache receive a copy. The cache then stores its copy with the appropriate address tag. On a write access, if the address being written to the memory has its copy in the cache, the cache updates value of the copy. Otherwise, the copy may be brought into the cache and then updated (write-allocate), or it may be updated in memory and not brought into the cache (write-no allocate).

The cache may employ two policies in handling a write to a block that is present in the cache: write-through, which updates both the cache and memory upon each write, and write-back, which writes the cache only. With write-through, read misses never produce copy-back writes to memory. With write-back, data are written to memory only when the data is removed from the cache (on cache miss), and an update has occurred. The cache checks the status bit and writes-back on miss only dirty (i.e., modified) blocks.

An *m*-way set-associative cache has as many as *m* alternative locations to write a new block on miss and, therefore, requires a specific policy (e.g., least recently used) for block replacement. A direct-mapped cache does not need such a policy because it has only one location from which to choose. If a program happens to repeatedly access words from two different blocks that map to the same line, however, the direct-mapped cache will continually swap the blocks, spending power and time. In comparison to the other alternatives (of the same cache size), direct-mapped cache is the fastest and the smallest, but has the highest miss rate. Set-associativity elevates the hit rate on expense of both the access time and hardware cost. For a fixed cache size, an *m*-way set-associative cache has as *m*-times as long word-lines and *m* times

as more circuits for tag comparison and data selection. Therefore, caches with associativity $m > 32$ usually employ content-addressable memory (CAM) to store the tags and search them in parallel.

8.3 Factors Influencing Energy Consumption in Caches

There are two major components of energy dissipation in cache: dynamic energy, which is attributed to signal transitions in activated bit- and word-lines, sense-amplifiers, comparators, and selectors during reads/writes, and static energy, which is due to the total amount of leakage (or subthreshold) current, I_L, through inactive or OFF-transistors. The dynamic energy consumed per access is a sum of the energy spent on searching within the cache, an extra energy required for handling the writes and energy consumed by block replacement on cache miss. In CMOS circuits, energy consumption is proportional to the switching capacitance and the square of supply voltage, V. Thus, if we make N accesses to a CMOS cache that has miss rate of α, we will dissipate the following amount of energy:

$$\text{Energy} = N^*(C^*V^2 + \beta^*k^*E_m + \alpha^*2^*E_m) + V^*I_L \quad (8.1)$$

Here, C is the cache capacitance switched per access; β is the ratio of cache-writes to the total number of cache accesses, k is the write-policy dependent coefficient, and E_m is the energy required by one low-level memory access. Because β and E_m are independent to cache organization, optimizing cache for energy entails an attempt to minimize N, C, V, I_L, k, and α. This section briefly discusses these factors describing their relative importance, as well as the interactions that complicate the energy optimization process.

8.3.1 Miss Rate

Each external access requires many clock cycles and at least by two orders of magnitude more power than an on-chip access. Miss rate multiplies the number of external accesses by a factor of two and, therefore, has the highest optimization priority. Three parameters have an impact upon the cache miss rate: cache size, block size, and associativity:

1. Cache size. As cache size increases, the cache miss rate drops. A 64-KB cache, for example, has 6 times less misses than 1-KB cache for the 32-B block size and 10 times less misses for the 64-B block size [12]. Large on-chip caches reduce the overall energy consumption but cause a linear extension of bit-lines and more energy loss in the lines. This energy loss eventually may dominate other sources and result in energy increase, not savings. The cache energy, which is first lowered as cache size increases, starts to grow up as cache exceeds 32-KB in size [12]. In addition, caches larger than a few tens of KB are difficult to access in one cycle. If latency is not so important, the multi-cycle access provides a good opportunity to save energy by selectively enabling only relevant ways and sets in array.
2. Block size. With increase of block size, miss rate decreases due to enhanced spatial locality. As block grows in size, however, the width of data read to and from the cache also grows, leading to more energy dissipation in bit-lines, sense-amplifiers, and most important, in inter-memory traffic. Eventually, this energy cost may outweigh the energy savings of a smaller miss ratio because some instructions or data brought in on a miss will not be used. Usually, processors use smaller block sizes for low-latency and low-bandwidth L1 caches, and large blocks for high-latency and high-bandwidth L2 and L3 caches.
3. Set associativity. Miss rate decreases with more degrees of associativity though this effect diminishes with increasing cache size [13]. A four-way 64-KB set-associative cache (32-B block size) has a 14 and 44% miss rate advantage over a two-way set-associative cache and direct-mapped cache, respectively [14]; however, high associativity increases the cache access time. Although the energy consumption of each bit-line in an m-set associative cache is usually decreasing as associativity increases, the number of bit-transitions in cache usually grows with associativity. In addition, an *m*-way set-associative cache increases sense-amps power by *m* times. Meanwhile, sense-amps

consume less energy than the bit-lines in conventional caches. Therefore, associativity affects energy less than the cache size.

8.3.2 Write Policy

The write-through policy is simpler to implement than the write-back but requires more external accesses. With write-through, we access external memory the same number of times as we write to cache. With write-back, external memory is accessed only when block is replaced. Thus, we can simplify Equation 8.1 by using $k = 1$ for write-through and $k = \alpha$ for write-back. Because $\alpha << 1$, the write-back policy generally leads to better performance and energy-efficiency; however, it may cause temporal data inconsistency (i.e., when external memory and the cache associate different data with the same physical address). Problems may also arise when several processors with independent caches are sharing the same low-level memory. Regarding to write policy, there is also a choice whether to use write-allocate or write-no allocate on a cache-miss. Write-allocate appears to be a better choice for power because subsequent writes (or reads) to the allocated block may be done directly in the cache.

8.3.3 Cache Accesses Rate

High access rate (N) directly increases the number of tag checks and data reads, magnifying the cache energy consumption proportionally. If cache is not large enough to capture the majority of code and data used in relatively long execution period, the average amount of memory transfers will also grow up elevating both the fetching time and total energy dissipation. A common policy to reduce cache accesses is to use separate L1 data and instruction caches and increase memory hierarchy. An extra level in cache hierarchy introduces an extra delay and, therefore, is used only when latency increase is acceptable.

8.3.4 Switching Capacitance Per Access

In conventional caches, bit-lines, word-lines, and sense-amplifiers are the major contributors to the switching capacitance per access. In an *m*-set associative SRAM cache of size S, block size of L bytes, and tag size of T bits, the number of bit-lines is proportional to $2*(m*8*L + T)$ and the number of word-lines to $S/(m*L)$. Thus, we have: $C \propto 2*(m*8*L + T)*S/(m*L)$. This provides us with four options to lower switching capacitance: decreasing the set-associativity, m, reducing the block size, L, reducing the cache size, S, and shrinking the tag bit-width, T. Note that all these options, except T, inversely affect the miss rate and, therefore, are viable only when cache switching activity dominates energy consumption.

8.3.5 Voltage

Because supply voltage (V) has a square impact on energy, voltage scaling offers the most effective means to minimize energy dissipation. Unfortunately, we pay a performance penalty for voltage reduction with delays drastically increasing as V approaches the threshold voltage Vt of the devices. Recent advances in technology have scaled both V and Vt aggressively, while maintaining a single clock access latency to L1 caches; however, reducing Vt increases leakage current, I_L, and makes the leakage term in Equation (8.1) appreciable. Another important concern for low V-low Vt regime is the fluctuation in Vt. As V approaches 1 V, a Vt variation of ±0.15 V causes delay changes by a factor of three. Such a large variation in nominal delay values cannot be tolerated. This sets a major limitation on how V can be reduced unless the Vt fluctuation is diminished to the level of ±0.05 V [15]. Thus, lowering the threshold has a limited option for countering the effect of reducing V.

8.3.6 Leakage

Reducing energy consumption in CMOS circuits by lowering the supply voltage increases leakage energy dissipation. Leakage current I_L is independent of the circuit activity but dependent on the device area and temperature. Therefore, it occurs as long as power is supplied to the CMOS device. The on-chip L1

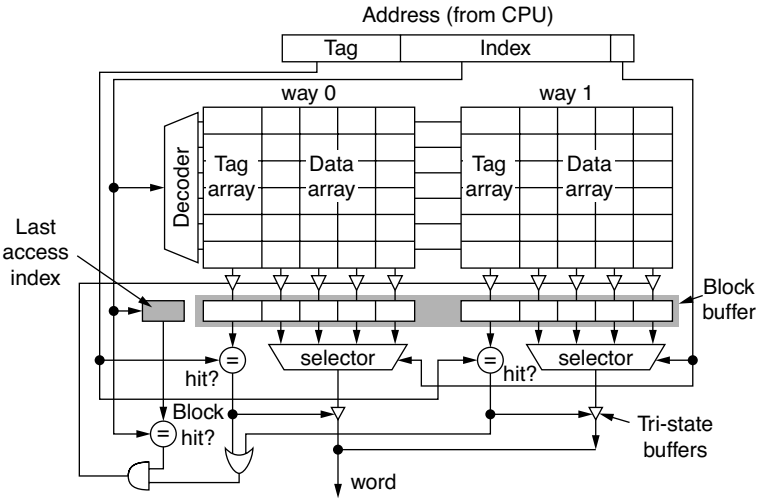


FIGURE 8.2 Block-buffered cache.

and L2 caches are one of the main candidates for leakage reduction because they utilize a significant fraction of chip transistors, most of which are inactive during long periods. At 0.13-μm technology, leakage energy already accounts for 30% of L1 cache energy and as much as 80% of L2 energy [16]. As technology moves below 0.1 μm, leakage energy increases exponentially, dominating the total energy used by the CPU [17]. Therefore, issues to reduce leakage without affecting performance become very important for the future low-energy cache design.

8.4 Energy Reduction Techniques

8.4.1 Reducing Cache Access Rate

When accessing a cache it is very likely that a requested word is confined to the block or one of the blocks in the set, that was last accessed. Thus, we can avoid some references to the cache by placing the most recently fetched block into a buffer and then reading-out the block data directly from the buffer without activating cache [18,19]. Figure 8.2 illustrates a structure of block-buffered cache. The cache checks first whether the line address of the current access is currently resident in the buffer. If there is a block hit, then the cache data is read from the block buffer without accessing the cache arrays. The normal access, including bit-line precharging and row-access decoding is performed only when a block-buffer miss occurs. Although effectiveness of this method strongly depends on the spatial locality of the access pattern and the block size, it can save 40 to 50% of energy over nonbuffered cache [19].

The main disadvantage of the block buffering is cache-access latency increase. To overcome this problem, Kamble and Ghose [20] proposed performing cache precharging and block-buffer access in parallel. Compared with the original block buffering [19], it spends some energy in precharging bit-lines on a block-buffer hit but does not affect cache latency.

A further extension to the block buffering is to use multiple (e.g., four) line buffers [21] or even a small direct-mapped level-0 (L0) cache, placed between processor and L1 cache [22,23]. This "filter" cache helps to omit many of the data accesses, so the L1 cache is not referenced as frequently. If data is found in the filter cache, energy is saved. If data is not found, however, an extra cycle is needed to access larger L1 cache. A 256-byte filter cache backed by a 32-KB L1 cache saves energy by 65% on average but causes a 29% performance degradation in comparison to a conventional cache due to access time increase on a filter cache miss.

Efficiency of using the L0 cache can be improved in two ways: by increasing data locality or by dynamic access management. The first approach puts in L0 cache only the most frequently executed instruction

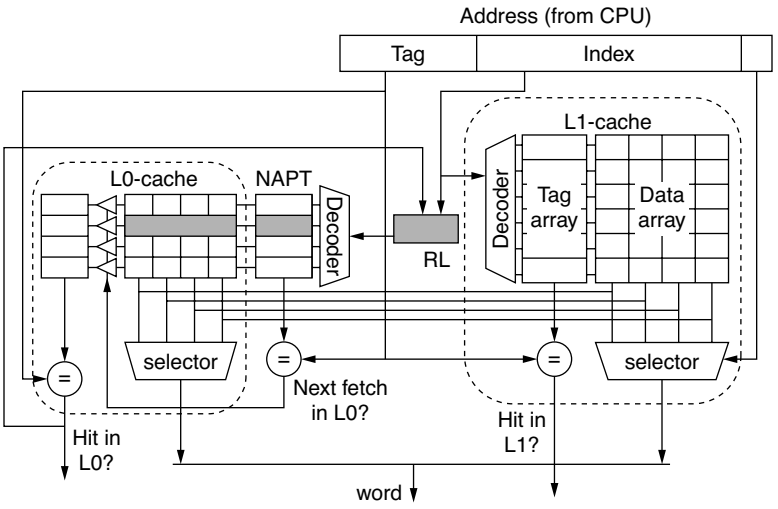


FIGURE 8.3 L0-cache bypassing hardware.

blocks or loops, identifying them based on branch prediction unit [24], loop instructions [26], or specific instructions inserted in the application code [25]. The second approach dynamically predicts the L0-cache miss and, if the prediction is correct, it accesses the L1-cache directly to reduce the miss penalty [27]. In small loops, consecutive addresses differ by a few least significant bits. Consequently, if current access hits (misses) in the L0 cache, the remaining accesses to the same block are likely to hit (miss) in the L0 cache. As illustrated in Figure 8.3, the approach adds to the caches a next address prediction table (NAPT), a register (RL) to store the index of the most recently accessed line in L0, and a comparator. The NAPT has the same number of entries as L0-cache, but stores only four lowest bits of the next address tag. Both L0-cache and the NAPT are read in parallel. If current access hits L0-cache and the NAPT entry matches the current address tag, then the next access is directed to L0 cache, and RL is updated. Otherwise, L1 cache is accessed, and the L0-cache is refilled.

8.4.2 Reducing Switching Capacitance Per Access

As discussed in Section 8.3, several degrees of freedom are inherent in cache organization: cache size, associativity, block size, and tag size. Reducing switching capacitance in cache can be achieved along different directions that exploit one or more of these freedoms. Most techniques utilize the same basic idea: divide the cache arrays into pieces, and then selectively activate on cache access only one that can hold the data. The partitioning can be determined structurally or behaviorally. The structural partitioning affects cache structure, while keeping the cache-access operation unchanged. In opposite, the behavioral partitioning modifies the cache-access sequence but leaves the cache structure unchanged.

8.4.2.1 Structural Partitioning

8.4.2.1.1 Word-Line Segmentation

Word-line segmentation is similar to column multiplexing used in SRAM memories [28]. Cells in each row are grouped into blocks, and a local word line, as depicted in Figure 8.4, accesses memory cells in each block. The local word-lines *LWLj* are connected to a global word *WLs* line through a transfer gate. Therefore, only cells in the activated block have their bit-line pair driven. With k local lines, the word line capacitance is reduced by $1/k$.

In caches, this method is usually applied by splitting the data array into subbanks and using the low-order bits (block offset) of the address to disable subbanks that are not accessed. Figure 8.4 illustrates a two-way set-associative cache divided in subbanks. Because only one subbank is active per access, the

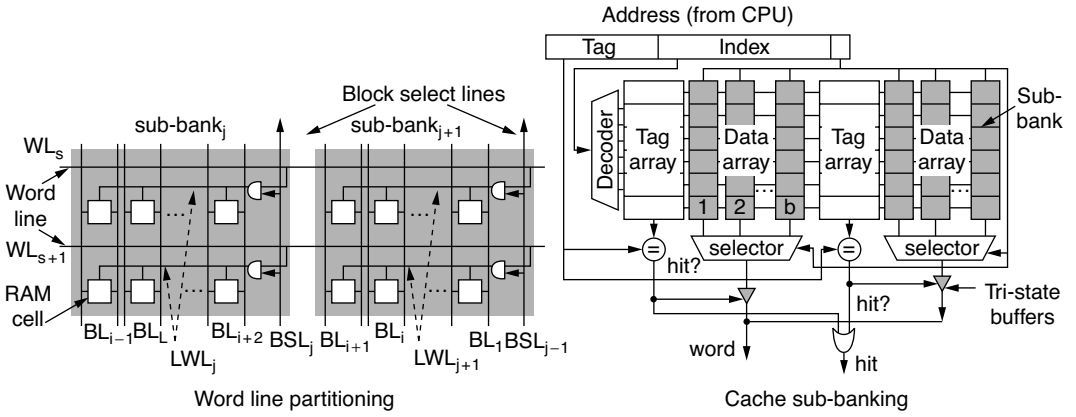


FIGURE 8.4 An illustration of word-line partitioning and cache subbanking.

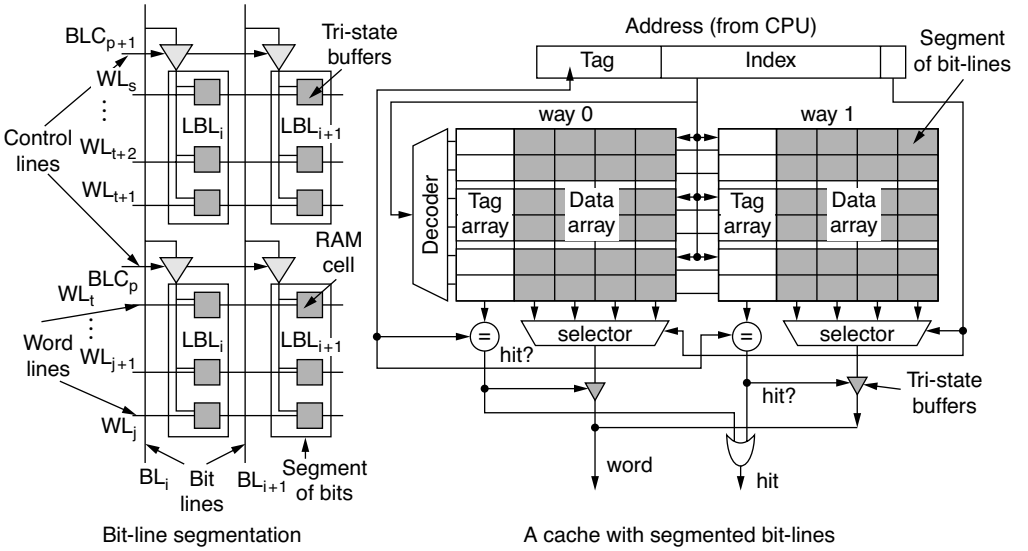


FIGURE 8.5 An illustration of bit-line segmentation.

more subbanks in the cache, the better. For example, a 32-KB cache with a block size of 2, 4, 8, and 16 words per block saves energy by 46%, 63%, 80%, and 89%, respectively [19].

8.4.2.1.2 Bit-Line Segmentation

The idea of this method [28] is to split every column of bit-cells into independent segments, as depicted in Figure 8.5. An additional pair of local bit-lines (LBL) runs across the segments. The lines are connected to the global bit-lines (BL) through three-state buffers, which are activated/deactivated by the bit-line control lines (BLC). Before a read access, all segments are connected to the common lines, which are precharged as usual. The cache address decoder identifies the segment targeted by the row address issued to the array and isolates all but the targeted segment from the common bit-line. This reduces the effective capacitive loading on the common line and eventually lowers the energy.

8.4.2.1.3 Bit-Line Isolation

Along with the cache division into subarrays, holding each bit-line and output driver of each subarray in a constant state of precharge when the array is not being accessed can reduce the switching activity. The lines remain in a precharge state unless a subarray access is required. In the architecture, column

switches are placed on bit-lines between memory cells and the sense amplifiers to isolate parasitic capacitor from of bit-lines from the active sense amplifiers [29]. The latch-type sense amplifiers engage after the column switches turn off. Thus, the sense amplifier load is reduced, and the switching capacitance decreases.

8.4.2.1.4 Multiple Cache Decomposition

The key feature of this technique [30] is to split the entire cache into independently addressable, small modules, each of which is an actual cache with its own cell array and peripheral circuits. A cache generally can be divided into *M* identical independently selectable modules, with each module further divided into K subbanks. The subarrays may be created on any appropriate access boundary: byte, word, double-word, and so forth. The latency of this multi-divided cache architecture is equivalent to that of the single module and energy consumption is only 1/(M*K) of the regular, nondivided cache. The smaller the cache size, the bigger savings. A typical example is the SA-110 [9], which divides its 32-way associative 16-KB L1-I and L1-D-caches into 16 fully associative sub-arrays, so that only one eighth of the cache is enabled per access. With a 160-MHz target clock frequency, the SA-110 designers were able to maintain single cycle cache latency with this degree of associativity.

8.4.2.1.5 Cache Decomposition by Data Types

Locality of cache accesses increases when subcaches are organized by data types. For example, a cache might have three modules: one for stack data only, another for global data, and the third for other data types [31]. The modules may be different in size (e.g., 4 KB, 4 KB, and 32 KB, respectively) and activated separately. When only the stack-module is referenced, the overall energy dissipated in cache can be reduced by 70% in comparison with a conventional cache.

8.4.2.2 Behavioral Partitioning

Current L1 set-associative caches are designed to operate in a single clock cycle. To ensure single-cycle operation, the caches probe all the data ways in parallel with the tag lookup, although the output only of a single matching way is used. The energy spent accessing the other ways is wasted. To save energy, designers make the way select in tag array to precede search in the data array [29]. Although this ordering can be efficiently used in multi-cycled L2 caches, it slows the access time of L1 caches by almost 60% [32]. Next, we present techniques that try to balance the speed-energy characteristics of L1 cache by dynamically optimizing the cache-access.

8.4.2.2.1 Phased Array Activation

The key idea of this method [33] is to drive the tag array and the data array by two different clocks: *C1* and *C2* (see Figure 8.6). If clock frequency is high, the difference between C1 and C2 is small. In this case, both ways in the data array start the access before the tag array dispatches the way select signal. Once the way select signals are produced in the tag array, unselected ways of the data array stop their access. If the clock frequency goes lower, the difference between C1 and C2 becomes long enough to allow generation of the way select signals to complete before the rising edge of C2. Thus, the only one way of data array is activated, resulting in low-energy operation. Although this so called "*automatic-power save architecture*" reduces the fastest access speed by a quarter, a low area overhead and large energy savings (almost 60%) make it very promising to adjust energy consumption to workload variation. The approach has been adopted in Hitachi's SH3 and Super-H microprocessors.

8.4.2.2.2 Way Prediction

Way prediction was originally proposed to reduce cache access time [34]. The idea is to predict the way in which data can be found prior to the cache access and probe the way instead of waiting on the tag array to provide the way number. Because only the predicted way is accessed, the method also offers immediate benefits for energy savings. A way-predicting scheme [35] speculatively chooses the most-recently used way to access, while disabling the nonaccessed ways. When prediction is correct, the cache completes operation in one cycle (see Figure 8.7), consuming single-way energy only. Otherwise, it searches the other ways in parallel and spends two clock cycles without any energy reduction. The way-

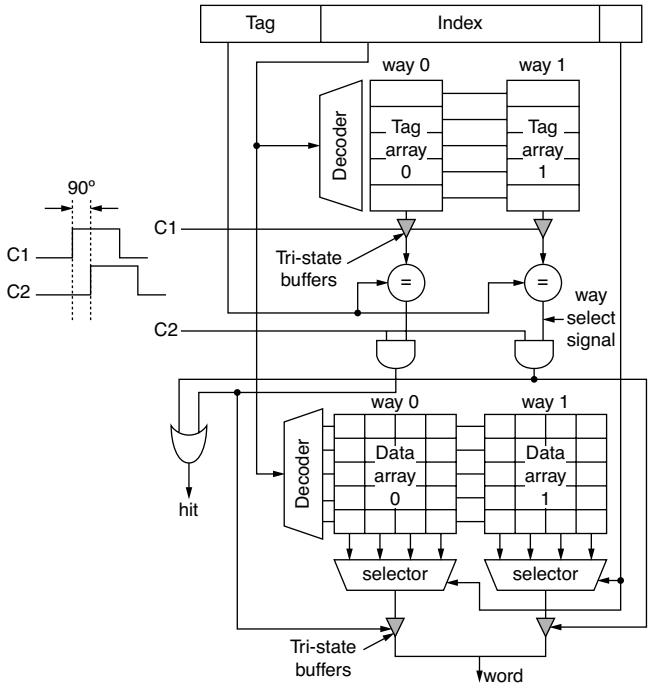


FIGURE 8.6 Automatic power-save cache architecture.

prediction improves the energy-delay product of cache by 60 to 70%; however, it incurs large performance loss on each misprediction.

Enforcing way prediction with other techniques can save cache energy without compromising performance. One approach is to unite way prediction with selective direct mapping [36]. Because 70 to 80% of blocks accessed in L1 D-cache are nonconflicting, allocating these blocks to direct-mapped positions can avoid both way prediction and tag checks. To identify conflicting blocks, the reactive cache [37] uses a list of recently replaced blocks, or victims. On a replacement, if the evicted block is present

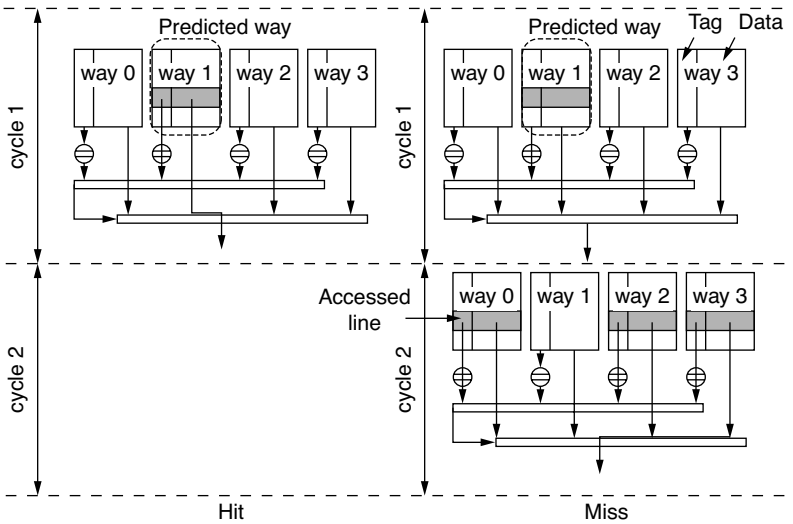


FIGURE 8.7 An illustration of a way-predicting scheme.

in the victim list, its entry's counter in the list is incremented; otherwise, a new victim list entry is created. Those blocks whose counter exceeds two are conflicting and placed in the set-associated position to avoid future conflicts. On access, the cache predicts whether the access is direct or set-associative. If prediction is correct, the cache probes only the matching data way, while a misprediction initiates a second probe of the correct data way. If a hit occurs in the direct-mapped way, the counter associated with the block is decremented. If a set-associative way hits, the counter is incremented. In comparison with original way prediction, the approach achieves the same energy-delay reduction but at less than 3% performance overhead.

Another approach [38] combines way prediction with phased-cache access based on a pseudo-associative cache. The cache utilizes a steering table and two schemes of phased-cache access to predict which way to probe first. The first scheme probes only a predicted way and, if the probe fails, it activates all the ways at the second attempt. The second scheme activates all the tag arrays and data array of the predicted way. If the prediction fails, the data array of correct way is subsequently accessed. A similar idea is advocated in [39]. In this proposal, prediction logic operates in parallel with the translation look-aside buffer (TLB) lookup. Based on the prediction, the cache controller activates appropriate subcache by sending the physical address. If the selected subcache hits, the access ends. Otherwise, the controller looks for the next subcache to probe. The reprobing continues until the data is found or it is determined not to be on-chip subcaches. The architecture reduces the number of external memory accesses, improving both the system energy and delay by 42 and 23%, respectively.

8.4.2.2.3 Selective Cache Ways

This method [40] is one of the first attempts to adapt cache associativity to workload variation as between applications as well as during execution of an individual application. The idea is to disable some cache ways for applications that do not require full cache associativity, while enabling them for applications in which high performance is necessary. Figure 8.8 is a two-way set-associative cache with selective ways. In addition to the standard cache modules, the structure includes a cache controller, gating hardware for disabling operation of particular ways, and a software visible cache way select register (CWSR). The CWSR is modified by the operating system (OS) and contains k bits, each of which signals the cache controller to enable/disable a particular way. A zero value of the CWSR bit nulls the corresponding set-

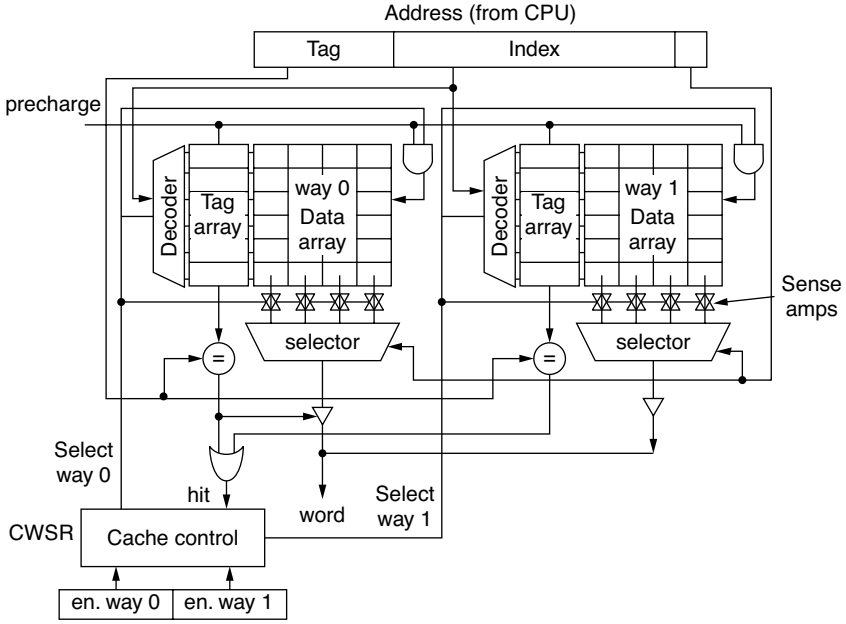


FIGURE 8.8 A two-way, set-associative cache using selective ways.

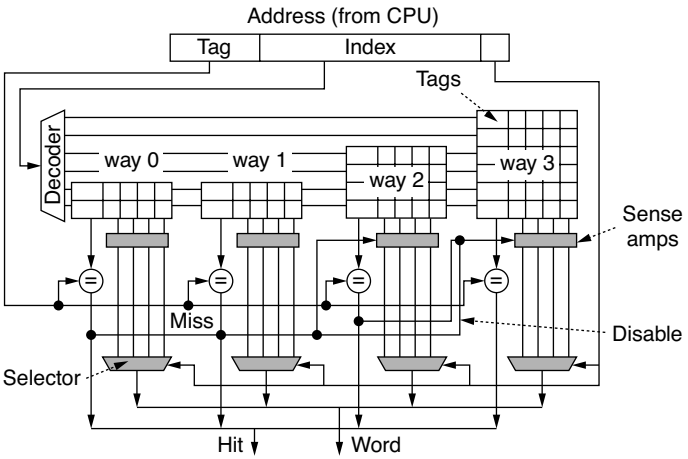


FIGURE 8.9 An asymmetric cache.

way signal thereby preventing a data way from precharging, bit selection, and firing the sense-amps. The OS monitors the performance degradation threshold and changes the number of ways enabled in the cache. Due to adaptive associativity, the cache energy can be reduced by 25 to 40% on average for a 64-KB, four-way set-associative cache with less than 2% loss in performance, and 10 to 25% for a 32-KB four-way cache. What it needs is specific software support for analyzing application requirements and modifying the CWSR. In addition, it requires an extra support for sharing data in disabled ways and maintaining its proper coherency state.

In conventional caches, some sets experience a higher access load and many more misses than others. Although most of the accesses tolerate quite a low associativity, frequently referenced addresses in programs require high associativity. Intuitively, we can reduce miss penalty by assigning more ways to the frequently used sets than others do (i.e., by making the cache asymmetric) [41]. Figure 8.9 gives an example. Extra associativity is shared by having two cache blocks from the large ways align with individual cache blocks of the smaller ways. In the asymmetric cache, the smaller ways are faster and consume less energy. A hit in a smaller way disables the sense amplifiers in all larger (or slower) ways. Because about half of the energy per access in dissipated on the data sense-amps, early hits on faster ways increase energy efficiency. Compared with the conventional cache, an asymmetric cache can save up to 17% with random replacement policy and 13% with least recently used (LRU) replacement policy. The hit latency is determined by the slowest way that produces a hit, plus overhead to route the data after the hit is detected.

8.4.2.2.4 Selective Cache Sets

Alternatively to the preceding approach, which alters set-associativity, this method [16,42] explores the second dimension in cache design, namely the number of cache sets, changing it in response to varying application demand for the cache size. Figure 8.10 is a two-way set-associative cache with dynamically resizable sets. The controller monitors cache operation in fixed-length intervals counting the number of misses. At the end of each interval, it increments/decrements the number of sets, depending on whether the miss count is lower/higher than a predefined threshold. The minimum number of sets achievable is a single sub-array per cache way. Because changing the number of sets alters both the required index and the tag bits, the cache includes a set-mask to indicate the number of index bits used in cache. Every time the cache is downsized, the mask shifts to the right, to enable fewer index bits (but more tag bits) and vice versa. The downsizing disables the high-order sets in the cache in groups of power of two. Because the cache maintains as many tag bits as required by the smallest number of sets, the tag array is larger than in the normal cache.

The method is orthogonal to selective cache ways and, therefore, can be combined into hybrid cache architecture to reduce energy dissipation. According to Yang [16], the method has better energy-delay

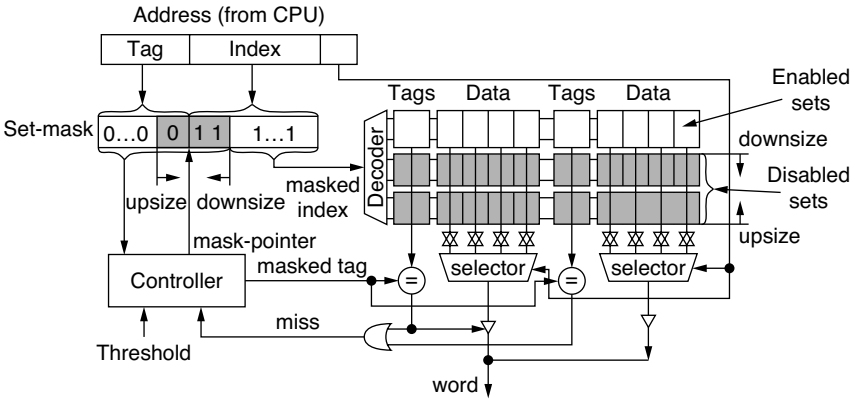


FIGURE 8.10 An organization of a two-way, set-associative cache with selective sets.

than the selective-ways when set-associativity is no more than four. At the same time, selective cache-ways is more beneficial for applications with small variation in cache size requirement. The hybrid cache has the best energy-efficiency but larger area.

8.4.2.2.5 Selective Line Sizing

Line (or block) length is the third dimension of cache optimization. The optimal line size, which results in minimal miss rate, varies with applications as well as within the application [43,44]. Most processors use only one cache line size and, therefore, often consume redundant energy. MIPS R3000 [45] supports multiple cache line sizes, but line size is configurable at boot time. It will be ideal if a cache line size can vary with application.

The line size can be controlled by software. If the compiler knows that the application only needs one word, only one word is fetched, and only that word and tag is activated in the cache. A cache can employ two fixed line sizes, changing them at runtime based on locality of data, as is done in Tang et al. [46]. The long "fetch size" is used for applications with good spatial locality and short fetch size for applications with good temporal locality. The line size is predicted for an interval of time by profiling the miss rate over a number of fetch sizes and selecting one with minimal miss rate. The dynamic profiling requires additional hardware and may affect performance on misprediction. If the prediction interval is much longer than the profiling time, however, the approach can benefit caches, especially those with large miss latencies.

Another example is the span cache [47]. In the RAM-based span cache line, each word can store both data and tag. To indicate whether the word is a tag or a datum an extra bit (t-bit) is attached to each word. The tags divide the set into spans of potentially different length. The length of line is determined by finding the next (t-bit) or the end of the set. A large overhead incurred by range-check and word-selection makes this cache quite challenging to build.

8.4.2.2.6 Reducing Switching Activity of Tag Checks

Conventional caches perform tag comparison on every access to detect whether the requested datum is within the cache. The tag operations associated with tag check are typically designed for a worst-case scenario and, therefore, utilize the entire effective address. In high associativity caches, the tag length approaches the length of the entire effective address, which results in energy expensive tag reads and comparisons. A k-way set-associative cache does k tag-checks in parallel even if only one way contains the requested data. If the same block is sequentially accessed, all the tag checks, except the first one, become unnecessary because all words within the same block have a common tag.

To avoid these redundant tag-checks, Panwar and Rennels [25] proposed testing whether the target instruction resides in the same cache block as the previous instruction. Although program counter can control this condition, it requires tagging each branch instruction to indicate whether the branch control

is transferred outside the current line. A variant of this technique is to store the address index of the last line accessed within each bank and enable cache tags only if a different line is being accessed. In addition, a possible alternative is to memorize the tag of the last cache line that was accessed and compare it against the tag of the next memory access before enabling the tag search [48]. The main disadvantage of these solutions is that they introduce a wide compare operation into the critical path of every cache access, enlarging the delay of this latency-sensitive path.

Several techniques have been proposed to reduce tag-checks in I-cache without sacrificing performance. One is way-memorization [49]. Similar to way-prediction, the technique stores way information (links) within the cache but in addition maintains one "valid bit" per link to indicate that the link is correct. If the valid bit is set, the cache follows the link to locate the next instruction, thereby avoiding a tag check. Otherwise, it falls back to a regular tag comparison to find the target instruction and update the link. The following instruction fetches reuse the valid link. The approach is orthogonal to other methods and can be applied regardless of cache associativity. Compared with way-prediction, the way memorization can save an extra 13% of energy but requires a large area due to keeping the links inside the cache.

Another technique is a history-based tag-comparison [50]. The idea is to omit tag check if the target instruction has been already referenced in the past and no cache misses have occurred since that reference. To validate the condition, the cache records the history of accesses through execution footprints placed in an extended branch target buffer (BTB). An execution footprint indicates whether the target instruction block associated with the branch currently resides in the I-cache. A footprint is recorded when the corresponding block is referenced and erased whenever a cache miss occurs. On access, the cache checks a particular footprint and, if it is valid, avoids the tag check. Otherwise, it invalidates all the previously stored footprints, while setting a corresponding footprint associated with the newly referenced block. The technique reduces the cache energy by 17% with almost no impact on performance.

A compiler-driven tag check reduction for I-caches is reported in Witchel et al. [51]. Because the compiler often knows when the program is accessing the same piece of memory, the number of tag comparisons can be significantly reduced if the software guides hardware. The proposed directly addressed (DA) cache is augmented with a number of DA-registers to indicate exact locations of cache lines in the data array as well as status bits. The DA-registers are amenable by software, using extended versions of load, store, and jump operations. A load instruction, for example, causes a proper DA-register to record the location of the referenced line, as depicted in Figure 8.11. A tag-unchecked load or store instruction analyzes the DA-register and if it is valid, the cache line pointed to by the DA-register is accessed without any tag-check, and the word specified by the line offset is transferred to processor. Otherwise, hardware

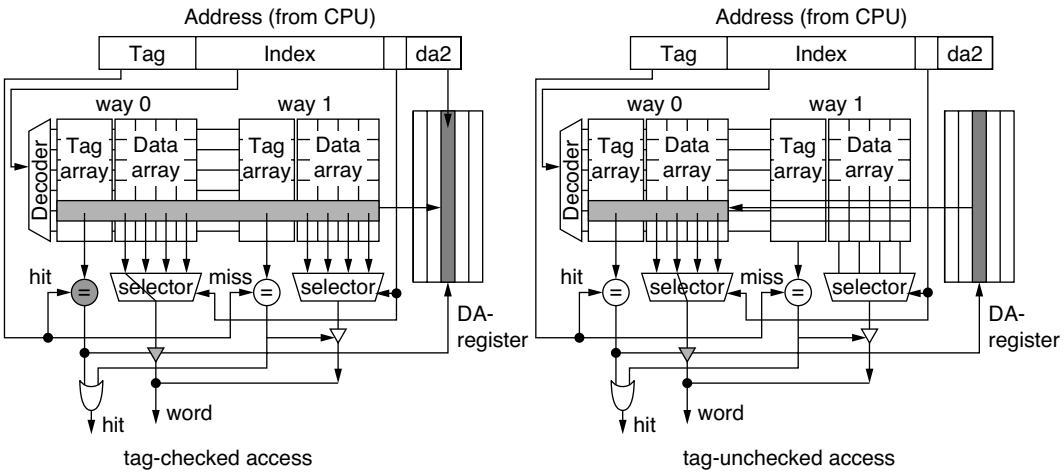


FIGURE 8.11 A direct-addressed cache.

performs the tag search and sets the DA-register. The technique saves energy by up to 40%; however, it might affect code compatibility due to the specific compilation scheme.

Another option in lowering the tag activity is to reduce the number of tag bits, which are read from the cache and used in the comparison. If referred locations are close in the address space, a small number of least significant bits can be enough to distinguish all the possible conflicting addresses in the line. The number of required tag bits can be determined for each loop by compiler, which inserts instructions to disable appropriate tag bits at the beginning of the loop and to enable the bits at the end of the loop, respectively [52]. During operation, this number is loaded to a control register which directly enables/disables particular bit-lines of the cache tag and gates the tag comparator cells. The method also can be applied for data caches if the compiler places the data sets within a region covered by one tag, or spanning more than one tag region by overlapping some of the data arrays [53]. The only problem is code compatibility.

8.4.2.2.7 Data Compression

Bit-line switching is the main source of energy consumption in cache memories. Subbanking, segmented bit-lines, and hierarchical bit-lines decrease bit-line capacitance switched on each access by exploiting the address locality. Interestingly, data values in caches also exhibit high locality, being asymmetrically distributed across the bit-lines. Leveraging this uneven data distribution can increase energy savings.

Dynamic zero compression [54] is based on observation that over 70% of the cache values are zeros; so compressing these zeros can avoid redundant bit-line charging/discharging. The scheme adds an extra bit to each cache byte to indicate whether the byte contains all zeros. On a read access, the scheme prevents bit-line discharge by disabling the local word-line for each byte when the zero-indicator bit (ZIB) is set. If the ZIB is not set, the eight bits are read normally. On a write access, only the ZIB is written if byte is zero. Otherwise, both the data bits and the ZIB are written. The approach reduces the data cache energy by 26% and instruction cache energy by 10% under 9% area overhead and 7% clock penalty. It, however, is applicable to zero patterns only.

Besides zeros, just a few distinct values occupy the majority of cache locations [55]. These values are scattered somewhat uniformly across cache and remain almost the same during program run. We can exploit this phenomenon by dividing the data array into two parts, as depicted in Figure 8.12 [55]. The frequent values are stored in encoded form (2 to 7 bits) in the low-bit data array, while nonfrequent values are stored in unencoded form (32 bits) in both data arrays. A flag bit is attached to each word in the low-bit data array to indicate whether the word is encoded or not. On read access, the cache reads

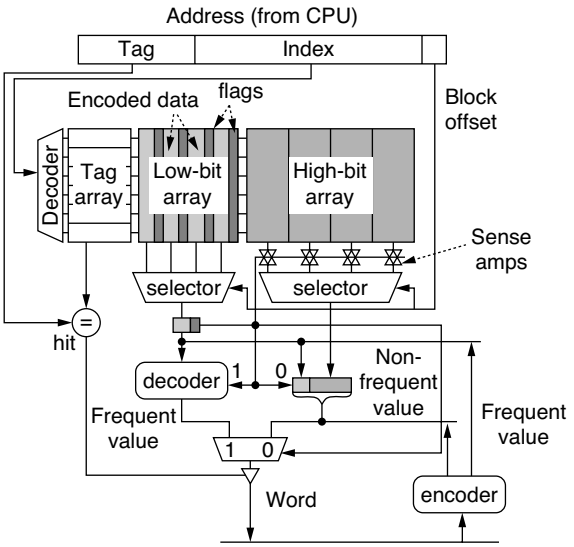


FIGURE 8.12 The frequent-value cache.

out the low-bit data array; if the flag points to an encoded word, then it decodes the word and completes the access without activating the high-bit array. Otherwise, it does not perform decoding but takes another clock cycle to read the remainder of the word from the second data array. The full value in this case is obtained by concatenating the two parts of the word. On write access, the incoming value is checked within the CAM-based encoder; if it is frequent, it is stored in cache in the encoded form jointly with the flag. Otherwise, the flag is stored with unencoded, original word. Although up to 32% energy saving has been reported for the eight-way 64-KB cache, the main challenge exists in implementation of the CAM-based data coding-decoding (codec) capable of detecting frequent values with low impact on performance and area.

Obviously, the compression in caches is neither limited to these two schemes nor to data arrays. According to Mahapatra et al. [56], existing compression techniques can reduce access times for L1 cache by 1/3, area by 2/3, and power consumption by 1/2.

8.4.3 Voltage Reduction

A widely used technique to limit the voltage swing in RAM is the word-line pulsing [57]. Pulses, short enough to read or write the cell arrays, activate the word-lines. The pulse level does not swing fully between V_{dd} and ground level: it is restricted to narrow range, and it is wide enough for correct sense-amplifier operation. This technique is effective in caches regardless of operating frequency.

Due to critical dependence of processor performance on cache operation, reducing voltage in caches without increasing the delay is very difficult. If cache is block-buffered, however, the supply voltage to buffer can be reduced. In block-buffered cache (Figure 8.2), the buffer and the arrays are operated by a single voltage that must be high enough to charge (discharge) the cache circuits in the clock-time interval. Because the circuit capacitance operated on the block-hit is less than that accessed on the block-miss, the block-buffered cache has an idle time on each block-hit. This idle time can be traded with voltage if the block buffer is operated by a dual voltage supply [58]. At the first phase of clock cycle 1, the cache precharges the memory arrays for read, decodes the address, and compares the address tag with the block-buffer tag. At the next phase, if there is a block-hit, the cache disables the arrays from the read port and enforces the low voltage. This voltage slows down the word selection from the block, filling the idle time with action. In opposition, when the block-miss is detected, the high voltage is selected to accelerate the operation. In this case, the cache reads out the selected block into the array output latches. In the next clock cycle, it performs tag comparison I (phase 1) and, if there is a hit, drives the matched block from the data array, copying it concurrently to the buffer and outputting the requested data word from the block (phase 2). The method maintains the performance of a single-voltage block-buffered scheme, while saving its energy consumption by up to 36% for programs with high data correlation.

8.4.4 Leakage Energy Reduction

Many techniques are used to reduce leakage energy dissipation at the circuit/technology level (see Chapter 3). Most of these techniques, however, affect circuit performance and, therefore, are not suitable to performance-critical circuits, such as caches [59]. Efforts have been put to devise micro-architectural techniques capable of reducing leakage energy dynamically (i.e., during program run) with less cost and area overhead. Usually, powering off unused SRAM devices reduces leakage energy. To shrink it dynamically we need a policy: when and what to switch off. Current microprocessors use a simple OS-driven policy: deactivate the entire processor when it enters a sleep mode.

A more intelligent solution is to disable cache sets that eventually become inactive during program execution [60]. The proposal combines selective cache sets [16] with a circuit technique, called *gated-V_{dd}*, which inserts an extra transistor in the supply voltage (V_{dd}) or the ground path (V_{ss}) of the cache's SRAM cells; the extra transistor is "turned on" in the active sets and "turn off" (or gated) in inactive sets. In comparison to conventional cache, the proposed dynamically resizable I-cache (or DRI cache) [42] reduces energy-delay by 62% with 4% latency increase.

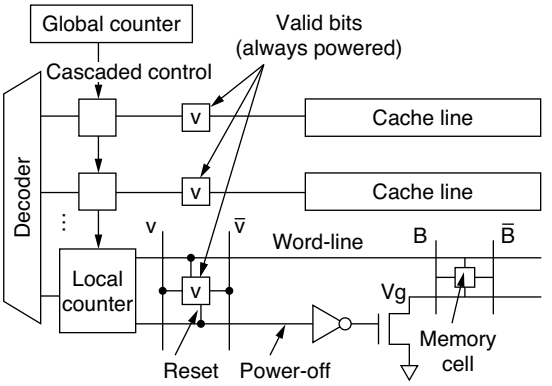


FIGURE 8.13 An illustration of cache-decay scheme.

Cache-line decay [61] exploits a similar idea but at a granularity of cache-line and without resizing. Motivated by the fact that almost half of cache lines remain unused for a long period of time, the technique turns-off the least recently used cache lines assuming that these lines will be not referenced in the future. In design, a counter is attached to each cache line, as depicted in Figure 8.13. The counter is incremented periodically at fixed time intervals (set by the global counter) and cleared on each access to the line. Once a line counter reaches its maximum count, the corresponding cache-line is invalidated, and its voltage supply is turned off via a gated-V_{dd} transistor. The scheme results in roughly 70% reduction in L1 data cache leakage energy.

The main drawback of these approaches is that the state of the cache line is lost when it is turned off. Reloading the line from L2 cache not only hurts performance, but could also diminish energy savings. To avoid these pitfalls, it is necessary to use adaptive algorithms and be conservative about which line is turned off. Another drawback of these schemes is dependence of arbitrary parameters that must be tuned per application to minimize the performance penalty. In the case of DRI cache, the number of active cache sets is increased or decreased based on the miss bound which must not exceed the application's typical miss rate. In case of cache line decay, the control parameter is line decay interval. Tuning such parameters by application profiling may be difficult because the results differ as within as well as across application. To overcome this limitation, Zhou et al. [62] suggests keeping the tag arrays active to dynamically monitor the cache miss rate and adjust the miss rate based on relative factor instead of an arbitrary, absolute value. The proposed adaptive mode control (AMC) examines the ratio of sleep misses to ideal misses, and if it is "too small," reduces the line decay interval to deactivate more lines. If the ratio is "too large," the line decay interval is increased. Otherwise, the interval remains constant. The hardware implementation of the AMC is relatively compact and requires two counters and a small logic to modify the new line-decay interval based on the miss ratio. On average, the method can turn off up to 73% of I-cache lines and 54% of D-cache lines with less than 2% performance loss.

Leakage reduction in caches can also be achieved by putting a cache line into a low-energy "drowsy" mode, as proposed in [63]. To implement drowsy caches, authors add a "drowsy bit" to each cache line and supply two voltages (V_{dd}-low and V_{dd}-high) to the cell array. The lines are put into drowsy mode either periodically (simple policy) or based on access statistics (i.e., only those lines which have not been accessed in a fixed time interval). Whenever cache is accessed, it reads the drowsy bit of corresponding line; if it indicates normal mode, the line is read without losing performance. If it indicates that the line is in drowsy mode, however, the line is woken up in the next cycle and then read. Compared to caches that use gated-V_{dd} technique, drowsy caches preserve line information and switch faster between the power modes. The penalty they pay for being wrong is a single clock cycle to wake up drowsy lines; however, drowsy caches depend on process variation and do not reduce leakage energy of L1-cache as much as the others. Even in drowsy mode, a cache consumes a quarter of the normal mode energy.

Several works exploit the fact that leakage of a block depends on input pattern and internal state [64,65]. To minimize leakage they apply, a combination of input patterns and internal state (so called sleep-vector) is applied to set internal latches into the correct state and turn inputs to the correct polarity. As Heo et al. [66] demonstrates, the leakage current from a bit-line into the memory cell depends on the stored value on that side of the cell. That is, there is no leakage if the bit line is at the same value as that stored in the cell. Therefore, if we assume that there are more zeros than ones in cache, we could reduce the bit-line leakage of inactive subbank by setting the true line to zero, while keeping the complement line precharged. If this assumption is incorrect, however, the "sleep vector" approach leads to additional energy dissipation. Another solution is to use leakage-biased bit-lines [66]. Instead of setting bit-lines of inactive subbanks to a sleep-value, this scheme turns-off both the precharging transistors and sense-amplifiers, letting the bit-lines float to midrail. Because the precharge transistor switches exactly the same number of times as in conventional SRAM, the energy-performance overhead of the scheme is small; the precharge is only delayed until the subbank is accessed, and the wake-up latency is just that of the precharge phase. Using the scheme to deactivate SRAM read paths within I-caches saves over 24% of leakage energy and 22% of total I-cache energy at 0.07-μm processes. The cache, however, must be designed to deactivate blocks for multiple cycles and preferably to give them an early notice when to be reawakened.

8.5 Conclusion

We surveyed techniques that have been proposed to reduce energy consumption in caches. The presented survey is not comprehensive; instead, we have focused on the architectural optimizations for reducing the cache accesses, switching activity, voltage, and leakage. Even though a broad range of techniques have been proposed, many possible research directions remain.

First, most of the techniques presented here, except structural partitioning, have not been implemented. The challenge remains to integrate them into the design in which power plays as large a role as performance. Second, the cache design space is large and involves complicated tradeoffs. Exploring alternative hardware/software schemes to determine the less energy-consuming cache configuration for given performance constraint and voltage level is another big challenge. Third, static cache design cannot respond to variable applications and thus is inefficient from the energy reduction perspective. If we want to keep energy dissipation within the bounds of the future microprocessor generations, we need to move forward self-adjusting and adaptive cache architecture that can quickly and efficiently respond to the application change as well as varying data statistics. Another challenge is to develop architecture-driven techniques capable of performing cache reconfiguration based on application requirements on cache size or IPC. This is promising not only for applications where throughput can be traded for low energy, but also for reducing leakage. Fourth, leakage energy reduction by powering off unneeded portions of the cache induces additional off-chip accesses and increased latency. A definite challenge exists for exploring cache resizing trade-offs to carefully balance leakage energy with other energy components.

8.6 Acknowledgments

The research was supported in part by The Ministry of Education, Technology, Science, Sports, and Culture of Japan, Grant-in-Aid for Creative Basic Research (A) No.14GS0218, Grant-in-Aid for Scientific Research C (2) No.14580399, and Grant-in-Aid for Encouragement of Young Scientists (A) No.14702064.

References

[1] Naffziger, S.D., et al. The implementation of the Itanium2 microprocessor, *IEEE J. Solid-State Circuits,* Vol. 37, 11, 1448, 2002.

[2] Intel, A detailed look inside the Intel® NetBurst™ micro-architecture of the Intel Pentium4 processor, Nov. 2000, http://developer.intel.com/design/pentium4/manuals/248966.htm.

[3] Jain, A., et al. A 1.2-GHz alpha microprocessor with 44,8GB/s chip pin bandwidth, I*EEE Int. Solid-State Circuits Conf., Dig. of Tech. Papers,* 2001, 240.
[4] Heald, R., et al. A third-generation SPARC V9 64-b microprocessor, *IEEE J. Solid-State Circuits,* 37, 11, 1526, 2000.
[5] Johnson, D.J.C., HP's Maco processor, *Microprocessor Forum,* Oct. 2001.
[6] Behling, et al. The Power4 processor introduction and tuning guide, *IBM Redbooks,* Nov. 2001, available from http://www.redbooks.ibm.com.
[7] Northrop, G., et al. 600-MHz G5 S3/390 microprocessor, *1999 IEEE Int. Solid-State Circuits Conf., Dig. of Technical Papers,* 88, 1999.
[8] Alvarez, J., et al. 450-MHz Power PC microprocessor with enhanced instruction set and copper interconnect, *IEEE Int. Solid-State Circuits Conf., Dig. of Tech. Papers,* 96, 1999.
[9] Montanario, J., et al. A 160-MHz 32-b 0.5-W CMOS RISC microprocessor, *IEEE Int. Solid-State Circuits Conf., Dig. of Tech. Papers,* 1996.
[10] Chandrakasan, A. and Brodersen, R., *Low-Power Digital CMOS Design,* Kluwer Academic Publishers, Dordrecht, 1996.
[11] *Power-Aware Design Methodologies,* Rabaey, J. and Pedram, M., Eds., Kluwer Academic Publishers, Dordrecht, 2002.
[12] Shiue, W.-T. and Chakrabarti, C., Memory design and exploration for low-power, embedded systems, *J. VLSI Signal Processing,* 29, 167, 2001.
[13] Gary, S., Low-power microprocessor design, in *Low-Power Design Methodologies,* Rabaey I. and Pedram M., Eds., Kluwer Academic Publishers, Dordrecht, 1996, 255.
[14] Hu, Z., Martonosi, M., and Kaxiras, S., Improving cache power efficiency with asymmetric set-associative cache, *Proc. Workshop on Memory Performance Issues,* held jointly with *ISCA-2001,* Goteborg, Sweden, June 30–July 1, 2001.
[15] Kobayashi, T. and Sakurai, T., Self-adjusting threshold voltage scheme for low-voltage high-speed operation, *Proc. IEEE Custom Int. Circuits Conf.,* 1994, 271.
[16] Yang, S.-H., An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches, *Proc. 8th Int. Symp. on High-Performance Comput. Architecture,* 2001.
[17] Borkar, S., Design challenges of technology scaling, *IEEE Micro.,* 19(4):23, 1999.
[18] Bunda, J., Athas, W., and Fussel, D., Evaluating power implications of CMOS microprocessor design decisions, *Proc. Int. Workshop on Low-Power Design,* 1994, 147.
[19] Su, C.L. and Despain, A.M., Cache design trade-offs for power and performance optimization: a case study, *Proc. Int. Symp. on Low-Power Design,* 1995, 69.
[20] Kamble, M.B. and Ghose, K., Analytical energy dissipation models for low-power caches, *Proc. Int. Symp. on Low-Power Electron. and Design,* 1997, 143.
[21] Ghose, K. and Kamble, M.B., Reducing power in superscalar processor caches using sub-banking, multiple line buffers and bit-segmentation, *Proc. Int. Symp. on Low-Power Design,* 1999, 70.
[22] Kin, J., Gupta, M., and Mangione-Smith, W., The filter cache: an energy-efficient memory structure, *Proc. 30th Annu. IEEE/ACM Int. Symp. on Microarchitecture,* 1997, 184.
[23] Bajwa, R.S., et al. Instruction buffering to reduce power in processors for signal processing, *IEEE Trans. on Very Large-Scale Integration Syst.,* 5(4):417, 1997.
[24] Bellas, N., Hajj, I., and Polychronopoulos, C., Using dynamic cache management techniques to reduce energy in a high-performance processor, *Proc. 1999 Int. Symp. on Low-Power Electron. and Design,* 1999, 64.
[25] Panwar R. and Rennels, D., Reducing the frequency of tag compares for low-power I-cache design, *Proc. 1995 Int. Symp. on Low-Power Electron. and Design,* 1995, 57–62.
[26] Lee, L.-H., Moyer, B., and Arends, J., Instruction fetch energy reduction using loop caches for embedded applications with small tight loops, *Proc. Int. Symp. on Low-Power Electron. Design,* San Diego, CA, 1999, 267.
[27] Tang, W., Gupta, R., and Nicolau, A., Design of a predictive filter cache for energy savings in high performance processor architectures, *Proc. Int. Conf. on Comput. Design,* 2001, 68–73.

[28] Itoh, K., Low-Power Memory Design, in *Low-Power Design Methodologies*, Rabaey, J. and Pedram, M., Eds., Kluwer Academic Publishers, Dordrecht, 1996, 201.
[29] Hasegawa, A., et al. SH3: high code density, low power, *IEEE Micro.,* 1995, 11.
[30] Ko, U., Balsara, P.T., and Nanda, A.K., Energy optimization of multilevel cache architectures for RISC and CISC processors, *Trans. IEEE Very Large-Scale Integration (VLSI) Syst.,* 6(2):1998, 299.
[31] Lee, H.S., and Tyson, G.S., Region-based caching: an energy-delay efficient memory architecture for embedded processors, *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Syst.,* 2000, 120.
[32] Wilson, S.J.E. and Jouppi, N.P., An enhanced access and cycle time model for on-chip caches. Technical report 93/5, Digital Equipment Corporation, Western Research Laboratory, Palo Alto, CA, 1994.
[33] Shimazaki, Y., An 8-mW, 8-KB cache memory using an automatic power-save architecture for low-power RISC microprocessors, *IEICE Trans. Electron.,* E79-C(12):1693, 1996.
[34] Calder, B. and Grunwald, D., Next cache line and set prediction, *Proc. Int. Symp. on Computer Architecture,* 1995, 287.
[35] Inoue, K., et al. Way predicting set-associative cache for high performance and low energy consumption, *Proc. 1999 Int. Workshop on Low-Power Design,* 1999, 273.
[36] Powell, M., et al. Reducing set-associative cache energy via way prediction and selective direct-mapping, *Proc. 34th Annu. IEEE/ACM Int. Symp. on Microarchitecture,* 2001, 54–65.
[37] Batson, B. and Vijaykumar, T.N., Reactive associative caches, *Proc. 2001 Int. Conf. on Parallel Architectures and Compilation Techniques,* 2001, 49–60.
[38] Huang, M., et al. L1 data cache decomposition for energy efficiency, *Proc. Int. Symp. on Low-Power Electron. Design,* Huntington Beach, CA, 2001, 10.
[39] Kim, S., et al. Power-aware partitioned cache architecture, *Proc. Int. Symp. on Low-Power Electron. Design,* Huntington Beach, CA, 2001, 64.
[40] Albonesi, D.H., Selective cache ways: on-demand cache resource allocation, *Proc. 32th Int. Symp. on Microarchitecture,* 1999, 248.
[41] Hu, Z., Kaxiras, S., and Martonosi, M., Improving cache power efficiency with an asymmetric set-associative cache, *Proc. Workshop on Memory Performance Issues,* Goteborg, Sweden, June 30–July 1, 2001.
[42] Yang, S.-H., et al. Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay, *Proc. 8th Int. Symp. on High-Performance Comput. Architecture,* 2003, 151–161.
[43] Gonzales, A., Aliagas, C., and Valero, M., A data cache with multiple caching strategies tuned to different types of locality, *Proc. Int. Conf. on Supercomputing,* 1995, 338.
[44] Inoue, K., Kai, K., and Murakami, K., High-bandwidth, variable line-size cache architecture for merged DRAM/logic LSIs, *IEICE Trans. on Electron.,* E81C(9):1438, 1999.
[45] MIPS Corporation. *MIPS R3000 Hardware Manual,* MIPS Corporation, Mountain View, CA.
[46] Tang, W., Veidenbaum, A.V., and Gupta, R., Architectural adaptation for power and performance, *Proc. 14th Annu. IEEE Int. ASIC/SOC Conf.,* Oct. 2001, 127–130.
[47] Witchel, E. and Asanovic, K., The span cache: software controlled tag checks and cache line size, in *Proc. Workshop on Complexity-Effective Design, ISCA-28,* Goteborg, 2001.
[48] Burd, T., Energy-efficient processor system design, Ph.D. thesis, University of California–Berkeley, May 2001.
[49] Ma, A., Zhang, M., and Arsanovic, K., Way memorization to reduce fetch energy in instruction caches, *Proc. Workshop on Complexity-Effective Design, ISCA-28,* Goteborg, Sweden, June 30–July 1, 2001.
[50] Inoue, K., Moshnyaga, V. G., and Murakami, K., A history-based I-cache for low-energy multimedia applications, *Proc. Int. Symp. on Low-Power Electron. and Design,* 2002, 148.
[51] Witchel, E., et al. Direct addressed caches for reduced power consumption, *Proc. 34th Annu. IEEE/ACM Int. Symp. on Microarchitecture,* Austin, TX, Dec. 1–5, 2001.

[52] Petrov, P. and Orailoglu, A., Energy-frugal tags in reprogrammable I-caches for application-specific embedded processors, *Proc. ACM CODES,* May 6–8, 2002, 181–186.

[53] Petrov, P. and Orailoglu, A., Data cache energy minimization through programmable tag size matching to applications, *Proc. IEEE Int. Symp. on System Synthesis,* Sept. 30–Oct. 3, 2001.

[54] Villa, L., Zhang, M., and Asanovic, K., Dynamic zero compression for cache energy reduction, *Proc. 33rd Annu. IEEE/ACM Int. Symp. on Microarchitecture,* Monterey, CA, Dec. 10–13, 2000, 214–222.

[55] Yang, J. and Gupta, R., Frequent value locality and its applications, *ACM Trans. on Embedded Comp. Syst.,* 2(3):1, 2002.

[56] Mahapatra, N., et al. The potential of compression to improve memory system performance, power consumption and cost, *Proc. 22nd IEEE Int. Performance, Computing, and Communication Conf.,* Phoenix, AZ, Apr. 9–11, 2003.

[57] Mai, K., et al. Low-power SRAM design using half-swing pulse-mode techniques. *IEEE J. Solid-State Circuits,* 33, 1659, 1998.

[58] Moshnyaga, V.G. and Tsuji, H., Reducing cache energy dissipation by using dual voltage supply, *IEICE Trans. on Fundamentals,* E84-A(11):2762, 2001.

[59] Hamzaoglu, F., et al. Dual-Vt SRAM cells with full-swing single ended bit-line sensing for high-performance on-chip cache in 0.13-um technology generation, *Proc. 2000 Int. Symp. on Low-Power Electron. and Design,* July 2000, 15–20.

[60] Powell, M.D., et al. Gated-V_{dd}: a circuit technique to reduce leakage in cache memories, *Proc. 2000 ACM/IEEE Int. Symp. on Low-Power Electron. and Design,* 2000, 90.

[61] Kaxiras, S., Hu, Z., and Martonosi, M., Cache decay: exploiting generational behavior to reduce cache leakage power, in *Proc. 28th Int. Symp. on Comput. Architecture,* June 30–July 4, 2001, 240–251.

[62] Zhou, H., et al. Adaptive mode control: a static-power-efficient cache design, *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques,* Sept. 8–12, 2001, 61–70.

[63] Flautner, K., et al. Drowsy caches: simple techniques for reducing leakage power, *Proc. of the 29th Annu. Int. Symp. on Comput. Architecture,* Anchorage, AK, May 25–29, 2002, 148–150.

[64] Halter J.P. and Najm, F., A gate-level leakage power reduction method for ultra-low-power CMOS circuits, *Proc. IEEE Custom Integrated Circuits,* 1997, 457.

[65] Ye, Y., Borkar, S., and De, V., A technique for standby leakage reduction in high-performance circuits, *Proc. IEEE Symp. VLSI Circuits,* 1998, 40.

[66] Heo S., et al. Dynamic fine-grain leakage reduction using leakage-biased bitlines, *Proc. 29th Int. Symp. on Computer Architecture,* Anchorage, AK, May 25–29, 2002.

9

Memory Organization for Low-Energy Embedded Systems

Alberto Macii
Politecnico di Torino

9.1 Introduction

Memory design for multi-processor and embedded systems has always been a crucial problem, because system-level performance depends strongly on memory organization.

The proliferation of embedded systems, and the corresponding new chip and chip-set designs, have brought additional attention to storage units. Indeed, the heterogeneity of components and structures within embedded systems and the possibility of using application-specific storage systems have added a new dimension to memory design. Moreover, new degrees of freedom have been opened since the introduction of embedded memory arrays in different technologies, such as SRAMs, DRAMs, EEPROMs, and FLASH, and their realization on the same silicon substrate hosting the processing units.

Embedded systems are often designed under stringent energy consumption budgets, to limit heat generation and battery size. Because memory systems consume a significant amount of energy to store and to forward data, it is then imperative to balance energy consumption and performance in memory design. Contemporary system design focuses on the trade-off between performance and energy consumption in processing and storage units, as well as in their interconnections. Although memory design is as important as processor design in achieving the desired design objectives, the former topic has received less attention than the latter in the literature.

There are two key issues in low energy memory design for embedded systems:

1. Reduce the energy consumed in accessing memories [12,14,28,29], which takes a dominant fraction of the energy budget of an embedded system for data-dominated applications [12].
2. Minimize the amount of energy consumed when information is exchanged between the processor and the memory by reducing the amount of required processor-to-memory communication bandwidth.

Regarding memory access optimization, the possibility of integrating one or more memories on the same die as the processor offers new opportunities for energy-efficient design. In fact, from one side,

accessing on-chip memory is much faster and energy-efficient than relying exclusively on off-chip memories [20,34]. On the other side, memory size and organization can be tailored to application requirements, and application-specific memory architectures can be developed to minimize memory energy for a given embedded application. Among the available solutions for memory hierarchy customization, memory partitioning has demonstrated very good potential for energy savings, as well as excellent suitability for usage in automated design environments. The principle of such a method is to subdivide the address space in several clusters and to map them to different memory banks that can be independently enabled/disabled.

Concerning communication bandwidth optimization, Burger [11] discussed that memory bandwidth is becoming more important as a metric for modern systems because of the increased instruction-level parallelism generated by superscalar or very long instruction word (VLIW) processors, and because of the density of integration, which allows shorter latencies. Unlike latency, but similar to energy, bandwidth is an average-case quantity, and is related to memory traffic. Therefore, not all solutions that reduce latency necessarily translate into bandwidth improvements, as for energy. A good way for decreasing the memory traffic, and memory energy as well, is to compress the information that is transmitted between two levels of the memory hierarchy.

This chapter focuses both on methods that aim at reducing the energy consumption by optimizing the memory hierarchy and on techniques that target the reduction of the energy consumed in memory transfers.

In particular, Section 9.2 surveys several effective memory partitioning approaches, especially focusing on methods that are suitable to be used in an automatic fashion, whereas Section 9.3 discusses solutions for information compression including both code and data.

9.2 Memory Partitioning

Within a given memory hierarchy level, energy consumption can be reduced by memory partitioning. The principle of memory partitioning is to subdivide the address space into several smaller and less energy consuming blocks, and to map such blocks to different physical memory banks that can be independently enabled and disabled. Memory partitioning by itself is a typical performance-oriented solution (because of the reduced latency due to accessing smaller blocks), and energy may be reduced only for some specific access patterns. What actually makes this class of techniques low-energy is the opportunity of selectively shutting down memory blocks that are not accessed, which has little or no effect on performance.

Arbitrary fine partitioning is prevented from the fact that an excessively large number of small banks is highly area inefficient, and imposes a severe wiring overhead, which tends to increase communication energy and performance. Partitioning-based techniques proposed in the literature differ in several aspects. First, the hierarchy level targeted for partitioning (from caches to off-chip memories). Second, the scope of partitioning: physical partitioning strictly maps the address space onto different, nonoverlapping memory blocks; logical partitioning allows some redundancy in the various blocks of the partition, with the possibility of storing addresses several times in the same level of hierarchy. Farrahi et al. [15] first studied memory partitioning to exploit sleep mode operation. This work is in the context of board-level memory optimization where memory blocks are large off-chip DRAMs that can be powered down when they are not storing live program variables, thereby eliminating memory refresh energy. Furthermore, it is assumed that activating an inactive memory incurs a significant energy cost. The technique presented by Farrahi et al. tries to cluster data into memories so that memory chips are transitioned in and out of the shutdown mode as little as possible. Several authors have analyzed partitioning of on-chip memories. In most cases, the various partitioning options at a given hierarchy level have been used as an additional dimension of the memory design space. For example, Su and Despain [33], Ko and Balsara [20], and Shiue and Chakrabarti [32] studied energy-efficient partitioned cache organizations, identifying cache sub-banking as an effective technique for reducing cache energy consumption.

Other solutions rely on hardware-based selective activation of individual ways for set-associative caches. Region-based caching proposes separate cache memories for stack data and global data, and a main cache for all other accesses. Clearly, non-accessed cache modules can be disabled for energy saving.

Coumeri and Thomas [13] studied embedded SRAMs, and described a partitioned SRAM model (called segmented configuration), providing energy models for partitioned memories as well. These techniques rely on a physical partitioning of the on-chip memory. Gonzàlez et al. [16] proposed logical partitioning, where the on-chip cache is split into a spatial and into a temporal cache to store data with high spatial and temporal correlation, respectively. This approach resorts to a dynamic prediction mechanism that can be realized without modification of the application code by means of a prediction buffer. Grun et al. [17] have extended the ideas behind this approach in the context of embedded systems for energy optimization. In their approach, data are statically mapped onto either cache, thanks to the high predictability of the access profiles in embedded programs. Depending on the application, data might be duplicated and thus be mapped to both caches.

Another class of logical partitioning techniques exploits buffer insertion along the I-cache or the D-cache or both to realize some form of cache parallelization. Such schemes can be regarded as a partitioning solution because the buffers and the caches are actually part of the same level of hierarchy. In this kind of architecture, data and instructions are explicitly replicated, and redundancy is an intrinsic feature of these approaches. Reducing the average cost of a memory access by increasing the cache-hit ratio saves energy. Another solution proposes the use of the buffer as a victim cache that is accessed on a main cache miss. In case of a buffer hit, the line is moved to the cache and returned to the CPU, while the replaced line in the cache is moved to the victim cache. In case of a buffer miss, the lower level of hierarchy is accessed and the fetched datum is copied into the main cache as well, while the replaced line in the cache is moved to the victim cache. In practice, the victim cache serves as an over-full buffer for the main cache. A similar approach has been introduced by Bahar et al. [2], where buffers are used for speculation: every cache access is marked with a confidence level, obtained by the processor state; the main cache contains misses with a high confidence level, while the buffers contains those with a low confidence level. Other techniques adopt a small associative buffer (e.g., 32 entries) in parallel to the L1-cache (called the noncritical buffer), used to "protect" the cache from being filled with noncritical (i.e., potentially polluting) data. Noncritical data are identified at run-time by monitoring the issue rate of the core. An alternative solution consists of filtering the data to be stored in the main cache through a small, highly associative cache close to the L1-cache. Unlike the victim cache (where data are kept before disposing them), the annex cache stores the data read from memory, which are copied into the main cache only on subsequent references to those data.

This idea of filtering cache accesses to reduce writes that are very likely to cause misses can be refined by selectively disabling the side buffers to save additional energy. Way-predicting caches attempt to minimize unnecessary way activation in set-associative caches, by predicting which way contains the data. Prediction is carried out based on memory access history.

9.2.1 Memory Partitioning for Low Energy

Moving from the observation drawn in Section 9.2 and finding inspirations from the existing memory partitioning techniques proposed in the literature, an automatic optimization methodology for on-chip memories to be used in embedded systems on chip (SoCs) is presented in Benini et al. [7]. Starting from the dynamic execution profile of an embedded application running on a given processor core, a multi-banked memory architecture optimally fitted to such a profile is synthesized. The rationale behind the approach we will describe in the following is to partition memory into multiple banks that can be independently accessed. Power-per-access is reduced as the size of a memory bank is decreased. On the other hand, as the number of banks increases, there is an unavoidable hardware overhead caused by:

1. Duplication of addressing and control logic
2. Increased communication resources required to transfer information

Such an overhead manifests itself in increased energy, access time, and area that prevent arbitrarily fine partitioning. Thus, finding an optimal partition with a tight constraint on the maximum number of memory banks is extremely important.

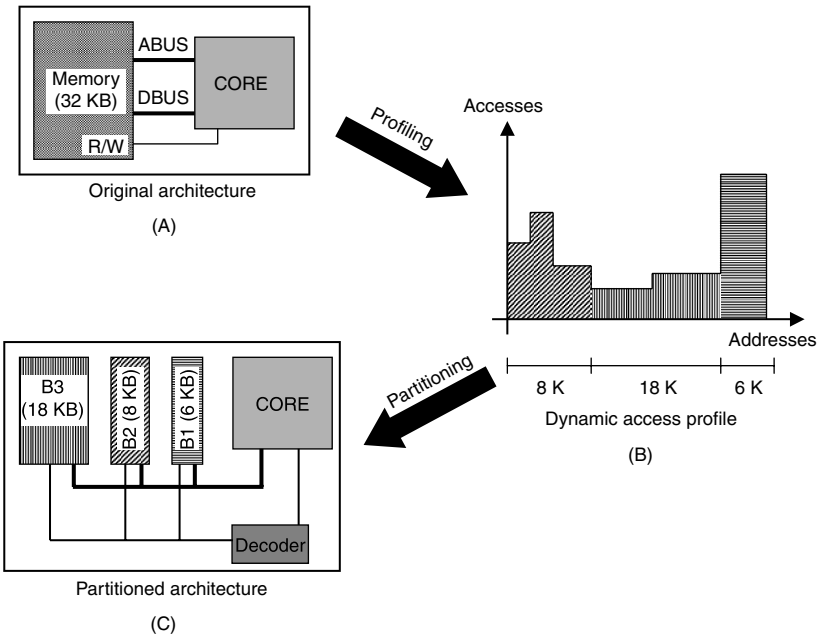


FIGURE 9.1 Example of memory partitioning.

Moving from the fact that in many nontrivial embedded applications the most frequently accessed addresses can fit into a relatively small memory space, the information that comes from the dynamic access profile, obtained through the simulation of the target application, is used to specify a range (a_{lo}, a_{hi}) of memory addresses that should be mapped onto the on-chip memory. For each address $a_{lo} \leq i \leq a_{hi}$, the profile gives the number of reads $r(i)$ and writes $w(i)$ to the address during the execution of a sample run of the target embedded application. Standard instruction-level simulators available for most processor cores can capture the profile.

In a traditional approach, all addresses in the range are mapped to a single memory array, the smallest array in the library, which is large enough to contain the specified range, as illustrated in Figure 9.1(a).

This solution is not optimal from the energy dissipation viewpoint. Assume, for the sake of illustration, that the dynamic access profile is that presented in Figure 9.1(b). A small subset of the addresses in the range has large $r(i)$ values for all its addresses (i.e., it is very "hot"; see the right horizontal-shaded slice of the profile). An energy-optimal partitioned memory organization is presented in Figure 9.1(c). It consists of three memories and a memory selection block (the decoder). The hot addresses are stored into a small memory (block B1), while two larger cuts (blocks B2 and B3) contain the other parts of the range. The average energy in accessing the memory hierarchy is decreased, because a large fraction of accesses is concentrated to a small, energy-efficient memory. Obviously, it is mandatory to account for the energy consumed in the entire partitioned memory system (i.e., the address and data buses), the decoder, and the control signals. In fact, these components introduce a nonnegligible overhead on energy consumption that may offset the advantages given by bank partitioning. Nevertheless, the obtained savings are significant especially when the access profile is highly nonuniform and high-access addresses are clustered into small banks (on average, for several embedded applications, energy savings are around 60% with respect to the traditional monolithic memory architecture).

The memory partitioning approach belongs to the class of memory optimization techniques that, moving from a given memory access trace and obtained by profiling an application, produce a customized memory hierarchy. The effectiveness of memory partitioning can be improved by adopting the concurrent optimization of memory access patterns and memory architecture. Techniques relying on such a concurrent optimization are the most powerful, yet also the most difficult to actuate. This fact is witnessed

by the few solutions proposed in the literature, the most popular being the DTSE hardware/software (HW/SW) exploration framework [12]. One of the biggest difficulties in concurrent optimization lies in the fact that the two dimensions of the problem are regarded as orthogonal: architectural optimization is viewed as a purely hardware task, while the optimization of the access patterns is viewed as a purely software task. Recently, in Macii et al. [27], the access pattern optimization problem was revisited from an architectural perspective: the design of an application-specific memory architecture is concurrently carried out with the optimization of the access patterns, yet done through the introduction of proper hardware. In practice, the access patterns are modified on the fly, without any intervention on the software application running on the processor.

Thus, the idea is to combine the memory partitioning methodology with a technique, called address clustering, which consists of reorganizing (through extra hardware) the address trace fed to a memory block, in such a way that the potential for the memory partitioning engine is maximized. This is equivalent to modifying the memory access profile, yet in a very transparent fashion to the programmer.

The address clustering problem consists of finding a relocation of a proper subset of the address space that maximizes the locality of the dynamic trace, with the ultimate objective of minimizing the energy consumption of the memory architecture for the given trace, possibly under area and cycle time constraints.

The actual energy consumption of a partitioned memory architecture is determined by the outcome of the memory-partitioning algorithm of Benini et al. [7]; however, running the partitioning engine for each candidate clustering solution may become quite computationally expensive. It is thus necessary to devise a high-level cost function that can be used into an exploration framework to evaluate the suitability of a clustering solution.

The potential of memory partitioning is related to the locality of the trace. In Macii et al. [27], it was thus defined the density of a profile, C, as the maximum value of the cumulative number of accesses for a sliding window of size W over the trace. This metric is suitable for use in an exploration engine because it provides, through a low-effort analysis of an address trace, a quantification of which percentage of the total number of addresses can be covered by clustering W words. There is then a strict correlation between the number M of addresses to be clustered and the size W of the sliding window.

Figure 9.2 reports an example of address clustering. Figure 9.2(a) depicts the original profile (i.e., as obtained by profiling the application); here, two addresses (*i* and *j*) are illustrated with bold lines, together with the sliding window W. Figure 9.2(b) depicts one instance of the swap between *i* and *j*. This transformation increases the spatial locality by aggregating highly accessed memory locations. The information about the swap is recorded in a table, as the one of Figure 9.2(c).

Results obtained by applying address clustering before partitioning demonstrate advantages ranging from 5 to 55% with respect to energy consumption of the memory architecture generated by plain application of memory partitioning.

9.3 Memory Transfer Optimization

Although memory energy consumption is relevant, additional improvements can be achieved by combining the minimization of memory and bus energy, that is, of the whole memory-processor interface. Optimizing the processor-to-memory bandwidth is one possibility to simultaneously reduce memory and bus energy.

Bandwidth can be increased either by directly reducing the processor-to-memory traffic or by indirectly compressing the information transmitted through the interface.

In most cases, the main objective of the information compression is to provide a high compression ratio; instead, parameters like compression (and decompression) time and complexity are not seen as critical ones because software compression routines or dedicated hardware units perform their functions in environments where timing, resource, and energy constraints are relaxed. Think, for example, of data compression for hard-drives or transmission over communication links, where transfer rates are significantly slower than the time required by the HW or SW compressor/decompressor to transform data in

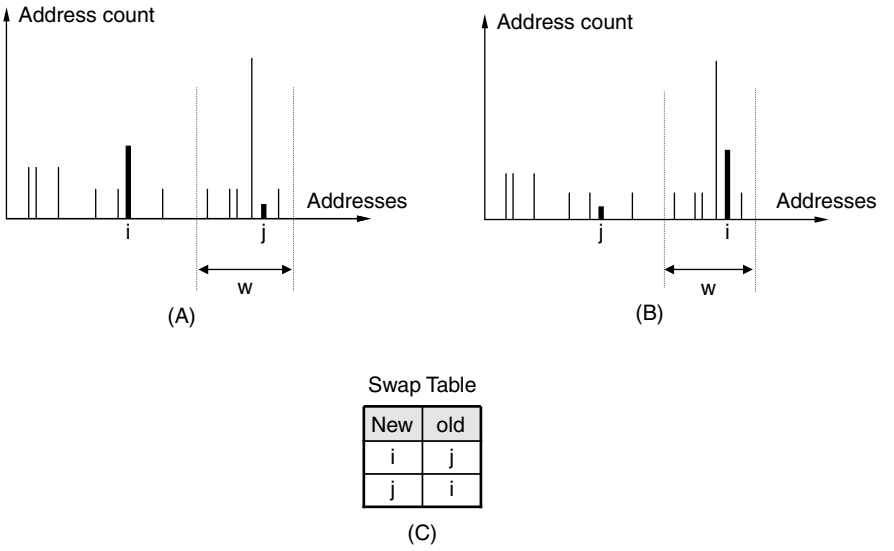


FIGURE 9.2 Example of address clustering.

the appropriate format. A different scenario must be faced when compression is applied to information being stored in memories or caches. Here, compression and decompression are subject to tight performance constraints because they should comply with data read and write speeds of modern, high-performance processors. As such, compression speed becomes the primary cost measure to be used for assessing the quality of a compression scheme. HW-assisted solutions are the only viable options in this context, where high compression ratios are traded for faster compression units. Besides speed, also size and complexity of HW compressors need to be controlled because modern SoCs call for implementations of such units near-by the core processors (for cache-compressed architectures) or between cache and memory (for memory-compressed systems). The problem of designing fast and compact HW compression units for usage in the memory path of a processor-based system has been studied extensively in the past (see, for example, Bunton and Borriello [10] and Lee et al. [22] for a survey of existing literature). Recently, HW memory compression has found its way in a number of commercial designs (see, for example, the MXT Pentium-based server by IBM [1]). Although reduction of memory and bus bandwidth has been, historically, the main motivation for resorting to HW-assisted memory compression, recent studies have demonstrated that this approach can also be exploited when the ultimate target is energy (or power) minimization of a processor-based system. In particular, successful attempts were made to limit energy consumption in systems containing embedded processors by reducing energy requirements of I-caches [5,26], program memory [6,36], and data memory [8].

In the next two subsections, we present a number of approaches that target the reduction of the memory-processor traffic and, as a consequence, the number of accesses to memory locations, exploiting memory compression. In particular, Subsection 9.3.1 ("Code Compression") discusses code compression techniques, whereas Subsection 9.3.2 ("Data Compression") focuses on data compression methodologies.

9.3.1 Code Compression

Currently, many embedded processors are based on high-performance RISC architectures, with on-chip cache [20] and full support for complex memory systems and peripheral controllers [30]. System integrators usually purchase these processors, as well as their software development environments, from third-party companies that specialize in embedded core design.

One of the key challenges in designing a complex system around a high-performance embedded RISC processor is to ensure sufficient instruction fetch bandwidth to keep the execution pipeline busy. The

regularity of RISC instruction sets eases application and compiler development, but hinders code compaction. For this reason, designers and researchers have put significant effort in devising techniques for improving code density and reducing instruction-related costs, in terms of speed, area, and energy [3].

Numerous code compression techniques have been proposed for reducing instruction memory size in low-cost embedded applications (refer to Lefurgy [24] for an extensive set of references). The basic idea is to store programs in compressed form and decompress them on the fly at execution time. Later, researchers have realized that code compression can be beneficial for energy as well, because it reduces the energy consumed in reading instructions from memory and communicating them to the processor core [6,26,36].

Code compression leverages well-known lossless data compression techniques [4], but it is characterized by two distinctive constraints. First, it must be possible to decompress a program in relatively small blocks, as instructions are fetched, and starting from several points inside the program (i.e., branch destinations). Thus, traditional lossless techniques that decompress a stream starting from a single initial point are not applicable without changes. Second, the decompressor should be small, fast, and energy-efficient because the corresponding savings in memory size and energy must amortize its hardware, without compromising the performance.

For simple processors with no instruction cache, the hardware decompression block is either merged with the processor core itself or placed between program memory and processor. The first solution has been implemented in several commercial core processors, in the form of a "dense" instruction set, with short instructions (e.g., ARM Thumb [31] and MIPS16 [19] instruction sets). The second solution has been investigated in several articles [6,25,36]. Supporting restricted instruction sets requires changes to the core architecture, while an external decompressor does not. Furthermore, with an external decompressor it is possible to aggressively tailor code compression to a specific embedded application. Thus, external decompression is well suited for embedded designs employing third-party cores.

The basic assumption of the method presented in Yoshida et al. [36] is that the firmware running on a given embedded processor normally uses only a small subset of the instructions supported by the processor. By replacing such instructions with binary patterns of limited width (i.e., $\lceil \log_2 N \rceil$, where N is the number of distinct instructions appearing in the code), memory bandwidth usage can be reduced, thus decreasing the total energy. Notice that two k-bit instructions are said to be distinct if they differ by at least one bit.

The solution proposed in Yoshida et al. [36] does not require ad-hoc compilers; in fact, the original machine instructions can be automatically replaced by $\lceil \log_2 N \rceil$-bit instructions by means of a script after the subset of instructions actually used by the program is identified through execution profiling or instruction-level simulation, and the number $\lceil \log_2 N \rceil$ is determined. The original machine code can thus be compressed to reduce the memory bandwidth that is needed to execute the program. The so-called instruction decompression table and the related control circuitry can be designed and placed between the processor and the memory. Thus, the architecture of the core processor is left unchanged. This is a big plus for system designers employing third-party, off-the-shelf cores and microcontrollers that are either not disclosed (IP hard or soft macros) or not easy to modify.

The work in Benini et al. [6] describes a new technique that builds upon the method of Yoshida et al. [36] by overcoming its major limitation: if the number of instructions used by the embedded code gets large, so does the number of bits of the compressed instructions. Besides increasing the size of the instruction decompression table, this may excessively complicate the implementation of the controller that handles instruction fetching and decoding, especially when the bit-width of the compressed instructions is not compatible with the available memory-addressing scheme (e.g., bit-width different from a multiple of 8 on a byte-addressable memory).

Moving from the observation that the number of instructions used by most programs, although limited with respect to the total number of instructions supported by the processor, has a highly nonuniform statistical distribution. In other words, some instructions are usually much more used than others, it is convenient considering for compression only the instructions used by the embedded code with the highest execution probability. This solution allows fixing *a priori* the bit-width of the compressed instructions;

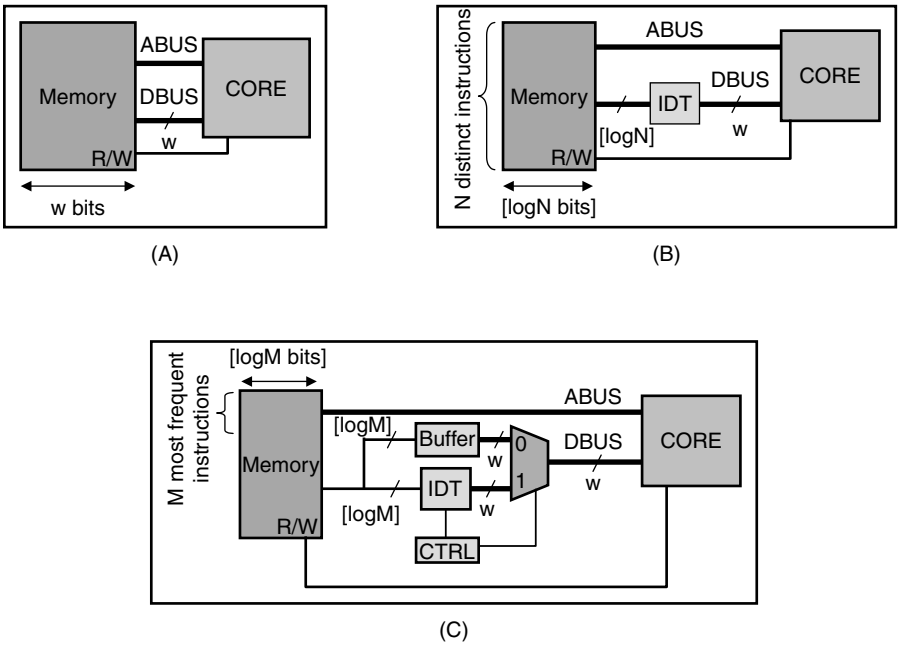


FIGURE 9.3 (a) Original architecture, (b) modifications proposed in Yoshida et al. [36], and (c) in Benini et al. [6].

in this way the size of the instruction decompression table is fixed and limited, the instruction fetching/decompression logic has reduced complexity, and the energy required to fetch the program from memory is minimized. Instruction decompression is performed on-the-fly by a hardware block located between processor and memory. No changes to the processor architecture are required because it always executes full-size instructions. Thus, this technique is well suited for systems employing IP cores whose internal architecture cannot be modified.

Figure 9.3(a) depicts a typical conceptual architecture of a processor-memory system, while Figure 9.3(b) and Figure 9.3(c) report the modified architectures proposed in Yoshida et al. [36] and Benini et al. [6], respectively.

In more advanced architectures that contain instruction caches, the decompressor can be placed either between the I-cache and the main memory (decompress on cache refill, or DCR architecture) or between the processor and the I-cache (decompress on fetch, or DF architecture). Both alternatives have been investigated in the recent literature [21,26].

The approach described in Larin and Conte [21] targets VLIW processors and compresses instructions using the Huffman algorithm. Basic blocks of compressed instructions are transferred and stored into the I-cache atomically. Compressed instructions are not aligned to cache line boundaries. On a cache access, two consecutive cache lines are decompressed and stored in a level-zero buffer. The following instructions are fetched in sequence from the buffer, until it is emptied, or a branch is executed. The hardware decompressor may have a very high cost; in fact, fetching and decoding two cache lines at a time imposes parallel Huffman decoding of multiple instructions in one clock cycle (even single-instruction Huffman decoding requires a quite large hardware block). Furthermore, branch targets addresses in compressed code are stored in a dedicated address remapping memory that must be accessed on every taken branch.

In Lekatsas et al. [26], it was demonstrated that from an energy and performance viewpoint, the DF architecture is superior to the DCR architecture, when decompressor overhead is small. The main reason for this effect is that instructions are stored in cache in a compressed fashion, effectively increasing cache capacity. Current silicon implementations of code compression are based on the DCR architecture

[18,23]; however, indicating that reducing decoding overhead is a nontrivial task that still entails significant challenges.

The main issue with the DF approach is that decompression is performed on every instruction fetch. In other words, the decompressor is on the critical path for the execution of every instruction, not only for cache refills. If its delay is not small, it may significantly slow down execution. Furthermore, it consumes energy on every instruction fetch, while in the DCR architecture it can be activated only on cache refills. Careful implementation of the decompression unit is thus key for making DF applicable in practice.

In Benini et al. [5], a novel DF architecture that focuses on reducing decoding overhead on energy and performance is proposed. This technique guarantees that storage requirements for the compressed program always decrease. Furthermore, the compression algorithm has been designed specifically for fast and low-energy decompression during cache lookup. Compressed instructions are always aligned to cache line boundaries, branch destinations are word-aligned, and instruction decompression is based on a single lookup into a small (and fast) memory buffer. The analysis is not limited to the architecture level, but present a complete implementation of the cache-decompressor block, including detailed analysis of its energy and delay. The achieved code size reductions are on average around 28%, while from the energy point of view the improvements vary a lot depending on cache size, original and compressed code size, dynamic memory access profile, and kind of adopted program memory (i.e., on-chip vs. off-chip). For example, for a 4 KB cache and an on-chip program memory, average energy savings are around 30%. This value grows to 50% for a system with a cache of the same size but an off-chip program memory.

9.3.2 Data Compression

The principle exploited in code compression (namely, reduction of memory traffic), can be extended to the case of data, with some additional difficulties.

Data compression techniques, together with hardware architectures, have been introduced recently in Benini et al. [8,9].

The approach of Benini et al. [8] relies on a fixed-dictionary scheme. It resorts to data profiling information to selectively compress cache lines before they are written back to the main memory, and to quickly decompress them when a cache refill operation is started. This solution is particularly suited to embedded systems, where the collection of data statistics to be used by compressor and decompressor is much more predictable than in general-purpose systems. The obtained reductions of memory traffic were around 42% for a significant number of benchmark programs. In this work, the authors consider systems with compressed main memory (i.e., information is stored in caches in uncompressed format), in which the compression HW is placed between caches and main memory.

A possible generalization of the approach of Benini et al. [8] to the case of general-purpose systems was sketched in Benini et al. [9]. The idea was that of avoiding the profiling step by doing online prediction of data statistics. Here, the architecture is able to adaptively update the compression dictionary according to the current data statistics. This improvement allows removal of the main limitation constituted by the need of off-line data profiling. In other words, the adaptive algorithm is applicable to systems where several programs need to be executed, and thus ad hoc data profiling information cannot be collected before the system is started.

In these approaches, the fundamental difference between code and data compression is that, for the latter, both compression and decompression are needed during the execution of a program, while for the former only decompression is required. This fact has far-reaching implications on the applicable compression algorithms and architectures; for instance, it rules out highly asymmetric schemes, where compression is much more involved than decompression.

Clearly, the energy cost of the compressor also needs to be accounted when evaluating the feasibility of an HW-based memory compression scheme. It was observed in Benini et al. [5] that the overhead introduced by the extra hardware is roughly proportional to the amount of storage space the compressor requires. As such, if care is given to the choice of the compression algorithm and to its implementation, the achieved bus and memory energy savings will offset the energy cost of the compressor.

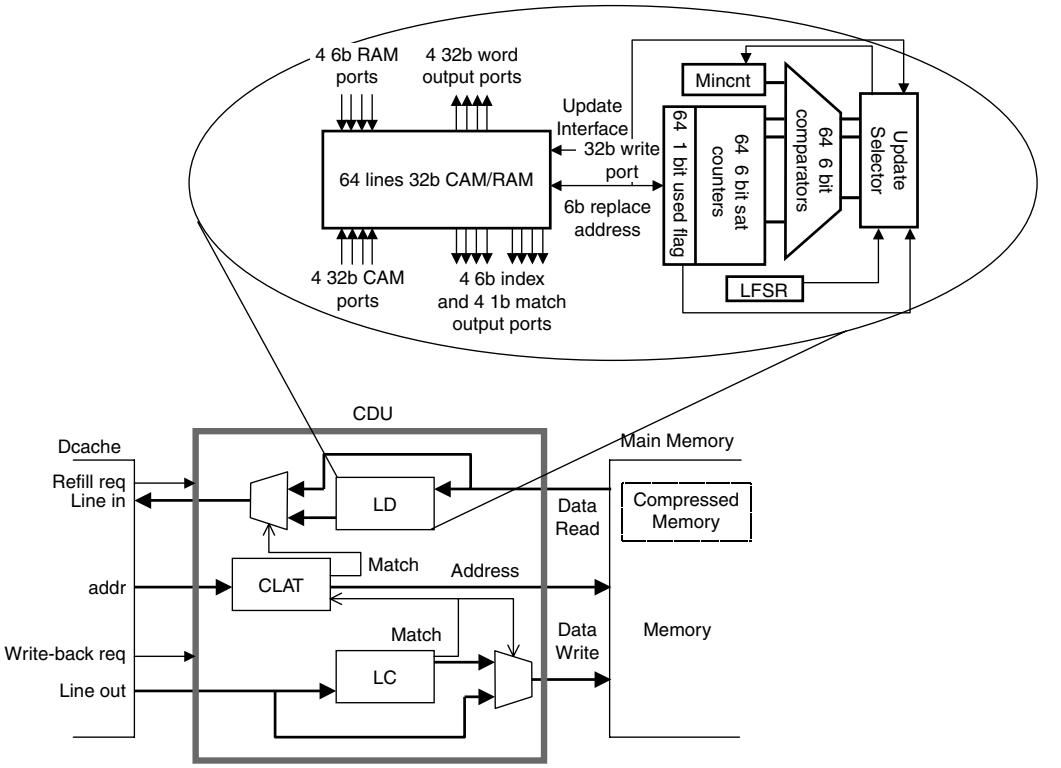


FIGURE 9.4 CDU architecture and implementation of a comp–decomp table.

Just to give to the reader the flavor of how the compressor can be implemented, Figure 9.4 reports a conceptual block diagram of the compression-decompression unit (CDU) adopted in Benini et al. [8,9] with the basic interface signals, together with a real hardware implementation of a compression–decompression table.

The CDU contains three major functional blocks, namely, the line compressor (LC), the line decompressor (LD), and the compressed line address table (CLAT). The compression–decompression table highlighted in Figure 9.4 is the basis of the LC and LD blocks.

The hardware implementation of the CDU is based on content-addressable memories and random-access memories (CAMs and RAMs, respectively), and it is mainly targeted toward energy minimization in the cache-bus-memory subsystem with a strict constraint on performance. As a result, average memory reductions evaluated on several benchmarks are around 23%, at no performance penalty (actually, on average, performance improved by 4%). Comparison to the memory traffic reductions achieved with the profile-driven method of Benini et al. [8] demonstrates clearly that the adaptive approach performs reasonably well in this respect (31% reductions, against 42% for the profile-driven solution).

A similar compression technique is the compressed cache presented in Yang et al. [35]. It is based on the frequent value locality, that is, the fact that very few values (typically small integers) account for usually around half of the total memory accesses, for most benchmarks. This allows storing the selected values in a compressed form. Compression/decompression is performed on the fly on accesses from/to the next hierarchy level.

9.4 Conclusions

Embedded systems are now becoming ubiquitous; in particular, they have large applicability in mobile, battery-operated personal communication systems, for which energy consumption is a major constraint.

Among the various contributors to the system's energy budget, memory plays a preeminent role; in fact, reading and writing data to the memory hierarchy is an operation that takes place very often, especially in data-dominated applications (e.g., video and audio playing). As such, memory energy optimization is one of the most promising and successful approaches to system power minimization.

References

[1] S. Arramreddy, et al. IBM X-Press memory compression technology debuts in a ServerWoks NorthBridge, *HOT Chips 12 Symp.,* Stanford University, Stanford, CA, August 2000.

[2] R.I. Bahar, G. Albera, and S. Manne, Power and performance trade-offs using various caching strategies, *ISLPED-98: ACM/IEEE Int. Symp. on Low-Power Electron. and Design,* pp. 64–69, Monterey, CA, August 1998.

[3] R. Bajwa, et al. Instruction buffering to reduce power in processors for signal processing, *IEEE Trans. on Very Large Scale Integration (VLSI) Syst.,* Vol. 5, No. 4, pp. 417–424, December 1997.

[4] T. Bell, J. Cleary, and I. Witten, *Text Compression,* Prentice Hall, New York, 1990.

[5] L. Benini, A. Macii, and A. Nannarelli, A code compression architecture for cache energy minimization in embedded systems, *IEEE Proc. — Comput. and Digital Techniques,* Vol. 149, No. 4, pp. 157–163, July 2002.

[6] L. Benini, A. Macii, E. Macii, and M. Poncino, Minimizing memory access energy in embedded systems by selective instruction compression, *IEEE Trans. on Very Large-Scale Integration (VLSI) Syst.,* Vol. 10, No. 5, pp. 521–531, October 2002.

[7] L. Benini, L. Macchiarulo, A. Macii, and M. Poncino, Layout-driven memory synthesis for embedded systems-on-chip, *IEEE Trans. on Very Large-Scale Integration (VLSI) Syst.,* Vol. 10, No. 2, pp. 96–105, April 2002.

[8] L. Benini, D. Bruni, A. Macii, and E. Macii, Hardware-assisted data compression for energy minimization in systems with embedded processors, *DATE-02: IEEE Design Automation and Test in Europe,* Paris, France, pp. 449–453, March 2002.

[9] L. Benini, D. Bruni, A. Macii, E. Macii, and B. Riccò, An adaptive data compression scheme for memory traffic minimization in processor-based systems, *ISCAS-02: IEEE Int. Symp. on Circuits and Syst.,* pp. IV-866–IV-869, Scottsdale, AZ, May 2002.

[10] S. Bunton and G. Borriello, Practical dictionary management for hardware data compression, *Commn. ACM,* Vol. 35, No. 1, pp. 95–104, January 1992.

[11] D.C. Burger, Hardware techniques to improve the performance of the processor/memory interface. Ph.D. dissertation, University of Wisconsin–Madison, 1998.

[12] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology Exploration for Memory Optimization for Embedded Multimedia System Design,* Kluwer, Dordrecht, 1998.

[13] S.L. Coumeri and D.E. Thomas, Memory modeling for system synthesis, *ISLPED-98: ACM/IEEE Int. Symp. on Low-Power Electron. and Design,* pp. 179–184, Monterey, CA, August 1998.

[14] S. Coumeri and D.E. Thomas, Memory modeling for system synthesis, *IEEE Trans. on Very Large-Scale Integration (VLSI) Syst.,* Vol. 8, No. 3, pp. 327–334, June 2000.

[15] A. Farrahi, G. Tellez, and M. Sarrafzadeh, Memory segmentation to exploit sleep mode operation, *DAC-32: ACM/IEEE Design Automation Conf.,* pp. 36–41, San Francisco, CA, June 1995.

[16] A. Gonzàlez, C. Aliagas, and M. Valero, A data-cache with multiple caching strategies tuned to different types of locality, *ICS-95: ACM Int. Conf. on Supercomputing,* pp. 338–347, Barcelona, Spain, July 1995.

[17] P. Grun, N. Dutt, and A. Nicolau, Access pattern based local memory customization for low-power embedded systems, *DATE-01: IEEE Design Automation and Test in Europe,* pp. 778–784, Munich, Germany, March 2001.

[18] IBM Corporation, *CodePack PowerPC Code Compression Utility, User's Manual Version 3.0,* IBM Corporation, Austin, Texas, 1998.

[19] K. Kissel, MIPS16: high-density MIPS for the embedded market. Technical report, Silicon Graphics MIPS Group, Mountain View, CA, 1997.

[20] U. Ko and P. Balsara, Energy optimization of multilevel cache architectures for RISC and CISC processors, *IEEE Trans. on Very Large Scale Integration (VLSI) Syst.,* Vol. 6, No. 2, pp. 299–308, June 1998.

[21] S. Larin and T. Conte, Compiler-driven cached code compression schemes for embedded ILP processors, *MICRO-32: 32nd Annual Int. Symp. on Microarchitecture,* pp. 82–92, Haifa, Israel, November 1999.

[22] J.-S. Lee, W.-K. Hong, and S.-D. Kim, Design and evaluation of a selective compressed memory system, *ICCD-99: IEEE Int. Conf. on Comput. Design,* pp. 184–191, Austin, TX, March 1999.

[23] C. Lefurgy, E. Piccinini, and T. Mudge, Evaluation of a high performance code compression method, *MICRO-32: 32nd Annu. Int. Symp. on Microarchitecture,* pp. 93–102, Haifa, Israel, November 1999.

[24] C. Lefurgy, Efficient execution of compressed programs. Doctoral dissertation, University of Michigan, Ann Arbor, MI, 2000.

[25] H. Lekatsas and W. Wolf, Code compression for embedded systems, *DAC-35: ACM/IEEE Design Automation Conf.,* pp. 516–521, San Francisco, CA, June 1998.

[26] H. Lekatsas, J. Henkel, and W. Wolf, Code compression for low-power embedded systems, *DAC-37: ACM/IEEE Design Automation Conf.,* pp. 294–299, Anaheim, CA, June 2000.

[27] A. Macii, E. Macii, and M. Poncino, Improving the efficiency of memory partitioning by address clustering, *DATE-03: IEEE Design Automation and Test in Europe,* pp. 18–23, Munich, Germany, March 2003.

[28] P. Panda and N. Dutt, *Memory Issues in Embedded Systems-on-Chip Optimization and Exploration,* Kluwer, Dordrecht, 1999.

[29] J. Rabaey and M. Pedram, *Low-Power Design Methodologies,* Kluwer, Dordrecht, 1996.

[30] S. Santhanam et al. A low-cost, 300-MHz, RISC CPU with attached media processor, *IEEE J. Solid-State Circuits,* Vol. 33, No. 11, pp. 1829–1839, November 1998.

[31] S. Segars, K. Clarke, and L. Goudge, Embedded control problems, thumb and the ARM7TDMI, *IEEE Micro*, Vol. 15, No. 5, pp. 22–30, October 1995.

[32] W. Shiue and C. Chakrabarti, Memory exploration for low-power, embedded systems, *DAC-36: ACM/IEEE Design Automation Conf.,* pp. 140–145, New Orleans, LA, June 1999.

[33] C.L. Su and A.M. Despain, Cache design trade-offs for power and performance optimization: a case study, *ISLPD-95: ACM/IEEE Int. Symp. on Low-Power Design,* pp. 63–68, Dana Point, CA, April 1995.

[34] T. Watanabe, R. Fujita, and K. Yanagisawa, Low-power and high-speed advantages of DRAM-logic integration for multimedia systems, *IEICE Trans. on Electron.,* Vol. E80-C, No. 12, pp. 1523–1531, December 1997.

[35] J. Yang, Y. Zhang, and R. Gupta, Frequent value compression in data caches, *MICRO-33: IEEE/ACM 33rd Int. Symp. on Microarchitecture,* pp. 258–265, Monterey, CA, December 2000.

[36] Y. Yoshida, B.-Y. Song, H. Okuhata, T. Onoye, and I. Shirakawa, An object code compression approach to embedded processors, *ISLPED-97: ACM/IEEE Int. Symp. on Low-Power Electron. and Design,* pp. 265–268, Monterey, CA, August 1997.

II

Low-Power Systems on Chips

10

Power–Performance Trade-Offs in Design of SoCs

Victor Zyuban
Philip Strenski
IBM Watson Research Center

10.1 Introduction

The design and implementation of processor cores is characterized by conflicting requirements of the ever increasing demand for higher performance and, usually, stringent power budget. Thus, compromises between performance and power need to be made early in the design cycle. In the design of a processor core, a very specific power budget typically exists, but power can be traded for performance in several ways.

At the system level, varying power supply is the most straightforward and well-understood method for controlling power. One advantage of this method is that power supply can typically be adjusted within a certain range even after the chip has been manufactured. In addition, scaling V_{dd} around the nominal value in application specific integrated circuits (ASIC) foundry technologies has a known cost which is "typically" 2% in energy per 1% in performance, although it can be anywhere from 0.5% to more than 30% in energy per 1% in performance. A notion of voltage intensity has recently been introduced [21] to quantify power–performance trade-offs through varying the power supply. Although a very powerful technique, scaling V_{dd} may have a relatively high performance cost for saving power in a processor core that does not meet its power budget, as discussed in Section 10.3 of this chapter.

Another method for making power–performance trade-offs is technology scaling, such as shrinking the oxide thickness and effective channel length. Although, such trade-offs are not generally available to average ASIC customers, some foundry technologies provide libraries and transistors with multiple threshold voltages, and some high-end microprocessor designs work with foundries (typically their own) to engineer these parameters effectively.

At the circuit level, power and performance can be traded by changing transistor sizes and power levels of ASIC cells, controlled by changing transistor tuning targets, or by restructuring logic to increase or

decrease the parallelism in circuits, for example, perform more computations in parallel in order to reduce the critical path. These trade-offs are controlled by either custom or logic designers or by running synthesis tools with different directives. Although more difficult to quantify, this method for power–performance trade-offs is at least as powerful as scaling the power supply. By scaling circuits, in "typical" designs that we analyzed, one percent in performance could be traded for from 0.5 to 5% in energy, or even higher, if the frequency target is too aggressive. The concept of *hardware intensity* was introduced in Zyuban and Strenski [21] to quantify power–performance trade-offs through scaling circuits.

Finally, scaling the processor core microarchitecture and, possibly instruction set architecture (ISA), is one more way for trading power and performance. This method involves changing machine organization, such as pipeline depth, issue width, the set of functional units, bypasses, number of ports and entries in queues and register files, the sizes of branch predictors, and other structures of the microarchitecture. Scaling microarchitecture is even a more powerful method for power–performance trade-offs than voltage and circuit scaling, but it is more difficult to quantify and optimize, and can only be used at early stages of the design. A concept of architectural complexity was introduced in Zyuban [19] to analyze power–performance trade-offs at the ISA and microarchitectural levels. It has been demonstrated that architectural complexity cannot only be measured but also set as a design target [10].

It was demonstrated in Zyuban and Strenski [21] that to develop an energy-efficient processor core, design decisions at all levels must be balanced in such a way that all forms of spending power, described above, have a similar marginal cost. The following sections summarize some of the most important formulas for balancing hardware intensity and power supply voltage derived in Zyuban and Strenski [21], and give a graphical interpretation of the major result. Then, the formula for balancing architectural complexity with voltage and hardware intensity, and the iterative process of refining the processor core architecture in the power–performance space are discussed in detail. Then, because making power-balanced decisions at the architectural level plays such an important role in the development of the energy-efficient processor cores, we give a derivation in Section 10.4 of a new form of the architectural energy-efficiency criterion that does not require evaluation of the relative changes in processor frequency. Section 10.5 discusses other power–performance metrics that are commonly used in the architectural community and some common mistakes made by architects when applying these metrics. Section 10.6 gives some examples of using the architectural energy-efficiency criterion, and Section 10.7 concludes the chapter.

10.2 Hardware Intensity

The concept of hardware intensity η was introduced in Zyuban and Strenski [21] as a quantitative measure of how aggressively the circuits in a processor are tuned to meet a target clock frequency (see related work in Brodersen et al. [1], Hofstee [9], and Oklobdzija et al. [14]). Hardware intensity shows the energy cost (in %) required to improve the delay D of a hardware macro by 1% through restructuring the logic and retuning the circuits, at a fixed power supply v,

$$\eta = -\left.\frac{D\partial E}{E\partial D}\right|_{\substack{\text{fixed}\\ \text{v}}} \quad \text{or} \quad \eta = -\left.\frac{\%E}{\%D}\right|_{\substack{\text{through}\\ \text{retuning}}} \tag{10.1}$$

Alternatively, we can define the hardware intensity η as a parameter in the cost function for optimizing hardware:

$$F_{cost}(E,D) = (E/E_0)(D/D_0)^{\eta} \quad \eta \geq 0, \tag{10.2}$$

where D is the critical path delay through the circuit, E is the average energy dissipated per cycle, D_0 and E_0 are the corresponding lower bounds that can be achieved through tuning and logic restructuring for

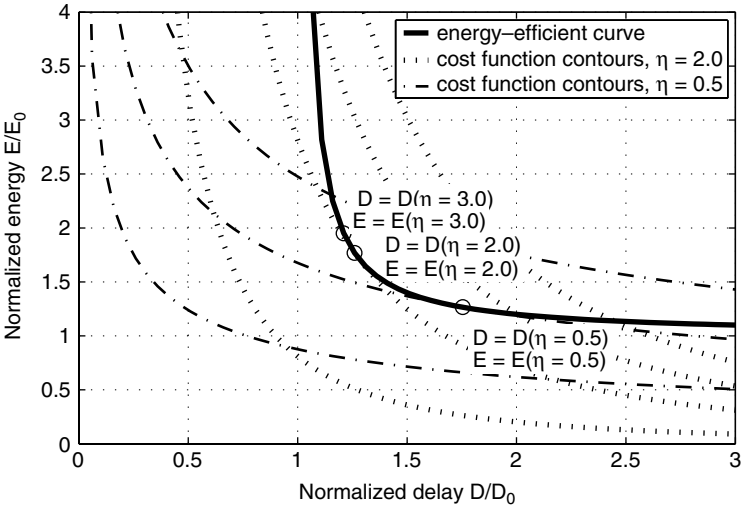


FIGURE 10.1 Typical energy-efficient curve and constant cost function contours for $\eta = 0.5$ and $\eta = 2.0$.

a fixed supply voltage. Under a very general assumption that the curvature of the energy-delay curve is larger than the curvature of the contour of the cost function (Equation (10.2)) at any point in the two tangent, $\frac{D^2}{E}\frac{\partial^2 E}{\partial D^2} > \eta(\eta+1)$, we can show that for any power supply voltage v, every point on the energy-delay curve corresponds to a certain value of hardware intensity η, $0 \le \eta < +\infty$. Then, the energy-delay curve in the energy-vs.-delay coordinates can viewed as a parameterized curve: $D = D(\eta, v)$, $E = E(\eta, v)$.

Figure 10.1 gives a graphical interpretation of the hardware intensity. The solid line plots a typical energy-delay curve for some hardware function. Dotted lines show several contours of the cost function (Equation (10.2)), for two values of hardware intensity η. Point (D,E) at which the energy-delay curve tangents the lowest of the contours $F_{cost}(E,D) = A$ (with the smallest value of A) corresponds to the implementation for this value of hardware intensity η. Taking advantage of the equivalence of the tangents to the energy-delay curve and the counter of the cost function, we get:

$$\left.\frac{\partial E}{\partial D}\right|_{\substack{\text{fixed}\\ \text{n}}} = \frac{\partial E(\eta, v)}{\partial \eta} \Big/ \frac{\partial D(\eta, v)}{\partial \eta} = -\frac{\partial F_{cost}}{\partial D} \Big/ \frac{\partial F_{cost}}{\partial E} = -\eta \frac{E}{D} \tag{10.3}$$

This establishes the equivalence of the two definitions of the hardware intensity in Equation (10.1) and Equation (10.2).

Then, by formally solving the problem of minimizing the delay as a function of η and v, subject to a constant energy constraint, the following relations were derived in Zyuban and Strenski [21] for the optimal balance between hardware intensity and power supply voltage:

$$\text{isolated macro} \quad \eta = \theta(v) \tag{10.4}$$

$$\text{composite macro} \quad \frac{w_j}{u_j}\eta_j = \theta(v) \quad 1 \le j \le M \tag{10.5}$$

$$\text{multi-stage pipeline} \quad \sum_i w_i \eta_i = \theta(v) \tag{10.6}$$

where w_i are energy weights of pipeline stages i in Equation (10.6), w_j and u_j are energy and delay weights of sub-blocks j in Equation (10.5), η_j are the hardware intensities in the corresponding sub-blocks, and θ is the voltage intensity defined as

$$\theta = \frac{E_v}{D_v} \quad E_v = \frac{v}{E}\frac{\partial E}{\partial v} \quad D_v = -\frac{v}{D}\frac{\partial D}{\partial v}. \tag{10.7}$$

Equation (10.4) has a simple interpretation, shown in Figure 10.2. The solid curve shows an energy-delay trade-off curve for variable hardware intensity at a fixed power supply, $v = 1.5V$, $\theta = 2$ in this example (hardware intensity energy-delay curve). The curve was fitted to simulation data [21] for an integer adder, obtained using the EinsTuner [6]. The dotted curves show the energy-delay curves for a fixed tuning point (fixed hardware intensity η) of the circuit, but varying power supply, plotted for an ideal $E \sim v^2$ and $f \sim v$ dependence, as commonly assumed in many studies. The dashed curves show simulated data (for a set of functional units (FUs), running PathMill and PowerMill), with 50 mV steps in the power supply marked with circles. The point at which the power supply energy-delay curve (dashed curve) tangents the hardware intensity energy-delay curve (solid curve) corresponds to the optimal balance between η and v.

To show this, suppose that the circuit is over-tuned, for example, $\eta = 4$. Then, retuning the circuit for a lower value of η will move the design point down the hardware intensity energy-delay curve. Increasing the power supply to recover the performance will move the design up the power supply energy-delay curve (dashed curve). Because the hardware intensity energy-delay curve (solid curve) is steeper than the power supply energy-delay curve, the same performance will be achieved at a lower energy. Similarly, if the circuit is under-tuned for, say, $\eta = 0.5$, then tuning the circuit for a higher η and then reducing V_{dd} to achieve the same critical path delay (if the faster operation is not needed) will result in a circuit operating at the same speed, but lower energy. Notice that this reasoning does not require that the curvature of the hardware intensity energy-delay curve be higher than that of the power supply energy-delay curve. Although this property was experimentally verified for a 0.13 μ and older CMOS technologies it may or may not hold true in future technologies, depending on the dependence of the gate and subthreshold leakage currents on the power supply. If the curvature of the V_{dd} energy-delay curve becomes higher, the range in which energy and performance can be traded through adjusting V_{dd} will be more limited, but the optimality relation (Equation (10.4)) will still hold.

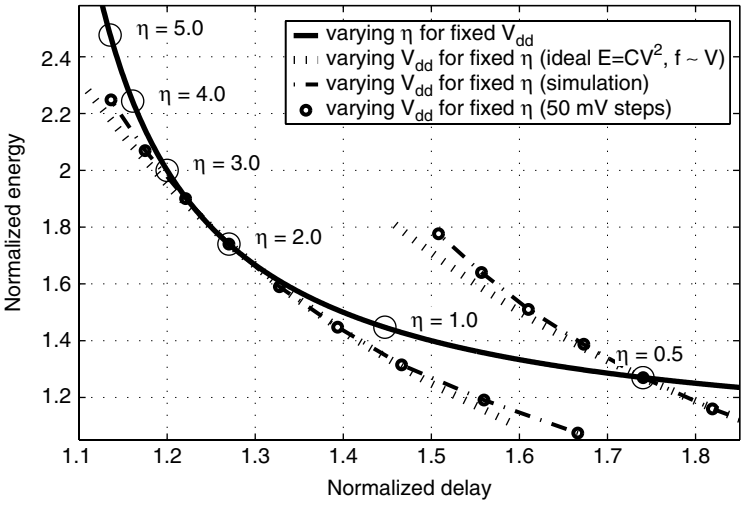


FIGURE 10.2 Graphical interpretation of the optimum hardware intensity balance for an isolated macro.

Although the optimal values of hardware intensities in different pipeline stages, η_i are different, it is useful to abstract for the higher-level microarchitectural analysis of energy-performance trade-offs a single aggregate quantity for hardware intensity η_{ag} that represents the whole processor, such that $\eta_{ag} = -\left.\frac{D\partial E}{E\partial D}\right|_v$, where D is the clock period, and E is the total average energy dissipated per cycle in the processor, $E = \sum E_i$. To derive an expression for η_{ag}, notice that increasing the clock cycle time by dD through retuning the circuits in all stages of the pipeline, increases the total energy of the pipeline by $dE = \sum dE_i = -\sum \frac{E_i}{D_i}\eta_i dD$, where the summation is performed over all stages of the pipeline. Since $D_i = D$ (all stages are tuned for the same delay), $\frac{dE}{E} = -\frac{dD}{D}\sum w_i\eta_i$, which means that the aggregate hardware intensity for a multi-stage pipeline is expressed through the hardware intensities of individual stages η_i as

$$\eta_{ag} = \sum_i w_i \eta_i \tag{10.8}$$

10.3 Architectural Complexity

Changing the processor architecture is another way to make trade-offs between performance and energy. Architectures that are more complex deliver higher architectural performance or instructions per cycle (IPC), but inevitably dissipate more energy per every executed instruction. Similar to building the optimal energy-delay curve in the circuit domain, an optimal energy-delay curve can be constructed in the architectural domain, as an envelope in the power-performance space of all feasible architectural alternatives [20]. Similar to Equation (10.1), the architectural complexity ξ can be defined as*

$$\xi = -\left.\frac{D\partial E}{E\partial D}\right|_{\text{fixed}\,\eta,v} \quad \text{or} \quad \xi = -\left.\frac{\%E}{\%D}\right|_{\text{through architecture}} \tag{10.9}$$

Similar to the optimal balance between η and v in the circuit domain, there exists an optimal balance between architectural complexity ξ and η and v in the unified architectural-circuit domain. By formally solving the problem of minimizing energy as a function of three variables $E = E(\xi, \eta, v)$, subject to a constant delay constraint $D(\xi, \eta, v) = D_0$, we arrive at**

$$\frac{\partial D}{\partial \eta}\frac{\partial E}{\partial v} = \frac{\partial D}{\partial v}\frac{\partial E}{\partial \eta} \quad \frac{\partial D}{\partial \xi}\frac{\partial E}{\partial v} = \frac{\partial D}{\partial v}\frac{\partial E}{\partial \xi} \tag{10.10}$$

Using Equation (10.1) and Equation (10.9), Equation (10.10) can be rewritten as

$$\xi = \eta = \theta \tag{10.11}$$

*It is assumed that the curvature of the architectural energy-delay curve is such that $\frac{D^2}{E}\frac{\partial^2 E}{\partial D^2} > \xi(\xi+1)$ is satisfied at every point.

**The converse problem of minimizing D subject to constant E leads to the same equations.

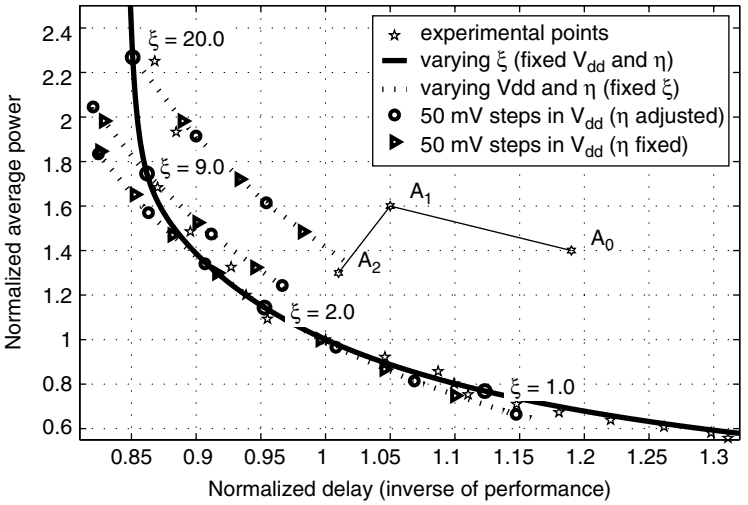


FIGURE 10.3 Graphical interpretation of the optimum architectural complexity balance.

Figure 10.3 gives a graphical interpretation of this relation. The solid curve shows the architectural energy-delay curve, plotted by curve-fitting the power-performance data reported in the optimal pipeline depth study [17]. Every data point, plotted as stars in Figure 10.3, represents a different pipeline depth, and we assume here that the processor microarchitecture was optimally tuned at every pipeline depth. The dotted curves in the figure show the circuit energy-delay curves for fixed ξ, with the power supply and hardware intensity varied simultaneously, so that the optimum balance (Equation (10.4)) is observed at each point. Circles mark 50-mV steps in V_{dd}, with the corresponding adjustment in η (Equation (10.6)). This data was obtained by simulating a set of representative circuits in a microprocessor [21]. For a reference, triangles show 50-mV steps in V_{dd} without adjusting η. Although the quality of the energy-delay trade-off of the fixed-η scaling is almost the same as that of the optimal v-η scaling, the span of the former is much smaller (larger change in V_{dd} is needed to achieve the same speed up or slow down).

The point at which the architectural energy-delay curve (solid curve) tangents the circuit energy-delay curve (dotted curve) is the point of the optimal balance between ξ, η, and v in Equation (10.11). To see this, suppose the architecture is over-designed (for instance, $\xi = 9$). Then, by reducing the architectural complexity, we can move the design point down the architectural energy-delay curve. Then the performance can be recovered by increasing V_{dd} (by 100 mV in this example) and tuning up circuits for higher η, according to Equation (10.11), which will move the design point up the circuit energy-delay curve. Because the circuit energy-delay curve is less steep than the architectural energy-delay curve, the same performance will be achieved at a lower power. Similarly, if $\xi < \eta$, for instance, $\xi = 1$, then increasing the architectural complexity to improve the architectural performance (moving up the architectural energy-delay curve) and reducing V_{dd} and η to save energy (moving down the circuit energy-delay curve) will result in the same performance at a lower power. As with hardware intensity, this relation is not dependent on assumptions about the relative curvatures of the various energy-delay trade-offs.

Although the nature of the energy-delay trade-offs at the architectural level is similar to that at the circuit level, one significant difference between them is that with recent advances in the circuit tuning techniques [6] all circuit-level implementations (provided an appropriate circuit topology is chosen) in a properly tuned processor can be assumed to be on the optimal energy-delay curve (Figure 10.2) (designs above the optimal energy-delay curve should be simply discarded), whereas getting the processor architecture that is on the architectural energy-delay curve (Figure 10.3) presents a significant challenge. The initial architectural proposal for a new processor is likely to be way off the optimal energy-delay curve. Multiple iterations of optimizing the architecture are required to transfer the design point to the optimal architectural energy-delay curve, and then, to the point of the optimum balance (Equation (10.11)), an

iterative process illustrated by sequence $A_0 \to A_1 \to \cdots A_n$ in Figure 10.3. To make the methodology useful for comparing architectural configurations that are not necessarily on the optimal energy-delay curve, the definition of ξ was extended to designs above the optimal energy-delay curve and a more general form of the energy-efficiency criterion was derived in Zyuban [19], and generalized to include hardware intensity in Zyuban and Strenski [21]:

$$\frac{\eta_{ag}}{I}\frac{\Delta I}{\Delta\xi} - \frac{\eta_{ag}+1}{N}\frac{\Delta N}{\Delta\xi} > -\frac{\eta_{ag}}{f}\frac{\Delta f}{\Delta\xi}\bigg|_{\text{fixed}\eta,v} + \frac{1}{E}\frac{\Delta E}{\Delta\xi}\bigg|_{\text{fixed}\eta,v} \tag{10.12}$$

If the inequality holds, then the architectural feature under evaluation is energy-efficient, that is, after adopting it, the processor will deliver higher net performance at the same power budget, after appropriate retuning and, possibly, adjustment in the power supply voltage are done to meet the power budget. In this formula $\frac{\Delta f}{f\Delta\xi}$, $\frac{\Delta I}{I\Delta\xi}$, $\frac{\Delta E}{E\Delta\xi}$, and $\frac{\Delta N}{N\Delta\xi}$ are relative increments in the processor frequency, architectural performance IPC, average energy per instruction, and the dynamic instruction count arising from a modification at the architectural or microarchitectural level, evaluated for a fixed hardware intensity η_{ag} and power supply v. Thus, all deltas in Equation (10.12) have the meaning of partial derivatives with respect to the architectural complexity.

The terms $\frac{\Delta I}{I\Delta\xi}$ and $\frac{\Delta N}{N\Delta\xi}$ in Equation (10.12) can be measured by running the benchmark suite on an architectural simulator. Next we present a methodology for estimating the two remaining terms, $\frac{\Delta f}{f\Delta\xi}$ and $\frac{\Delta E}{E\Delta\xi}$, and derive a new form of the energy-efficiency criterion that does not require estimating the term Δf.

10.4 Energy-Efficiency Criterion

10.4.1 Frequency-Invariant Formulation

The key assumption in deriving the energy-efficiency criterion (Equation (10.12)) was that of the optimal tuning of circuits in every pipeline stage (Equation (10.4), Equation (10.5), and Equation (10.6)) for every architectural alternative, so that the aggregate hardware intensity of the processor η_{ag} (Equation (10.8)) is unchanged between designs implementing the architectural alternatives. This assumption imposes special rules on calculating $\frac{\Delta f}{f}$, and $\frac{\Delta E}{E}$, in particular, these relative increments must be calculated assuming that the processor pipeline is reoptimized after every modification to the microarchitecture to satisfy Equation (10.4), Equation (10.5), and Equation (10.6).

Suppose, an architectural feature under evaluation introduces an additional complexity in several (or all) stages of the pipeline, which leads to increments $\Delta D_i\big|_{no\,retune}$ in critical path delays in the corresponding pipeline stages, assuming that no retuning is done to recover the clock frequency. Suppose that the corresponding increments in average energies are $\Delta E_i\big|_{no\,retune}$. The increments $\Delta D_i\big|_{no\,retune}$ and $\Delta E_i\big|_{no\,retune}$ should be evaluated consistently with the initial hardware intensities of the corresponding stages. For example, logic added to stage i should be tuned (or assumed to be tuned) according to Equation (10.5). Then after adding the logic, the aggregate hardware intensity in pipeline stage i will not change. The delay and energy increments may be either positive or negative, and in those pipeline stages that are

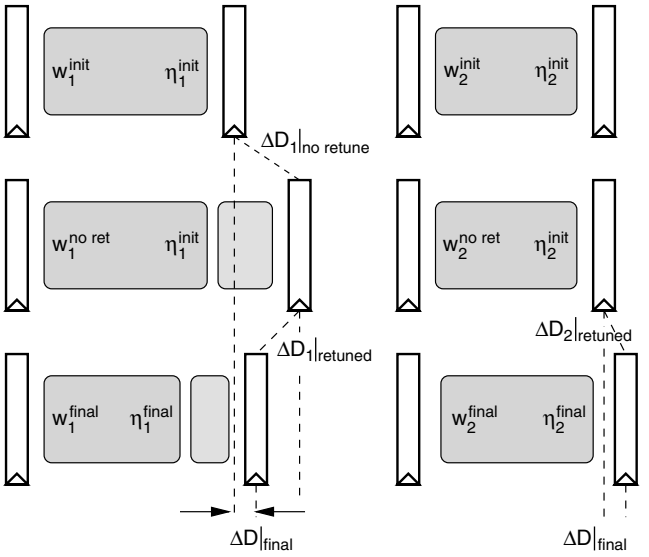


FIGURE 10.4 Retuning pipeline after architectural modification.

unaffected by the architectural modification, the delay and energy increments are zero, $\Delta D_i\big|_{no\,retune} = 0$, $\Delta E_i\big|_{no\,retune} = 0$, as shown in Figure 10.4.

Circuit designers usually have no difficulties estimating the "nonretuned" increments in delay and energy. For example, adding an execution bypass in 10FO4 pipeline results in increments in the critical path delay and average energy of the execution stage of the pipeline which are approximately $\left.\frac{\Delta D_{EX}}{D}\right|_{no\,retune} = 0.2$, and $\left.\frac{\Delta E_{EX}}{E_{EX}}\right|_{no\,retune} = 0.02$, whereas adding an extra read port to a multiported register file may result in $\left.\frac{\Delta D_{RF}}{D}\right|_{no\,retune} = 0.1$, and $\left.\frac{\Delta E_{RF}}{E_{RF}}\right|_{no\,retune} = 0.2$, with no impact in other stages of the pipeline.

To recover the clock frequency, circuits in those stages of the pipeline that are negatively affected by the architectural modification need to be tuned up for a higher hardware intensity. To restore the energy-optimal balance in the pipeline $\eta_{ag} = \theta$, circuits in all remaining stages need to be tuned down for a lower η, so that

$$\Delta\eta_{ag} = \sum_i \eta_i \Delta w_i + \sum_i w_i \Delta\eta_i = 0 \tag{10.13}$$

where $\Delta\eta_i$ is the increment in the aggregate hardware intensity in stage i as a result of retuning, $\Delta\eta_i = \eta_i^{final} - \eta_i^{initial}$, as illustrated in Figure 10.4, whereas Δw_i is the net increment in the corresponding energy weight, as a result of both adding hardware and subsequent retuning, $\Delta w_i = \frac{\Delta E_i}{E} - w_i \frac{\Delta E}{E}$.

We designate by $\Delta D_i\big|_{retune}$ and $\Delta E_i\big|_{retune}$ the increments in delay and energy in the pipeline stage i as a result of retuning the processor, whereas by $\Delta D_i = \Delta D$ and ΔE_i we designate the net increment in delay and energy in pipeline stage i as a result of both modifying the function and subsequent retuning:

$$\Delta D = \Delta D_i\big|_{no\,retune} + \Delta D_i\big|_{retune} \tag{10.14}$$

$$\Delta E_i = \Delta E_i\big|_{no\,retune} + \Delta E_i\big|_{retune} \tag{10.15}$$

Thus, the net delay and energy increments in every pipeline stage consist of increments due to a change in the functionality resulting from a microarchitectural modification, and additional increments as a result of retuning the circuits. The net delay increment ΔD does not need any index because all pipeline stages are assumed to have the same delay before and after the retuning, $D_i = D$. The relative increment in the maximum clock frequency is related to ΔD as

$$\left.\frac{\Delta f}{f}\right|_{\text{fixed }\eta, v} = -\frac{\Delta D}{D} \tag{10.16}$$

Assuming small changes in hardware intensities in all pipeline stages, and neglecting second order terms, the increments in energies $\Delta E_i\big|_{retune}$ because of the retuning can be expressed through the corresponding increments in delays $\Delta D_i\big|_{retune}$ as follows:

$$\Delta E_i\big|_{retune} = -\eta_i \frac{E_i}{D} \Delta D_i\big|_{retune} \tag{10.17}$$

Using Equation (10.14) and Equation (10.15), the final increments in energies can be expressed as

$$\Delta E_i = \Delta E_i\big|_{no\,retune} - \eta_i \frac{E_i}{D}\left(\Delta D - \Delta D_i\big|_{no\,retune}\right) \tag{10.18}$$

The total increment in energy of the whole pipeline, $\Delta E = \sum \Delta E_i$, is calculated by summing Equation (10.18) over all pipeline stages and taking advantage of Equation (10.8) and Equation (10.16):

$$\left.\frac{\Delta E}{E}\right|_{\text{fixed }\eta, v} = \left.\frac{\Delta E}{E}\right|_{no\,retune} + \sum_i \eta_i w_i \left.\frac{\Delta D_i}{D}\right|_{no\,retune} + \eta_{ag} \left.\frac{\Delta f}{f}\right|_{\text{fixed }\eta, v} \tag{10.19}$$

Substituting this expression into the earlier derived energy-efficiency criterion (Equation (10.12)), we notice that the term $\Delta f / f$ cancels out, since in both expressions it has the same meaning of a partial derivative with respect to architectural complexity ξ. Then, dropping $\Delta\xi$ in the denominators of all terms we arrive at the form of the energy-efficiency criterion that does not require estimating the increment in frequency:

$$\eta_{ag} \frac{\Delta I}{I} - (\eta_{ag} + 1) \frac{\Delta N}{N} > \left.\frac{\Delta E}{E}\right|_{no\,retune} + \sum_i \eta_i w_i \left.\frac{\Delta D_i}{D}\right|_{no\,retune} \tag{10.20}$$

where $\left.\frac{\Delta E}{E}\right|_{no\,retune}$ is the total increase in average energy dissipated per instruction, assuming no retuning, $\left.\frac{\Delta E}{E}\right|_{no\,retune} = \sum \left.\frac{\Delta E_i}{E}\right|_{no\,retune}$, summation being done over all stages in the pipeline, affected by the architectural modification.

Equation (10.20) is a more convenient form of the energy-efficiency criterion than Equation (10.12). According to Equation (10.20), to evaluate the energy-efficiency of an architectural feature, the architects must supply the relative gain (or loss) in the architectural performance $\frac{\Delta I}{I}$ and relative change in the

dynamic instruction count $\frac{\Delta N}{N}$ that result from this feature. These estimates can be obtained by running an architectural simulator, or timer, like Turandot [11,12]. The second term, ΔN is nonzero if changes to the instruction set architecture (ISA) are considered, or compiler optimizations are analyzed for energy efficiency. It may also be nonzero if microarchitectural changes are considered in a speculative issue processor that impact the average number of instructions executed from mispredicted paths.

The architect needs to consult circuit designers to estimate the impact of the architectural feature under consideration on the average energy dissipated per instruction and the critical path delay through every stage of the pipeline affected by this architectural feature. A significant advantage of the derived formula is that in estimating the relative changes in energy and critical path delays the circuit designer does not need to worry about retuning the circuits to recover the frequency, or reducing the positive timing slack to save power in logic on paths that are no longer critical. Then, the relative increments in critical path delays are summed, multiplied by the appropriate energy weights and hardware intensities. The higher the energy weight w_i and the hardware intensity η_i of a part of the pipeline i affected by the architectural feature the higher the weight of the increase in the critical path delay through this part of the pipeline.

The energy weights w_i in Equation (10.20) are typically available as part of power budgeting at the early stages of the definition of the processor pipeline. The only additional data that is needed to use the energy-efficiency criterion are hardware intensities η_i in all blocks of the processor. Those quantities can be measured by static tuning tools, like the EinsTuner [6], based on the simulations of previous designs, or set as targets at early planning of the microarchitecture, similar to the way the power targets are budgeted.

Then Equation (10.20) is evaluated. If the inequality holds, then the architectural feature under evaluation is energy-efficient, that is, after adopting it, the processor will deliver higher net performance at the same power budget, after appropriate retuning and, possibly, adjustment in the power supply voltage are done to meet the power budget.

Figure 10.5 gives a graphical interpretation of the architectural energy-efficiency criterion (Equation (10.20)). Modifying the architecture from an alternative A to B is evaluated for energy efficiency

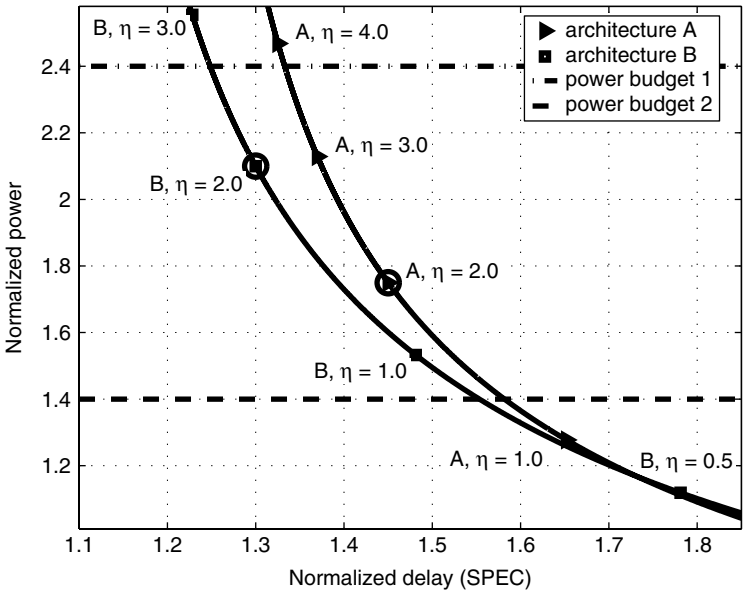


FIGURE 10.5 Graphical interpretation of the energy-efficient criterion.

using Equation (10.20). The points corresponding to the implementations of both architectural alternatives with the same hardware intensity $\eta = 2$ are marked with circles, and the curves passing through these circles show implementations of architectural alternatives *A* and *B* with different hardware intensities, assuming that the power supply is adjusted accordingly, to keep the optimum balance between the hardware intensity and power supply (Equation (10.6)). In other words, these curves show where the corresponding design points will move in the energy-delay space if the same architectures are implemented with more or less aggressive circuits.

If the inequality in Equation (10.20) evaluates as true for the architectural change from alternative *A* to *B*, then in the neighborhood of these points, the circuit energy-delay curve passing through point *B* is below (or left of) the circuit energy-delay curve passing through point *A*, as shown in Figure 10.5. This means that for any power budget, within a certain range of the initial points, implementations of architecture *B*, deliver higher performance than implementations of architecture *A*, for the same power budget.

Notice that the curves may intersect, as shown in Figure 10.5, which means that architectural alternative *B* is more energy efficient than *A* only within some range of the initial design points. This demonstrates the fact that an architecture optimized for a certain range in the power-performance space may not perform well outside of this range. For example, a high-performance core, scaled down to operate in the lower performance space may not be competitive with a core specifically optimized for the low-power, low-performance applications. Another conclusion from this analysis is that an accurate estimate of the available power budget for a core is essential for developing an energy-efficient architecture because only by knowing the power budget can we estimate the maximum value for architectural complexity, hardware intensity, and power supply that can be used in the core.

Figure 10.3 plots a possible outcome of overestimating the power budget available for a processor core at the microarchitecture definition stage, and choosing an overly high value for the architectural complexity. Suppose, at early design stages, the power budget is estimated to be at 2.2 and the microarchitecture of the processor core is optimized for the architectural complexity of $\xi = 20$. Suppose that at a circuit phase of the design it is discovered that the actual power budget is only 1.2. Since changing the architecture at this point would result in missing the product release schedule, the only way to bring the processor power under the budget is to redesign all circuits for a lower hardware intensity and reduce the power supply, or just reduce the power supply, if it is too late for redesigning the circuits. This will send the design point down the dashed curve passing through the point $\xi = 20$ in Figure 10.3, and leading to an almost 15% loss in performance compared to the design originally optimized for the power budget of 1.2 with the architectural complexity of $\xi = 2.0$. Thus, such late changes in the design may lead to a significant performance degradation, and scaling down the power supply to bring an overpowered processor core to the power budget may have a very high performance cost. This demonstrates the importance of accurately estimating the available power budget at early design stages, and disproves the notion, common in architectural community, that relative power estimates are always sufficient when proposing new architectural features.

10.5 Other Power–Performance Metrics

Until energy efficiency criterion (Equation (10.20)) was introduced, the most popular power-performance metric used in the architectural community has been [2–5,7,8,13,15,18]

$$\frac{MIPS^{\gamma}}{Watt} \tag{10.21}$$

with the value of parameter γ ranging from $\gamma = 0$ to $\gamma = 3$, depending on the class of the microprocessor. As discussed in Zyuban [19], the prior art metric (Equation (10.21)) is a special case of the integral form of the derived metric (Equation (10.12)), with γ set to $\gamma = \eta_{ag} + 1$.

Another recent work proposed the following metric for evaluating architectural features [16]:

$$3\frac{\Delta IPC}{IPC} > \frac{\Delta Power}{Power} \tag{10.22}$$

Notice that Equation (10.22) is also a special case of Equation (10.12), with $\eta_{ag} = 2$, $\Delta f = 0$ and $\Delta N = 0$, because in clock-gated designs $Power \sim E \times IPC$, and $\frac{\Delta Power}{Power} = \frac{\Delta E}{E} + \frac{\Delta IPC}{IPC}$.

The main advantage of the derived criterion (Equation (10.20)) is that, in addition to being formally derived and being more general than the preceding metrics (Equation (10.21) and Equation (10.22)), all its terms have a clear meaning and an unambiguous method for estimating them as "naive" increments in energies and delays. On the other hand, the metrics in Equation (10.21) and Equation (10.22), though correct, may be confusing to use by an architect, because they hide important assumptions about the method for estimating increments in million instructions per second (MIPS) and Watts. In particular, estimating the term $\Delta Power$ may be ambiguous because it requires knowing both Δf and ΔE, which are interrelated and depend on the assumptions about the allowed change in the clock frequency and retuning the pipeline after modifying the architecture. When using these metrics, some architects assume that circuit designers will do whatever is needed to recover the extra delay due to an introduced architectural feature, and set $\Delta f = 0$, neglecting the increase in energy due to redesigning and retuning the circuits. Others calculate the extra delay introduced due to an added architectural feature and set $\frac{\Delta f}{f} = -\frac{\Delta D}{D}$, assuming that nothing can be done at the circuit level to recover the frequency, and neglecting that circuits in stages not affected by the change will have a timing slack and could be tuned down to save power. In both cases, the conclusion of applying the metrics in Equation (10.21) and Equation (10.22) may be incorrect. The next section presents a typical example of incorrectly using the metrics in Equation (10.21) and Equation (10.22).

10.6 Example: Adding an Execution Bypass

As an example, we evaluate the energy-efficiency of implementing an execution bypass in the integer unit (IU) of a microprocessor with a target cycle time of 10FO4, and an aggregate hardware intensity target of $\eta_{ag} = 2$ (shown in Figure 10.6). This microarchitectural feature affects only the register file (RF)

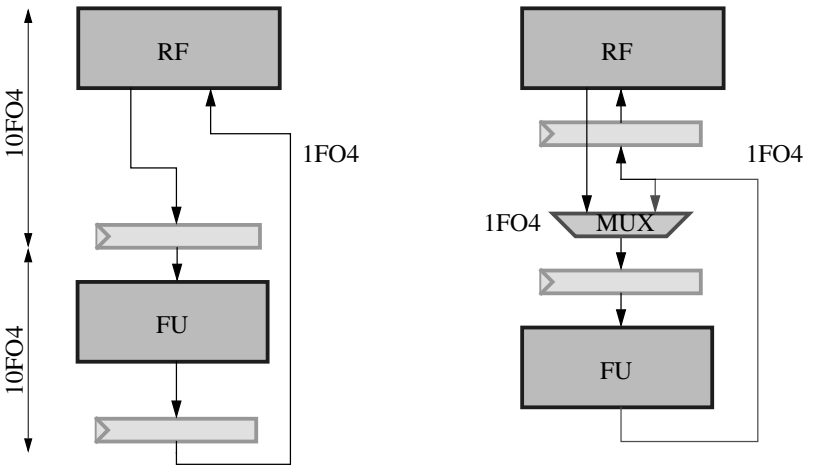


FIGURE 10.6

and execution (EX) stages of the processor. The delay insertion penalty of the bypass multiplexer in front of the latch is approximately 1FO4, and the delay of the bypass wire, including the rebuffering is approximately 1FO4. Then the relative "nonretuned" increments in the critical path delays through the register file and execution stages are $\left.\frac{\Delta D_{RF}}{D}\right|_{noretune} = 0.1$ and $\left.\frac{\Delta D_{EX}}{D}\right|_{noretune} = 0.2$.

Adding the bypass multiplexer and the bypass wires also introduces an energy overhead, dissipated whenever the IU is accessed. Based on simulation results, we estimate the relative energy overhead of the bypass wires and multiplexers as 5% of the average energy dissipated in the IU. Suppose the energy budget of the IU is 10% of the total energy dissipated by the microprocessor. Then, $\left.\frac{\Delta E}{E}\right|_{noretune} = 0.05 \cdot 0.1 = 0.005$.

Suppose, the aggregate hardware intensity of the microprocessor is $\eta_{ag} = 2.0$, but since pipelining the register file access and integer functional units has a high cost in IPC degradation, a higher value of hardware intensity is budgeted to them, $\eta_{RF} = \eta_{EX} = 3$. Also, suppose, $w_{RF} = 0.04$ and $w_{EX} = 0.06$. Then, using the criterion in Equation (10.20), we determine the relative increment in IPC that needs to be demonstrated to justify adding the execution bypass in the IU as $\frac{\Delta I}{I} > 2.7\%$.

Notice that if we used the $\frac{MIPS^3}{Watt}$ metric then, depending on the assumption about the change in frequency, the architect could arrive at different conclusions. If the view is taken that the circuit designers can do nothing to recover the frequency, then the metric in Equation (10.21) leads to the answer $\frac{\Delta I}{I} > 20\%$. On the other hand, if the view is taken that circuit designer will do whatever is needed to recover the frequency, and the architect does not need to worry about it, then the metric in Equation (10.21) leads to the answer $\frac{\Delta I}{I} > 0.25\%$. Similarly, the metric in Equation (10.22) leads to $\frac{\Delta I}{I} > 0.25\%$, assuming the frequency is unchanged. In both cases, the conclusions about the energy-efficiency of the IU execution bypass produced by straightforwardly applying the metrics in Equation (10.21) and Equation (10.22) are incorrect.

10.7 Conclusions

This chapter analyzed common approaches to trading power and performance in the design of processor cores for systems on chip, such as varying the power supply, hardware intensity, and architectural complexity. It was demonstrated that in order to develop an energy-efficient processor core, that is a core that delivers maximum performance at a strictly limited power budget, design decisions at all levels must be balanced in such a way that all forms of spending power have a similar marginal cost. A criterion for optimizing the core architecture was described which is useful for guiding the iterative architectural optimization process that leads to the optimal balance between the architectural complexity, hardware intensity and power supply. It was demonstrated that a single core may not be competitive in both high and low performance domains, and accurate estimates of the available power budget for a core are essential for developing an energy-efficient architecture, as opposed to making decisions based on relative power estimates only. It was also demonstrated that scaling down the power supply to bring an overpowered processor core to under the power budget might have a very high performance cost.

10.8 Acknowledgment

The authors thank Dr. Jaime Moreno and Dr. Kevin Warren for their management support.

References

[1] R. Brodersen, M. Horowitz, D. Markovic, B. Nikolic, and V. Stojanovic. Methods for true power minimization. In *Proc. ICCAD,* 35–42, November 2002.

[2] D. Brooks and P. Bose et al. Power-aware microarchecture: design and modeling challenges for next-generation microprocessors. *IEEE MICRO,* 20(6):26–44, November 2000.

[3] T. Burn and R. Broderson. Energy-efficient CMOS microprocessor design. *Proc. 28th Annu. Hawaii Int. Conf. on System Sciences,* 288–297, 1995.

[4] J. Burr and A. Peterson. Energy considerations in multchip module-based multiprocessors. *Proc. ICCD,* 593–600, 1991.

[5] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power CMOS digital design. *IEEE J. Solid-State Circuits,* 27(4):473–484.

[6] A. Conn et al. Gradient-based optimization of custom circuits using a static-timing formulation. *Proc. Design Automation Conf.,* 452–459, June 1999.

[7] R. Gonzales, B. Gordon, and M. Horowitz. Supply and threshold voltage scaling for low-power CMOS. *IEEE J. Solid-State Circuits,* 32(8):1210–1216, August 1997.

[8] R. Gonzales and M. Horowitz. Energy dissipation in general-purpose microprocessors. *IEEE J. Solid-State Circuits,* 31(9):1277–1283, September 1996.

[9] H.P. Hofstee. Power-constrained microprocessor design. In *Proc. IEEE on Computer Design,* 14–16, September 2002.

[10] J. Moreno et al. An innovative low-power, high-performance programmable signal processor for digital communications. *IBM J. Res. Dev.,* 47(2/3):299–327, 2003.

[11] M. Moudgill, P. Bose, and J.H. Moreno. Validation of Turnadot, a fast processor model for microarchitecture exploration. *Proc. IEEE Int. Performance, Computing, and Communications Conf. (IPCCC),* 451–457, February 1999.

[12] M. Moudgill, J.D. Wellman, and J.H. Moreno. Environment of PowerPC microarchitecture exploration. *IEEE Micro,* 19(3):9–14, May/June 1999.

[13] K. Nowka, P. Hofstee, and G. Carpenter. Accurate power efficiency metrics and their application to voltage scalable CMOS VLSI design, *IEEE Trans. on VLSI Syst.,* 2003.

[14] V.G. Oklobdija, B.R. Zeydel, D. Hoang, S. Mathew, and R. Krishnamurthy. Energy-delay estimation technique for high-performance microprocessor VLS1 adders. In *Proc. 16th IEEE Symp. on Computer Arithmetic,* 2003.

[15] P. Penzes and A. Martin. Energy-delay efficiency of VLSI computations. *Proc. Great Lakes Symp. on VLSI,* 104–107, April 2002.

[16] J. Rattner. Making the right-hand turn to power-efficient computing. Keynote speech, *35th Annu. Int. Symp. on Microarchitecture,* November 2002.

[17] V. Srinivasan et al. Optimizing pipelines for power and performance. *Proc. 35th Annu. Int. Symp. on Microarchitecture,* November 2002.

[18] M. Stan. Low-power CMOS with subvolt supply voltages. *IEEE Trans. on VLSI Syst.,* 9(2):394–400, April 2001.

[19] V. Zyuban. Unified architecture level energy-efficiency metric. *Proc. Great Lakes Symp. on VLSI,* 24–29, April 2002.

[20] V. Zyuban and P. Kogge. Optimization of high-performance superscaler architectures for energy efficiency. *IEEE Symp. on Low-Power Electron. and Design,* 84–89, August 2000.

[21] V. Zyuban and P. Strenski. Unified methodology for resolving power-performance trade-offs at the microarchitectural and circuit levels. *Proc. Int. Symp. on Low-Power Electron. and Design,* 166–171, August 2002.

11

Low-Power SoC with Power-Aware Operating Systems Generation

Sungjoo Yoo
Aimen Bouchhima
Wander Cesario
Ahmed A. Jerraya
TIMA Laboratory

Lovic Gauthier
FLEETS

11.1 Introduction

Recently, embedded software (SW) is becoming increasingly important in system-on-chip (SoC) design [1]. Two important characteristics of embedded SW in SoC are:

1. Interaction with physical components in hardware (via I/O and interrupt)
2. Concurrency required to handle the interaction with physical components as well as to better exploit the processor computing power

To satisfy the characteristics, embedded SW needs the operating system (OS) both for the hardware (HW) interaction and for multi-tasking.

Embedded SoCs have strict constraints in energy consumption. The main drains of energy in SoC are processor, memory, on-chip bus, to name a few. From the viewpoint of embedded SW, the energy consumption is decomposed into two parts: energy consumption by the application SW and by the operating system.

For the reduction in energy consumption by the application SW, there have been presented many techniques categorized into dynamic voltage scaling [2] and dynamic power management [3,4]. Recent analyses and experiments have demonstrated that the actual OS can consume a significant portion of energy [5,6].

To reduce the energy consumption of SoC, we need methods to reduce the energy consumption of the OS as well as methods to reduce that of the application SW. To reduce the energy consumption of the OS, we can consider two approaches. One is to change the usage style of OS (e.g., replacing polling

operations into interrupts). The other is to reduce the energy overhead of OS itself (e.g., minimizing the OS size).

In this chapter, we handle the problem of minimizing the energy overhead of OS itself. The sources of energy consumption of the OS are twofold. One is the energy consumption of memory portion that contains the OS code. The other is the energy consumption of processor where the OS executes. To reduce the energy consumption of both sources, a possible method will be to design a small OS specific to the given application SW. The small OS implements only the functionality necessary to the application SW. We call such an OS application-specific OS.

The application-specific OS reduces the energy consumption of the memory by reducing the OS code size. To understand the effects of small OS code, for instance, assume a network-on-chip processor that contains 100 processors [7] and that each processor needs an OS with a code size of 100 KB. In that case, only for the OS code, we need 10 MB of memory and the energy consumption in the memory occupied by the OS code can be significant. Because a small OS usually enables fast OS execution, it can also reduce the energy consumption of processor by reducing the number of processor clock cycles necessary to perform the same OS functions.

The main difficulty in designing application-specific OSs is the design time. For the design, we need to tailor the functionality of OS to that of application SW. For instance, if the application SW does not use semaphore, the OS needs to remove the semaphore functionality from its implementation. Because many different OS functionalities (e.g., scheduling, inter-process communication, I/O, and interrupt management) and different implementations of the same OS functionality exist (e.g., different implementations of interrupt management), if the tailoring process is performed manually, application-specific OS design can be prohibitively time-consuming. Such a long design time of application-specific OS is not acceptable in ever tightening time-to-market pressure. Thus, we need new methods to accelerate the design of application-specific OS. In this chapter, we present a novel method to automatically generate application-specific OSs. The presented method generates very small OSs comparable to the smallest hand-written commercial OSs.

This chapter is organized as follows. Section 11.2 presents a short review of energy consumption issues related to the OS. Section 11.3 introduces SoC architecture. Section 11.4 explains the presented method. Section 11.5 discusses the effectiveness of the presented method with experiments. Section 11.6 concludes this chapter.

11.2 Related Work

Although most of SoC applications require low-power OSs, applications requiring ultra low-power OS are sensor network [8] and network on chip [7,9]. In Hill [8], an OS called TinyOS is presented for the sensor network where each sensor node has significant OS usage for data acquisition and transmission. In network on chip where each of (up to) hundreds of processors may need its OS, the energy overhead of OS can become significant.

In Dick et al. [5], an analysis of energy consumption by the OS (uC/OS-II in this case) is presented. The analysis demonstrates that we can obtain a 27.5 to 42.8% reduction in total energy by changing RTOS usage (e.g., replacing application code using polling by interrupt code). In Tan et al. [6], a technique of transforming SW architecture is presented to reduce the energy consumption of OS (ARM Linux in this case). The transformation gives a 10.7 to 66.1% reduction in total energy consumption. In this method, for the reduction in energy consumption, the OS code does not change, but the application code changes; however, our method changes (i.e., minimize) the actual OS code to reduce the OS overhead. Thus, the method in Tan et al. [6] and ours complement each other.

The size of the smallest versions of full-featured commercial OS (e.g., micro-kernel + file system and memory management) ranges between 80 KB (VxWorks [10]) and 100 KB (QNX Realtime [11]). Several very small OSs (i.e., micro-kernel designs including TinyOS [8] [~3.7 KB], pOSEK [12] [~2 KB], Chorus OS [13] [~10 KB], Ariel [14] [~19 KB], etc.) have been presented. In terms of OS size, the presented method yields OS sizes, 1.6 to 7.7 KB, comparable to the handwritten micro-kernels for our examples.

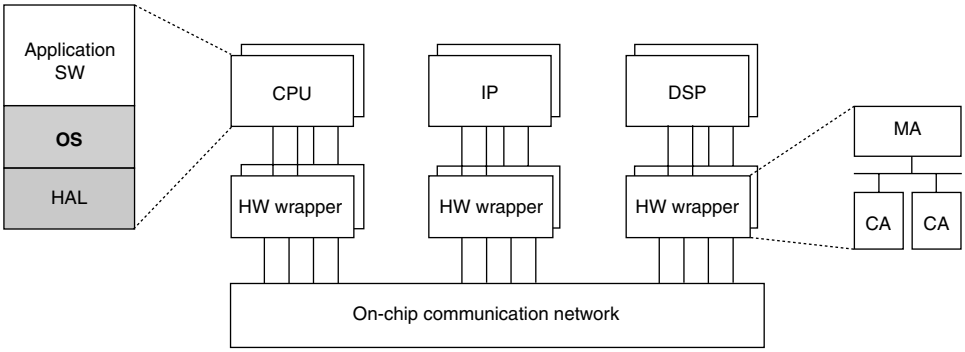


FIGURE 11.1 SoC architecture.

Compared with the existing micro-kernels, our contribution is to automatically generate small OSs without manual design.

11.3 Preliminary: SoC Architecture Generation

11.3.1 SoC Architecture

Figure 11.1 depicts a typical SoC architecture with heterogeneous processors (i.e., CPUs, DSPs, IPs, etc.) and the on-chip communication network (e.g., AMBA and packet/circuit switch network).

Embedded SW is generally organized as a stack of layers on top of the processor as depicted in Figure 11.1. The lowest SW layer called HW abstraction layer (HAL) provides the upper SW layer with an abstraction of underlying HW architecture. The OS provides the application SW with services such as task scheduling and synchronization, interrupt management, I/O, memory management, etc.

The processor is connected with on-chip communication network via an interface called HW wrapper. This chapter focuses on the OS generation in the SoC architecture. For further details of HW wrapper generation, refer to Lyonnard et al. [15] and Cesario et al. [16].

11.3.2 SoC Architecture Generation

The application-specific OS is generated as a part of SoC architecture generation. Figure 11.2 illustrates the SoC architecture generation flow called ROSES. From a high-level SoC specification called VADEL, the flow generates application-specific OS and HW wrappers.

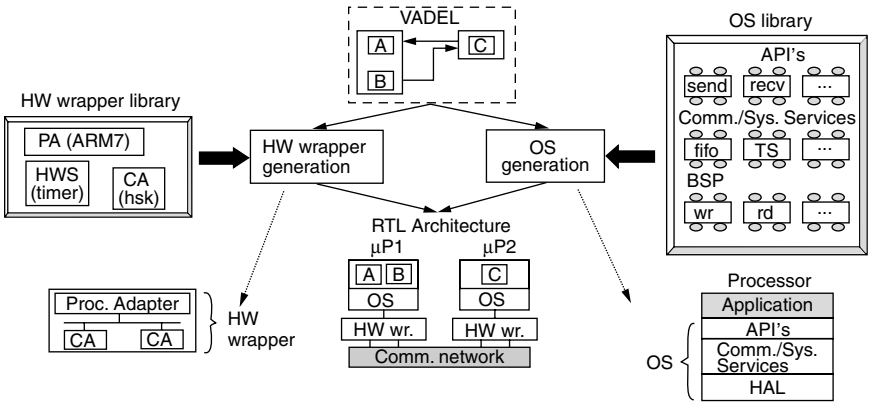


FIGURE 11.2 The flow of SoC architecture generation: ROSES.

In VADEL, the system is described with a network of hierarchical modules. Modules are connected with each other through its ports via communication channels. A module consists of behavioral parts and ports. Each module can be a leaf module or a hierarchical one composed of a network of module instances. We call a leaf module a *task*. For communication, ports provide module behavior with communication application programming interface (API) encapsulating communication details (i.e., communication protocol). In the real SoC implementation, the OS implements the API (illustrated in the OS library in Figure 11.2) in the form of service. Note that the module behavior (i.e., application SW code) communicates with the OS via the APIs. In fact, the API is a programming model used by the designer to write the application SW code. In our case, the API is extendable.

Architecture generation from the VADEL representation to SoC architecture consists of:

1. OS generation
2. HW wrapper generation

As illustrated in Figure 11.3, an OS consists of two types of services: communication/system services and HAL services. To generate application-specific OSs, only the services required by the application SW are selected from the OS library. Assembling HW components from the HW library, as illustrated in Figure 11.2, also generates the HW wrapper. In this paper, we focus on the OS generation flow. For the details of HW wrapper generation, refer to Lyonnard et al. [15] and Cesario et al. [16].

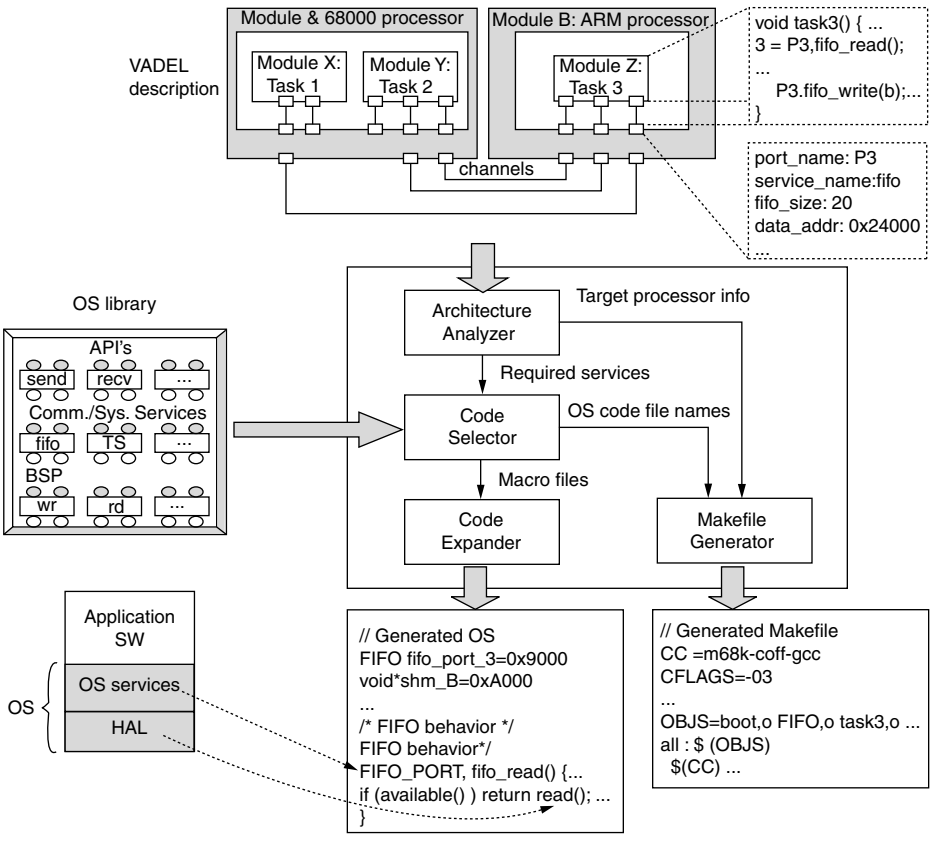


FIGURE 11.3 OS generation flow.

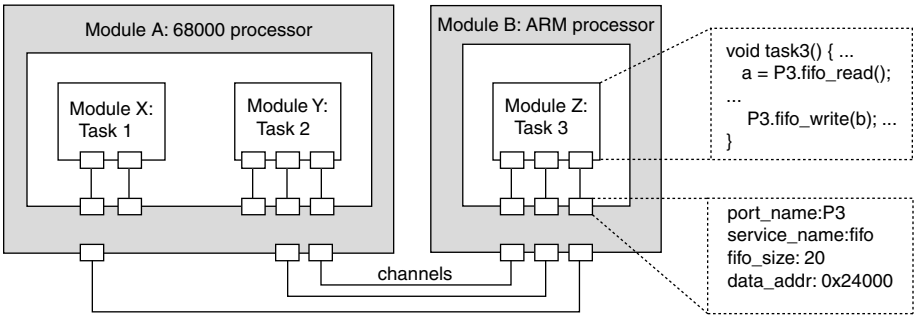


FIGURE 11.4 An example of VADEL description.

11.4 Automatic Generation of Application-Specific Operating Systems

Figure 11.3 depicts the flow of automatic OS generation. The input to the OS generation is the VADEL description that contains a network of hierarchical modules, communication between modules and parameters for OS generation.

As presented in the figure, the architecture analyzer takes the structural information and the parameters in the VADEL description. The code selector receives a list of services specific to the application and the architecture from the architecture analyzer and finds the full list of (original and deduced) services. The code expander generates the source code of OS. Then, the makefile generator gives makefiles. Thus, the outputs of the design flow are the source code of the generated OS and a makefile for each processor. To obtain the binary code to be downloaded onto the target processor memory, the designer runs a compilation of both generated OS and the application SW code using the generated makefile.

11.4.1 System Description Input

As the system description input, the flow takes a structural representation of communication in a hierarchical network of modules. Figure 11.4 is an example of hierarchical network of modules.

In the figure, two modules (tasks) X and Y are mapped on a 68,000 processor and the other module Z on an ARM7 processor. Each module (task or processor) has a set of ports (depicted as small rectangles in the figure). The port has parameters. Figure 11.4 is an example of port parameters (e.g., *port_name (P3), service_name (fifo),* and so forth). Each task has also parameters such as task priority.

Note that the OS is generated on a processor basis. Between modules, inter/intraprocessor communications are represented. For instance, there is an intraprocessor communication between modules X and Y (a line in the shaded region of the figure). In the figure, the three channels between module A and B exemplify interprocessor communication.

Figure 11.4 is an example of communication via the port in task 3. In the figure, the behavioral part of task 3 just calls a high-level communication function provided by port P3 (i.e., a port function), for example, *P3.fifo_read()* for communication via the port. In the VADEL description, we do not describe the behavior part of task, but list task source code files and the parameters of OS API that the task calls for.

11.4.2 OS Library

The OS library is the core of the OS generation tool. It provides small and flexible pieces of code that can be adapted and used to build the OS. The library contains two parts:

1. A code part containing the code
2. A description part

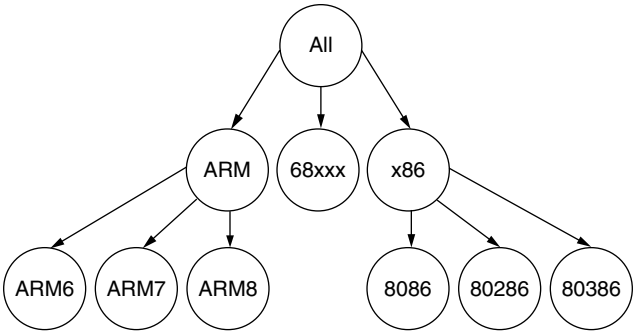


FIGURE 11.5 Processor dependency of OS code.

The code part is written in a macro language. The description part enables assembly of OS code pieces. Its main components are the OS element, the service, and implementations.

11.4.2.1 OS Element

An OS element represents a SW code piece of the generated OS. It provides services and may require other services. Depending on different target processors, an OS element can have several code pieces each of which is specific to a processor. We call such a processor-specific code piece an implementation. Note that each implementation code of an element can be described in a different language (for instance, C or assembly).

11.4.2.2 Service

A service can represent any kind of OS functionality. In our OS library, three types of services are used: communication, system, and HAL services. Communication services implement the API functions of communication between modules. For instance, the communication protocol FIFO is implemented as a communication service. Examples of system service are task scheduling service (e.g., function to schedule tasks in preemptive/nonpreemptive priority-based/round-robin schedulers) and timing service. Communication/system services use device driver services to implement their functionality. *HAL* services are specific to the processors where the application SW is running.

11.4.2.3 Implementations

Implementations of an OS element are organized in a tree relationship. An implementation can also be compliant with specific peripherals (mainly for driver descriptions).

Figure 11.5 is an example of such a tree relationship. In the figure, each oval represents a part of implementation of an OS element. For instance, an OS element, boot for ARM7 processor, can consist of:

1. A processor-independent code piece in C (the top oval in the figure)
2. ARM processor family-specific code piece in assembly (the oval denoted with ARM)
3. ARM7 processor-specific code piece in assembly

11.4.3 OS Code Generation

11.4.3.1 Architecture Analyzer

The architecture analyzer finds the following information from the system description input:

1. Application-specific services and their detailed parameters
2. Module-specific parameters
3. Modules interconnection topology

Application-specific OS services are extracted from the parameters of modules, channels, and ports in the system description input. For instance, if a port has a parameter for FIFO implementation, FIFO service is selected to be included into the generated OS. Further detailed parameters of required services

are also found in the VADEL description. For instance, the address range of FIFO communication and the interrupt level of interrupt-driven port can be found. A list of required service names is sent to the code selector.

Module-specific parameters (e.g., task priority, processor speed, and processor type) are also found from module parameters. The type of processor is sent to the makefile generator to choose the right compiler and to the code expander to produce the OS code to the processor. The interconnection topology (e.g., point to point and multipoint) is used to deduce internal data structure used for communication and synchronization.

11.4.3.2 Code Selector

The code selector takes as input the list of required service names from the architecture analyzer. It uses the OS library to check service dependencies and finds all the OS elements that have dependency relation with the required services and that are compliant with the target architecture. The dependency relationship can be transitive. For instance, if a FIFO communication service used in the application code requires an interrupt handling service, an OS element providing the interrupt handling service should be also chosen for inclusion in the OS to be generated.

An OS element is compliant with an architecture if one of its implementations is compliant with the architecture and if all the services it requires can be provided by other compliant OS element. Because an OS element may require some services, the previous algorithm is repeated recursively to perform a kind of transitive closure. Sometimes, several OS elements compliant with the architecture may provide the same required service. In such a case, it is up to the user to choose the good one. After the OS element selection is done, the code selector sends the list of the code file names to the makefile generator and the macro file names to the code expander.

11.4.3.3 Code Expander

The code expander takes as input a list of macro file names from the code selector and parameters from the architecture analyzer. It generates the final OS code by:

1. Associating the right parameters to each OS element
2. Expanding the macro codes of OS elements to source codes by calling an external macro processing program

Figure 11.6 is an example of code expansion. In Figure 11.6(a), a macro-code section is listed for a part of the semaphore synchronization mechanism. First, the code expander fixes the input parameters: *SYNC*, which is an array of identifiers for the synchronization units and *Sem_P*, which indicates that the service function is required. Then, it performs the macro expansion. For instance, in Figure 11.6(b), the data structure for the semaphores is generated with its size (*SIZE* = 2) (i.e., the number of semaphores, its queues, and its states [zero]). Note that the synchronization queue uses another service (in fact, also to be expanded) *Soft_Wait* that provides wait and signal mechanism.

11.4.3.4 Makefile Generator

The makefile generator takes as input:

1. Processor type information from the architecture analyzer
2. A list of source codes of the OS (in C and assembly) from the code selector
3. A list of the application SW codes described in the VADEL description

It determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW.

11.4.4 Application to Existing OSs

An existing OS can be integrated into the proposed flow of automatic generation (to be specific, in this case, automatic configuration) of application-specific OS. To explain the integration, we assume that the

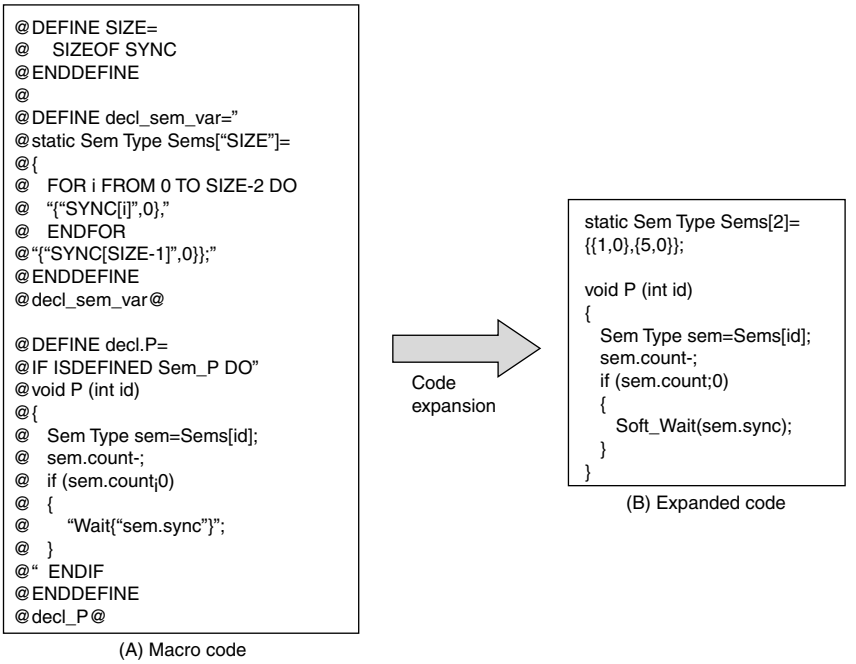


FIGURE 11.6 Code generation example.

existing OS supports OS configuration by "#" statements (i.e., configuration by defining required macros) without modifying the OS source code because most commercial OSs allow such a configuration. The integration can be done as follows:

1. Information of available OS services (e.g., API, macros defined for services) and dependency relationship between services in the existing OS are taken into the OS library.
2. To the OS generation flow in Figure 11.3, the designer gives the same system description input in VADEL.
3. The architecture analyzer performs the same operation (i.e., extracting required information such as services and target processor information) as described in this section.
4. The code selector finds all the required (derived) services from the OS library as explained in this section. Then, it selects, from the OS library, macro definitions corresponding to the required services instead of selecting macro code files as explained in this section. Note that in the case of automatic configuration of existing OS, the code expander does not generate the OS source code because the existing OS source code is not modified.
5. The makefile generator outputs a makefile with the selected macro definitions received from the code selector.

Note that compared with the original flow of automatic OS generation proposed in this chapter, in the case of integrating the existing OS into the flow, no change occurs in automatic execution of service extraction (by the architecture analyzer) and makefile generation. The code quality (i.e., size and execution time) of the automatically configured OS depends on the configuration granularity of OS services in the existing OS.

11.5 Experiments

We applied the proposed method to two system examples: a token-ring system and a very high data-rate digital subscriber line (VDSL) framer system.

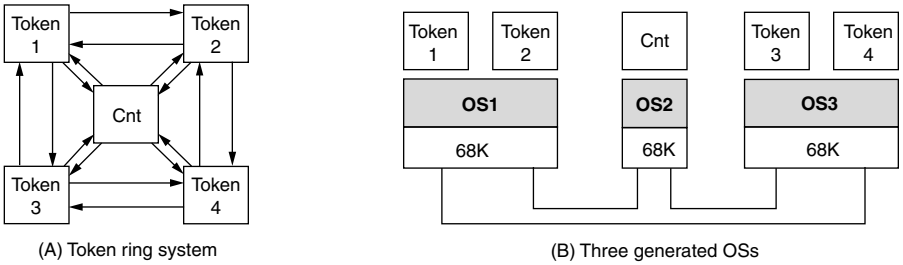


FIGURE 11.7 A token ring example.

11.5.1 Token-Ring Example

Token-ring system (1245 lines in SystemC) consists of four tasks (called *token*) that exchange tokens with each other and one counter task (called *Cnt*) that counts the number of tokens exchanged.

Figure 11.7(a) depicts the interconnection of tasks in the example. As presented in the figure, four *token* tasks make a bidirectional ring connection with each other. *Cnt* task is connected to all *token* tasks.

In our experiment, we implement the system example on a multi-processor target architecture with three 68 K processors depicted in Figure 11.7(b). In the figure, four *token* tasks are mapped to two processors (two tasks on each processor) and *Cnt* task is mapped to the other processor. In the system description input, we assigned equal priority to all the tasks.

For the communication channel between modules, we specified one word communication with non-blocking write/blocking read. In the example, the size of transferred data (i.e., counter value and token) is one word.

First, the system description input in VADEL is read into the architecture analyzer. Then, the code selector selects the following OS services for the two OSs (OS1 and OS3 in the figure) of the two processors each of which runs two *token* tasks:

1. Round-robin scheduler service with preemption service because tasks have the same priority
2. A timer service because the round-robin scheduler with preemption is used
3. Nonblocking write (called *exoutd*) and blocking read services (called *exinb*)

The code expander generates the OS source code that handles two tasks of equal priority and two communication service functions (*exoutd* and *exinb*). For the processor where only *Cnt* task is mapped, the same communication services are selected for OS2; however, no scheduler and timer services are selected because only one task exists on the processor.

We obtained three binary executables for three processors after running compilation with the generated OS codes and makefiles. We validated the system implementation in cosimulation with three instruction set simulators of 68,000 processor and a VHDL simulator. As the result, in this experiment, the generated OSs give very small code sizes: 797 lines (90% in C and 10% in assembly), compiled and linked to 1.86 KB for each of two processors with two Token tasks and 1.62 KB for the processor with one *Cnt* task. In terms of performance, it gives 83 instruction cycle latency in the channel read operation from interrupt trigger to the end of single-word data access.

11.5.2 VDSL Example

The design presented in this section was taken from the implementation of a VDSL modem using discrete components [17]. The block diagram for this prototype implementation is illustrated in Figure 11.8. The subset we will use in the rest of this paper is shaded in the figure. It is composed of two ARM7 processors and a part of the datapath, *TX_Framer,* described at the RT-level.

On the two processors, parallel tasks are running. The control over the three modules (two processors and *Tx_Framer*) is fully distributed. All three modules act as masters when interacting with their envi-

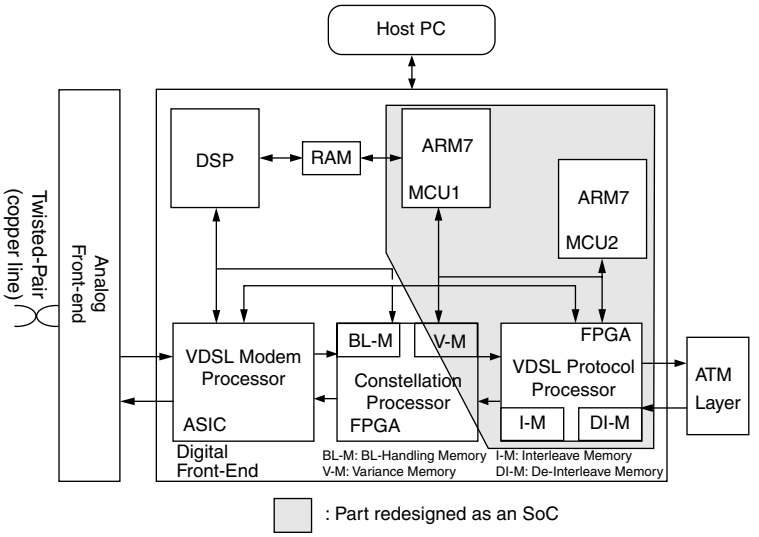


FIGURE 11.8 Entire VDSL system.

ronment. Additionally, the application includes some multipoint communication channels (shared memory services) requiring sophisticated OS services.

Figure 11.9 illustrates the VADEL description of VDSL example. Modules **VM1** and **VM2** correspond to the ARM7s on Figure 11.8 and module **VM3** represents *TX_Framer* block.

Given the VADEL description, our flow in Figure 11.2 generates two OSs for two ARM7 processors and three HW wrappers (i.e., two for two ARM7 processors and one for the Tx Framer). The architecture generation takes only a few minutes on a Linux PC 500 MHz. Figure 11.10 depicts the RTL architecture obtained after OS and HW wrapper generation. For further details of HW wrapper generation, refer to Cesario et al. [16].

Each generated OS is customized to the set of tasks executed on each of the processors. For example, SW tasks running on **VM1** access the OS using an API composed of two functions: pipe for communication with **VM2**, and signal to modify the task scheduling on runtime. The OS contains a round-robin

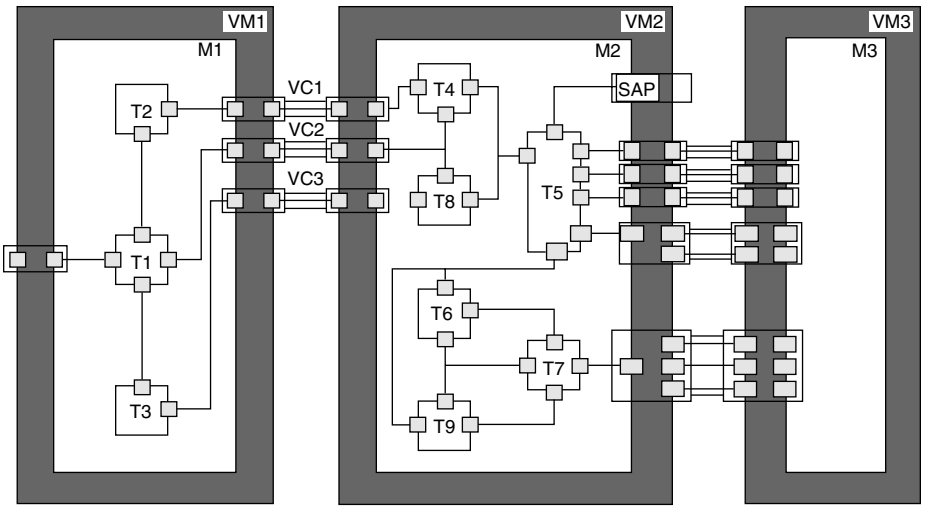


FIGURE 11.9 VADEL description of VDSL example.

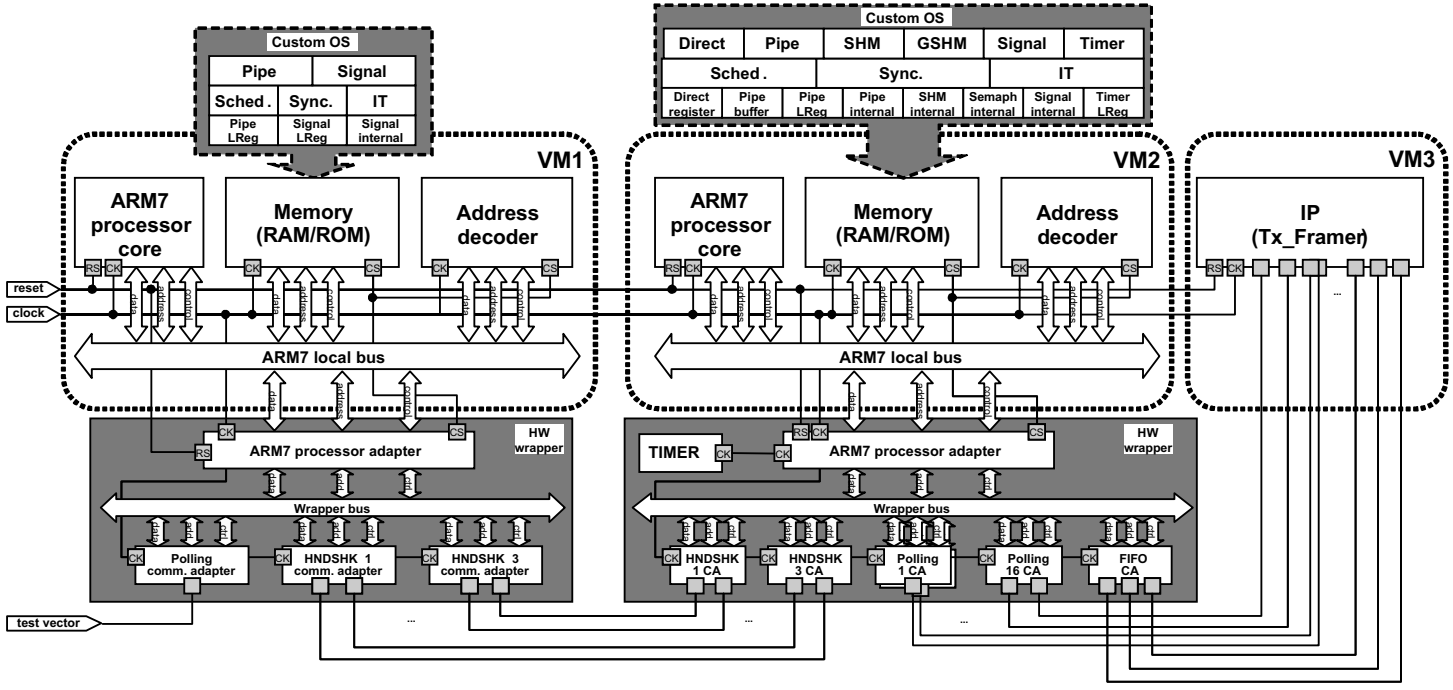


FIGURE 11.10 Generated SoC architecture.

TABLE 11.1 Results for OS Generation

OS Results	No. of Lines C	No. of Lines Assembly	Code Size (bytes)	Data Size (bytes)
VM1	968	281	3,829	500
VM2	1,872	281	6,684	1,020
Context switch (cycles)				36
Latency for interrupt treatment (cycles)				59 (OS) + 28 (ARM7)
System call latency (cycles)				50
Resume of task execution (cycles)				26

scheduler (*Sched*) and resource management services (*Sync, IT*). The HAL contains low-level code to access the HW (e.g., *Pipe LReg* to access *HNDSHK* CA in the HW wrapper as presented in Figure 11.10) and some low-level kernel routines.

The HW wrapper for processor **VM2** includes a TIMER block because task *T5* (see Figure 11.9) must wait 10 ms before starting its execution. When the timer expires, the TIMER block generates an HW interrupt. The task can configure this block using Timer API provided by the OS. The OS for **VM2** provides a more complex API. *Direct* API is used to write/read to/from the configuration/status registers inside *TX_Framer* block; *SHM* and *GSHM* are used to manage shared-memory communication between tasks.

Application code and generated OS (in C and assembly) are compiled and linked together to execute on each ARM7 processor. The HW wrapper can be synthesized using RTL synthesis. Table 11.1 presents the results regarding the generated OSs.

The OS sizes (4.3KB for VM1 and 7.7KB for VM2) presented in Table 11.1 are comparable to the sizes of commercial micro-kernels (2 to 10 KB). In terms of OS runtime, as listed in the table, context-switch takes 36 cycles, latency for the HW interrupt is 59 cycles (plus 4 to 28 cycles needed by the ARM7 to react), latency for system calls is 50 cycles, and task reactivation takes 26 cycles.

11.5.3 Gain Compared with Conventional OSs

In terms of OS size, our results (1.6 to 7.7 KB) presented in this section are comparable to the smallest micro-kernels (2 to 10 KB as explained in Section 11.2). Compared with the conventional small OSs, our contribution is to automatically generate small OSs thereby enabling to explore the OS design space, especially to obtain low-power OS implementations.

11.6 Conclusion

This chapter presented the problem of reducing the energy consumption by the OS. To minimize the energy overhead causes by the OS, we presented a method to design application-specific OSs with sizes that are minimal enough to provide the OS services required by the application SW. Because the application-specific OS reduces the overhead of memory usage and processor runtime, it can reduce the energy consumption caused by the OS itself. To resolve the important problem of application-specific OS design (i.e., slow design cycle) our method generates automatically application-specific OSs. The experiments demonstrate that the automatically generated OSs yield OS sizes (1.6 to 7.7 KB) comparable to those of commercial handwritten micro-kernels (2 to 10 KB). Fast design of small-size OSs will enable the exploration of the low-power implementations of OSs.

References

[1] Semiconductor Industry Association, *ITRS 2001 Edition,* available at http://public.itrs.net/Files/2001ITRS/Home.htm.

[2] P. Pillai and K. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, *Symp. on Operating Syst. Principles,* Oct. 2001.

[3] C. Pereira, V. Raghunathan, S. Gupta, R. Gupta, and M. Srivastava. An SW architecture for building power-aware real-time operating systems, Technical Report No. 02-07, University of California–Irvine, March 2002.

[4] Y.H. Lu, et al. Operating system directed power reduction, *Int. Symp. on Low-Power Electron. and Design,* 2000.

[5] R.P. Dick, G. Lakshiminarayana, A. Raghunathan, and N.K. Jha, Power analysis of embedded operating systems, *Design Automation Conf.,* June 2000.

[6] T.K. Tan, A. Raghunathan, and N.K. Jha, SW architectural transformations: a new approach to low-power embedded SW, *Design, Automation, and Test in Europe (DATE),* March 2003.

[7] L. Benini and G. De Micheli, Networks on chips: a new SoC paradigm, *IEEE Comput.,* 35(1), pp. 70–80, 2002.

[8] J. Hill. An SW architecture supporting networked sensors. Master's thesis, EECS, University of California–Berkeley, Dec. 2000.

[9] E. Rijpkema, K.G.W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip, *Design Automation and Test Conf. in Europe,* March 2003.

[10] VxWorks, available at http://www.windriver.com/products/vxworks5/index.html.

[11] D. Hildebrand, An architectural overview of QNX, available at http://www.qnx.com/literature/whitepapers/archoverview.html.

[12] pOSEK, A super-small, scalable real-time operating system for high-volume, deeply embedded applications, available at http://www.isi.com/products/posek/index.htm.

[13] Chorus operating system open source, available at http://www.experimentalstuff.com/Technologies/ChorusOS/index.html.

[14] Microware ariel technical overview, http://www.microware.com/ProductsServices/Technologies/ariel_technology_brief.html.

[15] D. Lyonnard, S. Yoo, A. Baghdadi, and A.A. Jerraya, Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip, *Design Automation Conf.,* 2001.

[16] W.O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, and A.A. Jerraya, Multiprocessor SoC platforms: a component-based design approach, *IEEE Design and Test of Comput.,* Vol. 19, No. 6, Nov.–Dec., 2002.

[17] M. Diaz-Nava and G.S. Okvist, The zipper prototype: a complete and flexible VDSL multi-carrier solution, *J. ST Microelectronics,* Sept. 2001.

12

Low-Power Data Storage and Communication for SoC

Miguel Miranda
Erik Brockmeyer
Tycho van Meeuwen
Cedric Ghez
IMEC

Francky Catthoor
IMEC and Katholiek University

12.1 Introduction

Despite the recent architectural advances for multimedia aiming at improving computational efficiency (e.g., subword parallel data level processing, reconfigurable computing, and networks on chip), the dominance in data storage and transfer of these systems still remains as one of the main bottlenecks for power and speed efficient implementations. The reason is the ever-increasing gap in speed and energy between the memory and the data processing subsystems.

To cope with such a gap, the addition of more layers to the memory hierarchy, from where data can be efficiently accessed from smaller, faster, and more energy efficient memories becomes mandatory. This is true both for systems on chip (SoC) platforms based on random access memories as well as for cache memories; however, the potential efficiency offered by these multi-layer memory organizations becomes attainable on condition that the storage and transfer of data between the different layers is done in an optimal manner.

Memory hierarchy layers can contain software controlled scratch-pad memories or caches. To guarantee an efficient transfer of data along the memory hierarchy, often requires that smaller copies of the data be made from the larger data arrays, which can be stored in the smaller layers [1,2]. Those copies must be selected such that they minimize the overall transfer cost. In this context, any transfer of data from a higher layer to the current one is considered an overhead for the current layer.

This happens most efficiently under full software control (e.g., a number of static random access memories (SRAM) used as a scratch-pad memories) because a global view on the transfer can be obtained at design time. In this case, copy operations should be explicitly present in the application code. This is mostly possible for design time analyzable applications that are characterized via Pareto curves collecting all optimal energy trade-offs for the different execution times (see Section 12.3). The decision of the selected Pareto point can also be made at design time for purely static applications.

Many real-life applications are dynamic in nature, however, and they cannot be completely characterized at design time. Traditionally, to cope with this dynamism, HW-controlled caches are used instead of SW-controlled scratch-pad memories. In this case, the hardware cache controller will make the copies of signals at the moment they are accessed (and the copy is not present yet in the cache); however, this is inefficient because data present in the cache and required in the near future can be (wrongly) evicted to accommodate new fetched data. This evicted data, when needed again by the processor, will have to be brought to the cache for a second time, thus leading to transfer and power overhead. To minimize such overhead, architects tend to use bigger caches with hardware controllers implementing complex mapping policies; however, this is not efficient for power given the extra overhead in every single access even when these are cache hits. According to Wilton and Jouppi [3], a 1-KByte (KB), four-way associative cache is 4 to 5 times more inefficient in terms of energy per access than a scratch-pad memory of the same size and 2 to 3 times more than a (one-way associative) direct mapped cache (DM-cache); the latter is only 60% less efficient than a scratch-pad memory of the same size. This overhead is due to accessing the tag array of the cache which size (thus energy overhead) considerably increases with the associativity factor.

For dynamic applications, a much better approach is to characterize the different run-time conditions of the application at design time and to generate for each a Pareto curve as demonstrated in Marchal et al. [4]. This is quite design time-consuming, but it can be sped up by appropriate tools (see, e.g., Section 12.3 and Marchal et al. [4]). In this case, the selection of the Pareto operation point will be made at run-time, which can be done very efficiently in the middleware layer and not at design-time, as is the fully static case, although this still allows employing SRAMs. When the application becomes even more data-dependent, however, the number of Pareto curves to be stored would become too high. In that case, the HW controller of a cache can still help to obtain better run-time results, starting from the code with copy candidates which are "locked" in the cache whenever appropriate (Pareto) design time analysis is available. The cache will then implement the right copy of data according to the selected Pareto point, but the HW controller takes over whenever no appropriate locking choice is available.

In this approach, it is crucial that we use energy-efficient cache memories with simple controller mechanisms and low tag overhead that are still SW-controlled (such as the use of address-lock mechanisms, by-passable transfers and SW-controlled write-backs). An example of such cache memories is a lockable DM-cache. DM-caches are very attractive for low energy operation but if not well steered from the SW, they can lead to an excess in misses, thus in power. Fortunately, an effective way to reduce this overhead in misses can be achieved by a careful placement of the data in the main memory. Nevertheless, this is only useful on condition that the additional overhead involved in this placement is kept limited. Therefore, if good design time analysis and decisions are available, the code can be written such that the controller is forced to make the right decision [5]. For this purpose, we have developed an array interleaving data-layout approach that involves a relatively small amount of overhead in required storage size. This approach is very efficient in reducing data-layout related conflict misses, which are known to be the major contribution in DM-caches. This scheme interleaves signals in the main memory to assign different signals to own locations in the cache. This method can be complemented with an array padding technique for fine-tuning the placement of signals in the cache.

12.2 Related Work

Optimizing the memory hierarchy for performance using software controlled memory layers has been a well explored topic [6–9]; however, several recent articles also address the energy-related issues [10–13], and Benini and Micheli [10] and Panda et al. [12] have published good surveys.

In Grun et al. [14], a clustering of the data sets into different memory types is proposed. These different clusters have certain memory type preferences and are assigned accordingly; however, the mapping is suboptimal, especially for the regular accesses, because it is based on average characteristics and does not allow accurate predictions. Only then, performance is measured by simulation. In addition, Panda et al. [15] make a distinction between caches and scratch-pad memories; however, no real layer assignment is made. For that purpose, the technique presented in Steinke et al. [16] assigns data (and instructions) to the scratch-pad memory; however, no consideration is made to benefit from data reuse [1,2] and memory inplace optimizations [19].

The closest work to our approach is presented in Kandemir and Choudhary [13]. It analyzes and exploits the temporal locality by inserting local copies. Their layer assignment builds a separate hierarchy per loop nests and then combines them into a single hierarchy; however, a global view of the assignment and lifetime of the arrays and copies is required for real life applications having imperfect nested loops. Moreover, no overhead estimation is made, which makes it impossible to trade-off copy sizes (see Section 12.3) vs. array sizes in a certain layer. Similarly, the work published Masselos et al. [17] lacks a global application view.

On the other hand, access trace based analysis techniques [18] have limited optimization capabilities. The quality of the analysis depends on the preceding compilation step. For instance, from an access profile point of view, all elements of an array are accessed equally while a small data reuse copy could be present. As a result, the search space cannot be explored properly.

To the best of our knowledge, no previous work has combined data reuse and inplace opportunities in a systematic way, leading to a technique for real life applications that is not based on simulation. Our approach allows finding the optimal assignment in a predictable way for both memories and hardware caches. Moreover, it decides the trade-offs in a controlled manner instead of evaluating them afterward by adding the relevant estimations.

For HW-controlled cache memories, source-level program transformations have been proposed to modify the execution order. This can greatly improve the cache performance of these applications [19–21], but still a significant number of cache misses are present in the experimental results. To remedy this, loop blocking has been first proposed primarily for improving the cache performance [22], but we observe that for multimedia applications, a significant amount of cache misses remain due to conflict misses. Similarly, storage order optimizations [19,23] are very helpful in reducing the capacity misses. Thus, mostly conflict cache misses related to the suboptimal data-layout remain. Array padding had been proposed earlier to reduce conflict misses [24–26]. These approaches are useful for reducing the inter-array conflict misses to some extent; however, existing approaches do not eliminate the majority of the conflict misses yet. Besides the studies conducted by Kandemir et al. [20], Panda and Dutt [25], and Burger et al. [27], very little has been done to measure the impact of data-layout transformations on the cache performance. Thus, there is a need to investigate additional data-layout organization techniques to reduce these cache misses [5].

12.3 SW-Controlled Memory Hierarchy Optimization

By exploiting data reuse information [1], a part of an array can be copied from a higher to a lower layer from where it is read multiple times. As a result, energy can be saved by ensuring that most accesses take place on the smaller copy signal and not on the larger (potentially also bigger, thus more energy consuming) original array.

Many different opportunities exist for deciding a data reuse copy. These are called copy candidates (CCs). Only when it has been decided to instantiate a CC do we call it a copy (although the copy candidates will be instantiated as arrays in the application, we reserve the name array for the "original array"). A relation exists between the size of a CC and the number of transfers from the higher layer to the local (potentially smaller) layer needed to update that one with new data. These transfers, hereafter called copy writes (CWs) are for cache memories the equivalent to minimal capacity misses according to our compiler centric cache miss classification (see Section 12.4). Read operations from the CCs are called copy read operations (CRs). Write operations needed to initialize the original arrays (e.g., with data coming from the application's test-bench) will be called compulsory writes.

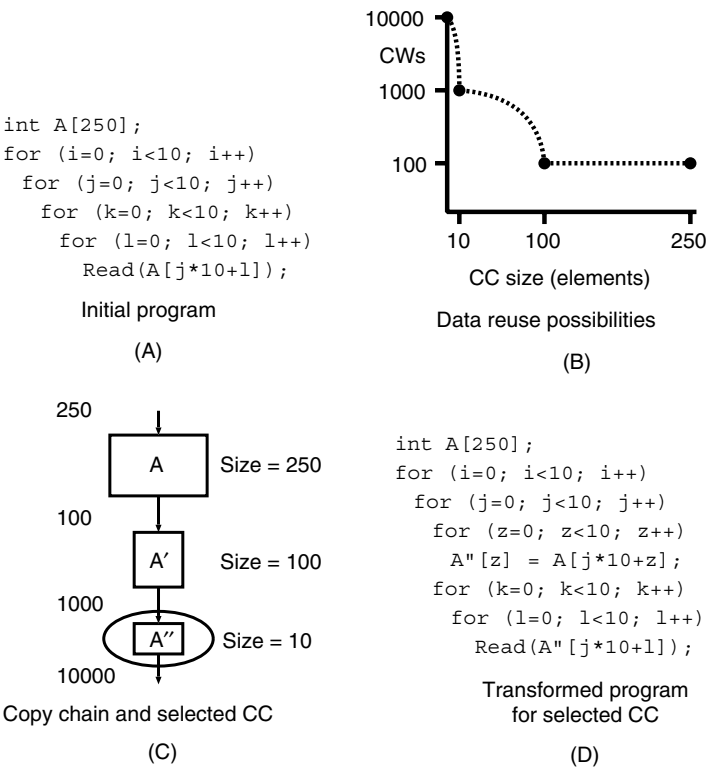


FIGURE 12.1 Analysis of the temporal locality of data: data reuse behavior.

Figure 12.1 presents an illustrative example of a loop nest with one reference to an array A with size 250. Originally, that array is accessed 10,000 times and several CCs are possible as depicted in Figure 12.1(B). For instance, we could select a CC A″ of size 10 by transforming the source code as illustrated in Figure 12.1(D) code. This is done by adding an explicit CC and corresponding copy statement in front of the k-loop. This copy statement will be executed 100 times, resulting in 1000 CWs to the array A. This CC selection corresponds to the second leftmost point plotted in Figure 12.1(B). Note that the good temporal locality of array A does not influence the amount of CWs from the next level. In theory, any CC size ranging from one element to the full array size is a potential candidate. In practice, however, only a limited set of CCs lead to efficient solutions. All possible CCs are plotted in Figure 12.1(B). The most promising CC sizes and CWs are kept and added to a data reuse chain as depicted in Figure 12.1(C). In this case, these are exactly those that have a relation to the loop bounds. That data reuse chain is completed with all 250 compulsory writes to array A.

The preceding example considers only a one array with a single read operation. In practice, however, several arrays exist in the code, each with one or more read operations, thus a data reuse chain will be associated to each read operation. To globally incorporate these issues, all data reuse chains of a given array must be combined in a data reuse tree. This concept is illustrated in the upper left part of Figure 12.2, where both data reuse trees for array A and array B are given. Array B is assumed to have only one read and it has no tree but a single chain associated. On the other hand, array A is assumed to have two references where one of them has no promising CC (indicated by the leftmost branch of the tree). More details on identification of data reuse chains and trees can be found in Achteren et al. [1] and Kandemir and Choudhary [13].

Obviously, larger CCs could retain the data longer and could therefore avoid more CWs. On the other hand, the larger the CC the bigger would be the memory required to hold the data and thus the bigger the overhead in energy per access will also be for each CR operation. Thus, a careful balance is required here between the number of CR and CW operations, and the layer implementation and associated energy

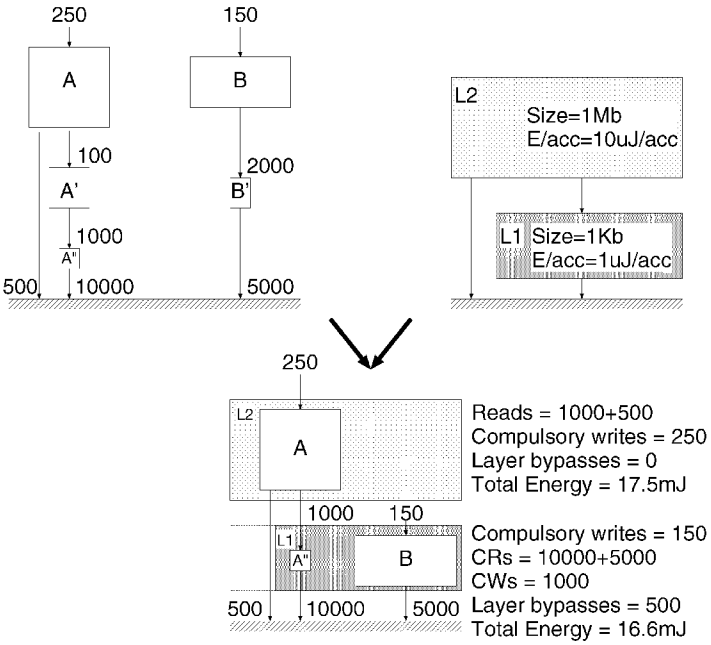


FIGURE 12.2 MHLA problem definition.

per access. This exploration is actually considered in the next step, when CCs and arrays must be mapped to the data memory hierarchy.

12.3.1 Memory Hierarchy Layer Assignment Techniques

We consider a generic SoC memory hierarchy template. It may contain multiple memory layers *Li*, each layer with multiple memory partitions (e.g., software controlled SRAMs, cache memories or a mixture of both) and off-chip (synchronous) dynamic random access memories ((S)DRAMs). Still, each partition can be organized on multiple memory modules, all of the same type but potentially having different sizes and number of ports.

In general, the energy consumed by the data memory hierarchy can be optimized for a target template by a careful selection of the set of CCs and arrays of the application, and by assigning these to the different layers of the memory hierarchy. This memory hierarchy layer assignment (MHLA) step results on an energy-efficient mapping of the different signals of the application to the memory hierarchy, and it is dependent on the different platform memory parameters. The goal is to select the mapping selection with the lowest energy possible where the total energy is the sum of the energy of all memory partitions.

The energy associated to each memory partition can be estimated by the number of accesses to this, multiplied by the energy per access of the partition (which can be potentially different depending on whether this is a read or write operation). On the one hand, the energy per access is a function of the memory size, partition type, and other memory parameters; thus it can be easily modeled in a memory library [23]. On the other hand, the number of accesses is the sum of compulsory writes, CRs, and CWs operations. This cost function is accurate in case of scratch-pad memories. For cache memories, the overhead in miss count should in principle be incorporated as well. As we will discuss in Section 12.4, however, that cost function is still valid for DM-caches on condition that additional optimizing memory data-layout transformations are applied to the application, a posteriori in a subsequent step. This is because the most important miss contributions are already being incorporated in the actual access count, namely the compulsory and minimal capacity misses that are by definition equivalent to compulsory writes and copy writes operations, respectively. For DM-caches, the major miss type contribution would

be coming from data-layout conflict misses where the overhead of these in the overall access count can be kept relatively low by the use of optimizing data-layout transformations [5] (see Section 12.4).

12.3.2 Illustration of the MHLA Techniques

The MHLA process and its mapping results are depicted in Figure 12.2. We assume CC A″ and array B are selected to be stored in L1 and array A in L2. This is a valid selection because we assume it fulfills the memory layer size constraints. Assuming this, on one hand we have for layer L2, 250 compulsory writes needed for the first initialization of array A. On the other hand, 500 reads are needed for reading A via the read operation having no promising CC associated (see upper left of Figure 12.2) and 1000 CRs needed for the update of A″ in layer L1.

Similarly in layer L1, 150 compulsory writes are needed for the initialization of array B, and 1000 CWs for updating CC A″. On the other hand, 10,000 CRs from CC A″ and 5000 CRs from array B will take place from the same layer. Note that the 500 reads from array A not having promising CC will not affect the activity of layer L1 for this architecture because a bypass from layer L2 is foreseen. This may not be the case for hardware controlled caches because all accesses have to pass through the cache when no bypass is foreseen. Also note that for caches no explicit copies are introduced in the code; however, the cache controller can be enforced to make the desired copy by a proper memory layout (see Section 12.4).

The assignment of arrays and CCs to the layers of the memory hierarchy must be performed globally to effectively minimize the energy consumption. The size of one copy must be traded for the size of another copy because the assignment must fulfill the maximum layer size constraints or because activity in that layer must be kept low for energy reduction.

A simple illustrative example of the MHLA balance having multiple references is given in Figure 12.3 for an example assuming only two read operations. The two curves at the left of the figure illustrate the amount of CWs for different CC of increasing sizes, thus it depicts the Pareto trade-off between CWs and CC-size. Only the evolution of CWs with CC-size is given because, in all cases, the number of CRs from the layer will not change. Array B has a maximum of 6000 CWs when no CC is selected and this can be reduced to 500 CWs when the largest CC is selected for layer assignment. Similarly, array A has more than 2000 CWs when the smallest CC is selected and less than 1000 when the largest CC is selected. The number of write operations to the layer is minimal when selecting the largest CC for both arrays; however, this assignment is not valid because the total required size for accommodating both CCs does not meet the maximum layer size constraint pictured at the right-hand side of Figure 12.3. There, all

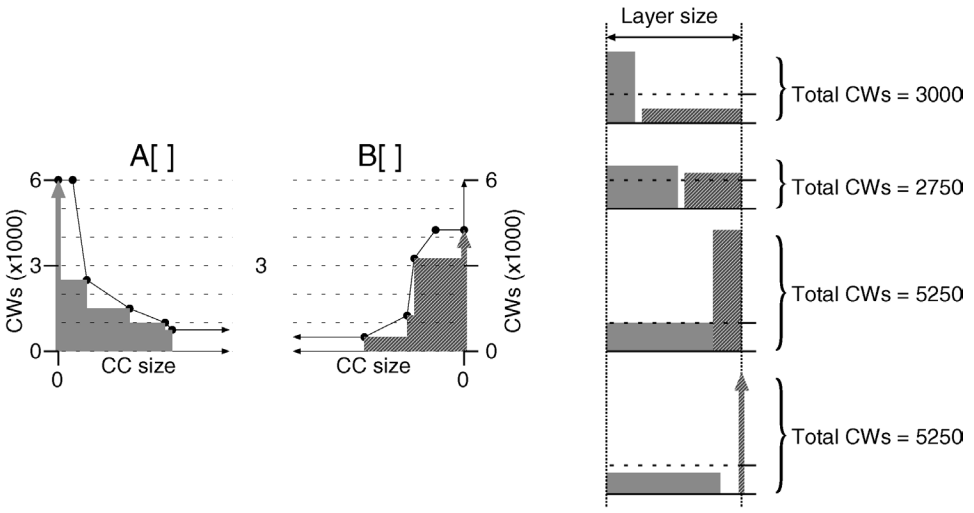


FIGURE 12.3 MHLA trade-offs for the selection of copy candidates.

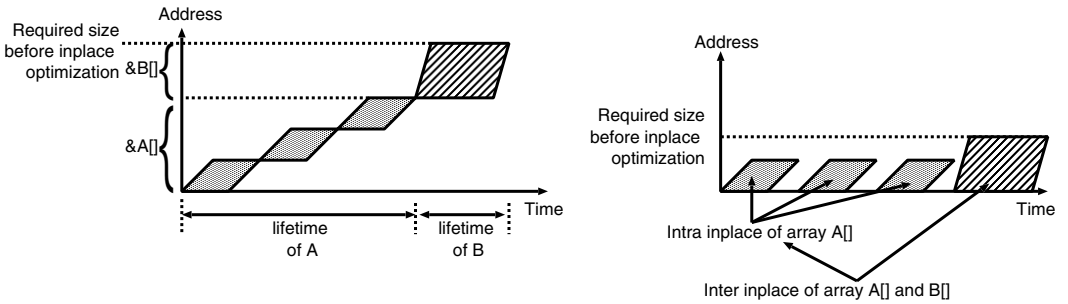


FIGURE 12.4 Illustration of inter/intra-inplace optimization for memory size estimation.

feasible combinations of CC selection to this layer are presented, together with the total number of CWs operations associated to the choice. The upper solution combines the largest CC of array B and the largest feasible CC of array A. The total number of 3000 CWs is the sum of the individual CCs. While constructing the other solutions, some space of the CC of array B is traded for the CC of array A. As a result, the number of CWs for A reduces and the number of CWs for B increases. In this simple example, it is easy to see that the optimal choice is the second in the row with 2550 CWs.

In general, however, realistic applications will have too many possibilities to still find the best solutions manually. Moreover, that explodes the search space when considering more the one layer for the mapping because unnecessary copies from one layer to the next one must be avoided. Additionally, the problem must be considered over the memory hierarchy globally because CWs of one layer can be traded for CWs on another layer. Thus at IMEC, formalized techniques have been developed [28], and these have been implemented on a research prototype tool to support this step.

12.3.3 Lifetime Analysis for Memory Partition Size Estimation

The exploitation of limited lifetime allows having smaller layers or storing more data in an equal sized layer. Both can have a huge impact on energy and performance. Especially, the short lifetime of the CCs should be considered carefully. In addition, it can be expected that a technique without inplace estimation could lead to suboptimal decisions for larger applications because most arrays will have a relatively smaller lifetime as the application complexity increases.

Figure 12.4 illustrates how to exploit the limited lifetimes of the signals to reuse memory locations. At the left of the figure, the plot illustrates how elements of arrays A and B are used in time. The shaded areas indicate which elements are used and for how long. Clearly, the declared size of array A could be reduced by a factor of three by reusing the same locations due to the fact the shaded areas are not overlapping on time. This type of memory location reuse is called intra-inplace [19]. Similarly, the elements of array B do not overlap in lifetime with any of the elements of array A. Thus, the complete array B could also reuse the locations of array A. This type of memory location reuse is called inter-inplace [19]. The results of both inplace opportunities are depicted at the right-hand side of Figure 12.4.

We have implemented a low complexity inter-inplace estimation technique in our prototype tool for a realistic estimation of the required storage in the partitions of the memory layers. That is based on tracking what signals are simultaneously alive in the innermost loops of the application. As a result, we only have to update the storage size of those inner loops that span the lifetime of the CC. This is accurate enough because, typically, most CCs have a very short lifetime and its size is already known from the data reuse analysis phase.

12.3.4 Relation to Other Steps of the DTSE Design Methodology

MHLA is usually applied as part of IMEC's data transfer and storage exploration (DTSE) script [29] developed to optimize the data transfer and storage issues in data-dominated applications. Two phases can be distinguished in this script.

First, a platform independent phase performs program transformations to improve locality of the reference and the required storage size [30]. If this phase is skipped, the approach still works, but the results will typically be far less optimal if the original temporal locality of the code is poor [29]. This will have a clear impact in quality of the search space due to less opportunities for CCs.

The second phase maps the application to the target platform. This is performed in four decoupled steps. MHLA is the first of the platform dependent steps, and it starts by determining for each data set a layer and a memory partition type, thus allowing detailed timing information about the array references be obtained.

This information is required for the next step that optimizes the required memory access parallelism for meeting the timing constrains. Techniques, such as Wuytack et al. [31] and Grun et al. [32], could be used here because certain accesses must still happen in parallel to meet the target timing budget. The conflicting memory accesses must be either stored in different memories or in a dual port memory. In the third memory allocation and assignment step, arrays and CCs assigned to the memory partitions are assigned to the various memory modules within each partition obeying the required parallelism [18,33].

A final data-layout step decides on the storage layout of the data inside the memory. This would further minimize the required size of the memory partition and also avoid the overhead in conflict misses in case of HW-controlled caches (see Section 12.4).

12.4 Case Studies for MHLA Exploration

Two real-life application drivers having different characteristics are selected to illustrate the impact of MHLA decisions on the energy consumed in the memory hierarchy.

The first driver, a quad-tree structured difference pulse code modulation (QSDPCM) algorithm, is an application from the video compression application domain consisting of a few large arrays exposing high temporal locality in the memory references, thus it is largely dominated by data reuse opportunities. It involves an inter-frame compression technique for video images based on a hierarchical motion estimation step, and a quad-tree based encoding of the motion compensated frame-to-frame difference signal [34].

The second driver is a wireless receiver for digital audio broadcast (DAB) containing many smaller arrays, with almost no data reuse opportunities present. The transmission system is based on an orthogonal frequency division multiplex (OFDM) transportation scheme using up to 1536 carriers [35].

For the target platform, we assume two layers of memory hierarchy, the local being implemented using a scratch-pad memory. Although as we will discuss in Section 12.4, our approach is not limited to these memory types, and it is also valid for HW- and SW-controlled caches using a varying number of layers. The used energy model is based on a real memory library. Because relative energy figures are sufficient for the exploration, these are presented relative to an off-chip memory of 1MByte (MB) of fix size. The largest on-chip memory considered is 16 KByte, which is a factor 3 less energy consuming than the off-chip memory. This is because the energy model used is slightly super logarithmic, so a memory that is 256× larger consumes 8.6× more energy per access. This energy model is also used for L0 and L1 in both drivers. This relation demonstrates a very conservative factor in energy per access between off- and on-chip memories.

12.4.1 The QSDPCM Driver

A global view of the QSDPCM's main signals and their data reuse chains is given in Figure 12.5. Many data reuse opportunities exist for the QSDPCM application as can be seen from the many data reuse chains. Figure 12.6(A) shows the energy contribution of the L1 memory (bottom bar) and the main memory (top bar). When increasing the layer size, the energy decreases because fewer accesses occur on main memory (due to the introduction of more CR operations). On the other hand, the reduction in CWs in L1 does not decrease much for a L1 size larger than 640 Bytes (B). Therefore, the L1 energy per

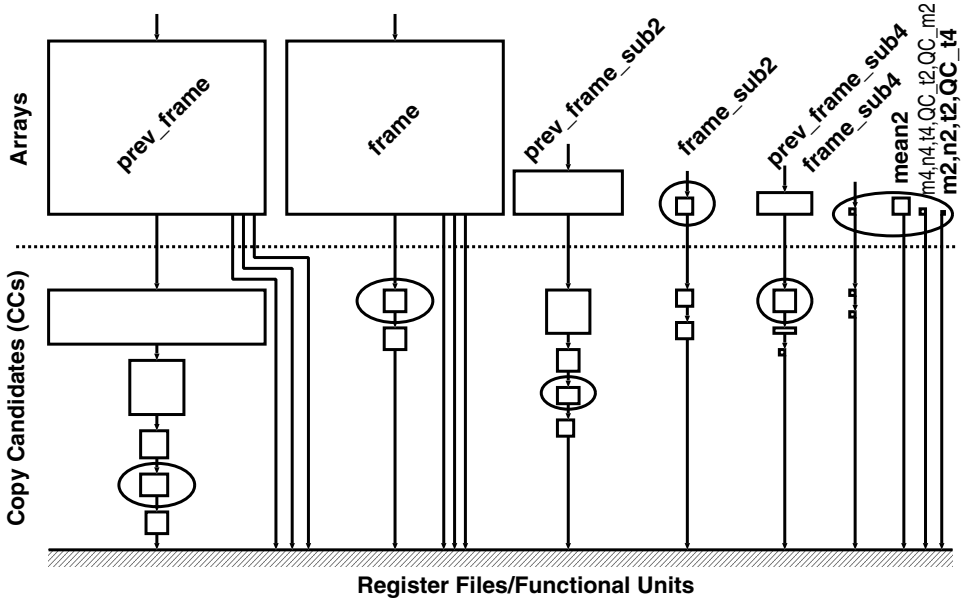


FIGURE 12.5 Assignment of arrays and CCs to the memory hierarchy for QSDPCM. Arrays and CCs marked inside circles have been selected for storage in the local memory.

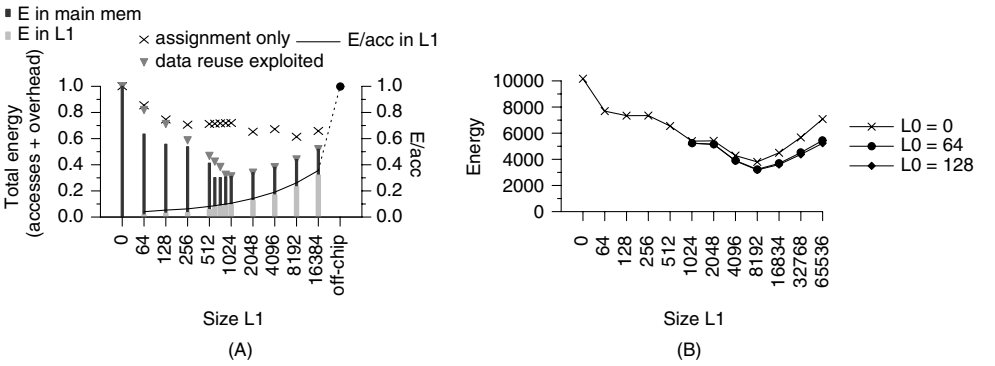


FIGURE 12.6 Memory layer size exploration for energy for both QSDPCM and DAB drivers: (A) varying L1 layer size for QSDPCM (solid line plots); (B) varying L0 (RF) and L1 layer sizes for DAB.

access penalty due to the larger L1 size is not compensated by the potentially less amount of CWs. Thus, the overall energy consumption increases from that point. Thus, an L1 of 640B is the optimal layer size for this application. The optimal assignment of arrays and CCs corresponding to this point is given in Figure 12.5. Arrays and CCs selected as the optimal ones to be stored in L1 are indicated by circles. The rest of arrays are stored in main memory. Nonselected CCs are not stored because copy operations are not needed for these.

To consider the effect of the number of hierarchy layers in energy, we have also reexplored the options by inserting an extra L0 layer. However, this has not resulted in a significant reduction in overall energy because the optimal L1 size for a two-layer platform is already small enough to exploit the data-reuse opportunities present in this application. This is also exposed by comparing our approach with an array-level assignment technique, thus without the introduction of CCs (indicated by crosses in Figure 12.5). In this case, a factor 2 in energy is gained.

TABLE 12.1 Energy Values, Optimal Layer Size, and Numbers of Accesses for the Three-Loop Transformed Versions of QSDPCM

	Nonmerged	Fully Merged	Partially Merged
Total Energy	71.97	47.17	46.47
L1 Energy	13.79	14.32	14.29
L2 Energy	58.18	32.85	32.17
Layer size for L1 (Bytes)	742	748	802
Layer size for L2 (Bytes)	114048	66352	63360
Number of L1 accesses ($\times 10^6$)	1.14	1.19	1.19
Number of L2 accesses ($\times 10^3$)	542	306	300

12.4.2 Global Loop Transformations for Improved Memory Hierarchy Utilization in QSDPCM

For the QSDPCM, we have also evaluated the gains in overall energy by disabling the inplace mapping heuristics mentioned in the earlier section for an accurate estimation of the storage size in L1. This case is indicated in Figure 12.5 by triangles. The use of the heuristic allows a L1 size of 640B instead of the 1KB required otherwise; however, the overall energy gain in this case is rather limited (only another 3.2%).

The limited effect of the inplace mapping optimization techniques is due to the overall loop structure of the code (with several loop nests each traversing an image frame), thus requiring intermediate buffers to store data between loop nests. However, it is possible to largely change the lifetime of the data, and thus the size of these buffers, by globally transforming the loop structure of the application source code as illustrated by Catthoor et al. [29].

To overcome this limitation and to study the effect that an optimized storage size would have in the memory hierarchy, we have implemented loop-merging transformations in the initial code because they eliminate the need for intermediate buffers and, potentially, reduce the lifetime of the data. All these optimizing transformations are part of the global storage cycle budget distribution (SCBD) step in the DTSE script [29,36].

Three loop-transformed versions of the QSDPCM have been implemented: the original version with all loops (hereafter called nonmerged), a version with most loops merged into a global one (fully merged); and an intermediate version of the last one where only selected loops have been merged together (partially merged). Table 12.1 gives an overview of the results obtained on each code version.

Clearly, the best solution for energy is for the partially-merged version. In all cases, the energy contributions from the L1 are very similar. This is somehow coincidental because the arrays and copies assigned to L1 are very different in all three cases and the amount of L1 accesses could have been more different; however, the L2 energy contribution is much smaller in the fully and partially merged code versions. This is due to the big reduction of the number of accesses to this layer. In this case, the total energy gain is rather small with the partially merged version, but exploring other solutions may provide better results. An important conclusion appears contrary to what intuitively one would expect, the most local code (fully merged) is not always the best one for energy, and an exploration of the possible transformation options is therefore needed.

For the fully-merged and the partially-merged codes, an exploration on the L1 size has been also performed during the MHLA step. It appears that for both versions the best hierarchy for energy is with an L1 memory of 1 KB. This appears to be a good trade-off between the amount of CW operations (important for small L1 memories) and the energy per accesses (that increases with the layer size). The same size of 1 KB for the nonmerged code has been kept to fairly compare all three versions.

12.4.3 The DAB Driver

Similar to the previous driver, an L1 size exploration is performed and presented in the top curve of Figure 12.6(B). For this driver, the minimum energy consumed is found for an L1 size of 8 KB; however,

TABLE 12.2 MHLA Results for DAB for the TriMedia TM1300 Processor

	Estimated Load/Stores	Actual Load/Stores	Required Registers	Data Cache Misses	CPU Cycles
Mild register usage	17,152	18,408	22	2895	91,552
Aggressive register usage	11,232	11,837	70	763	47,341

the introduction of an extra L0 layer has brought additional gains in energy in this case. By implementing this L0 layer as a register file (RF) with varying size from 64 to 128 B and reexploring the assignment, an extra 15% in energy is gained, while the optimal size for layer L1 remains unchanged. The reason is an additional reduction in 25% of the memory accesses in the OFDM block now becoming RF operations.

The most energy efficient assignment decision has been used to map the DAB application to the TriMedia processor. This processor is selected because it has a data memory hierarchy that matches the optimal architecture. The processor has an L0 layer of 128 registers, 16 KB cache and 8 MB of SDRAM; however, the exploitation of the L0 register file has to be carefully evaluated. On one hand, the number of data load stores will decrease as the size of L0 increases because more data remains in the L1. On the other hand, the higher register pressure might counteract this gain as register spilling is required to schedule all instructions given the actual RF size available in the processor. In addition, the unrolling transformations, required to keep the data in the RF, needs more instruction cache space.

The trade-off between the load stores, spilling, and instruction cache is given in Table 12.2. The native TriMedia simulator is used for the evaluation of three differently transformed implementations having a relatively more aggressive L0 register usage (second row). The large reduction of 34% in memory accesses has of course a large impact on data memory energy and performance (as presented in Table 12.2). In addition, the prediction of the MHLA technique has been very close to the actual number of accesses. After an examination of the analysis report provided by the compiler, the small difference between MHLA's estimated activity and the simulation results can be explained by the few register spilling operations.

Of course, the same result could be obtained manually by an experienced designer using a combined trial and error and simulation process; however, MHLA can significantly save design time by predicting beforehand the good solution. Moreover, this prediction gives the designer a target minimum for which to optimize. No time is wasted in finding more opportunities that actually will not pay off.

12.5 SW-Controlled Cache Miss Optimizations

Before introducing SW optimizations for HW and SW controlled caches, in this section we revise the traditional classification [37] of cache misses from a compiler viewpoint. This is the first step needed before developing design-time optimization techniques for cache memories. The definition of a cache miss does not change though. In the most general case a cache miss is any transfer of data from main memory to the main memory taking place whenever a load/store operation issues for the data that is not present in the cache. Transfers from the cache to main memory are write-backs and these are not focus of this work.

12.5.1 Compiler-Centric Cache Miss Classification

In computer architecture literature [37], cache misses have been traditionally divided in three categories: compulsory, capacity, and conflict miss types; however, that classification is purely architecture oriented and is clearly insufficient for compiler optimizations because they do not properly expose the nature of the miss type. For that purpose, we have defined a new taxonomy for data cache misses, which spans seven categories. These are compulsory, minimal capacity, block prefetch, block allocate, associativity conflict, replacement, and data-layout conflict misses. This new taxonomy, while isolating the different sources of overhead for cache misses, is much better suited for modern SW-controlled caches than previously proposed ones [37].

To study the nature of the miss contribution, we start by assuming two caches for comparison purposes: a real cache (our target cache) and an ideal cache. The ideal cache is a memory of infinite size with a single data element per line, implementing a fully associative mapping organization, with an optimal replacement policy and following an optimal block allocate policy. Unlike the write-around or fetch-on-write block allocate policies, we consider an optimal block allocate policy in which a write miss ensures that no block of data is unnecessarily allocated in a cache line when all data elements of that line are to be updated before the line is to be replaced. As for the optimal block replacement policy [37], the optimal block allocation policy cannot be implemented via a hardware mechanism because that would imply predicting the behavior of the algorithm in the future with respect to the actual state of the cache.

12.5.1.1 Compulsory Misses

Assuming our definition of ideal cache, a compulsory miss happens whenever a new data element of the application's test bench is needed by the application's algorithm and this data needs to via the cache or local memory for the first time. Any data written from the algorithm back to memory (back to the test bench) can be considered a compulsory write-back. All miss types not yet dealing with prefetching issues must be accounted at the data level. This applies both to compulsory and minimal capacity misses.

Compulsory misses are independent of the physical memory organization parameters (such as the cache memory size) and controller policy issues (such as the replacement policy). Thus, these are truly platform independent and their contribution depends solely on the application characteristics. This is the only miss type requiring a change in the execution order of the application to be avoided. This can be done by means of global data-flow [23] or global loop transformations [29]. All the rest of the cache misses are data-layout related for a given execution order, thus the way to avoid these is by transforming the placement of data in main memory. This miss type is valid for HW-controlled caches, SW-controlled scratch-pad memories and mixes, and it is one of the major (but largely unavoidable) contributors to overall miss count for DM-caches.

12.5.1.2 Minimal Capacity Misses

After assuming an ideal cache, platform constraints can start to be gradually introduced in our target cache, while still assuming optimal controlling policies. In this context, the first platform dependent, controller policy independent miss type is the minimal capacity miss and expressed as the additional contribution in misses in an ideal cache, which is constrained in size (by same capacity in bytes) as the real cache. Due to this size limitation, some data values are discarded and have to be retrieved later. If the cache is full, the data value to discard is the furthest to be used in the future (optimal replacement policy). Minimal capacity misses are significant contributors to the overall miss count, especially for small cache sizes; however, the MHLA techniques discussed in Section 12.3 are effective in minimizing the overhead of these, thus enabling the efficient utilization of small cache spaces.

12.5.1.3 Block Prefetch Misses

This miss type is related to the line size of the cache. By definition, a block is the group of data fetched from main memory and that is allocated altogether in the same cache line. A block prefetch miss is the next miss contribution for an ideal cache with the same number of lines and same line size as the real one. It characterizes the miss occurring from a data that is required in the very near future (such as data being subsequently fetched in the innermost loop of an application) and that is not present in the block that has been most recently loaded in cache. This miss contribution gives an indication on how inefficiently the memory layout adapts to the ideal situation where all data prefetched in a cache line is actually read before this line is to be flushed due to cache capacity limitations or other reasons.

Together with block prefetch misses, minimal capacity misses depend on the actual physical memory organization parameters (such as number of lines and line size), thus these are platform dependent miss types although still independent on the controller-policy of the cache (such as replacement policy and mapping organization). These miss types are valid for purely HW and SW controlled caches and mixes when prefetching is exploited in the purely SW controlled case.

12.5.1.4 Block Allocate Misses

At this point, the geometry of the cache is now fully fixed and the effect in miss rate due to a nonoptimal controlling policy is considered. These are platform and controller-policy dependent misses, and they are valid miss types for purely HW controlled caches but less relevant for mixes and irrelevant for purely SW controlled caches (pure compiler controlled policies).

In this context, the first platform and controller policy dependent miss type is the block allocate miss. This is the next miss contribution resulting from the (nonoptimal) allocation policy of the real cache. Some caches (especially the HW controlled types) fetch the block from main memory on a block write miss (e.g., following a fetch-on-write block allocate policy). This is done to ensure the consistency between the copy of the data in the cache and the value stored in memory. This extra read operation (block allocate miss) should only be needed in case the block is replaced before all its data elements have been updated; in any other case this transfer is considered redundant, thus a miss.

Block prefetch and block allocate miss types can be clustered in a common category (block miss) to collect all miss effects related to the organization of data in the cache in lines and the transfer of data in blocks. The impact of block misses can also be high when the effect of SDRAM page misses (in main memory) is incorporated; however, this is not the focus of this chapter but a topic for future research work.

12.5.1.5 Associativity Conflict Misses

This is the next miss contribution (measured at the block level) due to nonideal policies, and it results from the limited associativity of the real cache. A cache that is not fully associative will limit the replacement freedom, even when the mapping organization is exploited by using an optimal memory data-layout. For instance, if the same block has to be loaded in the cache several times (due to any of the previous miss effects), the block may be allocated in different places against what the optimal replacement policy will actually select. Indeed, a particular data-layout could eventually avoid the first allocations of the target block that will conflict with an existing one in the cache, but not for the subsequent allocations of the same block. The contribution of this miss type in the overall miss count for DM-caches is limited due to the fixed mapping organization.

12.5.1.6 Replacement Misses

This is the next miss contribution resulting from the (nonoptimal) replacement policy (typically least recently used or LRU [37]) of the real cache. An optimal replacement strategy uses the capacity of the cache and misses can be avoided because this policy exploits knowledge of the future accesses. For instance, data required in the near future will not be replaced. With a nonoptimal replacement policy, however, it can happen that data are flushed just before being required, which is the source of extra misses. Therefore, a suboptimal replacement strategy increases the replacement misses. In advanced multimedia caches, the replacement policy is software controlled and in DM-caches there is no replacement policy due to its fixed mapping organization. Therefore, these misses can be clearly ignored in our context.

12.5.1.7 Data-Layout Conflict Misses

Misses in this last category are purely devoted to isolate the effects due to a nonoptimal utilization of the cache mapping organization. At this point, both the geometry and controller policy of the cache are fully fixed and the only misses left are related to a nonoptimal utilization of the cache mapping organization. Therefore, data-layout conflict misses are the last miss contribution, thus by definition these are any misses that could be avoided by choosing an optimal (conflict oriented) memory data-layout (e.g., using array interleaving, array padding, selective gap placement, array merging, or any mix of these). Two types of data-layout conflict misses exist: inter-array when elements of different arrays conflict in the same cache line and, intra-array when the situation happens for elements of the same array.

These misses are typically the largest contributors to miss rate in DM-caches, once a good memory layer assignment has been selected (as we have achieved with the techniques of Section 12.3). Data-layout techniques for the elimination of these misses are the topics addressed in rest of this chapter.

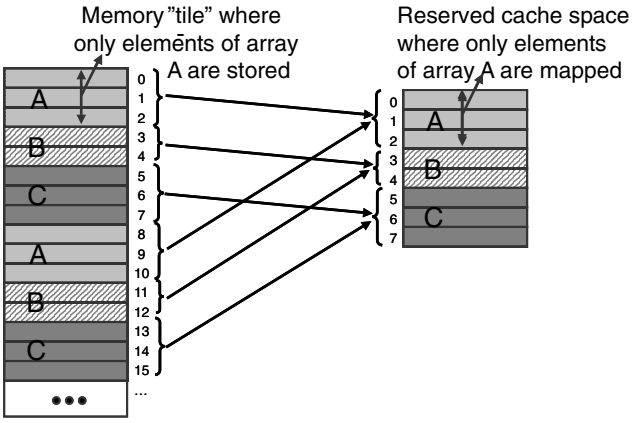


FIGURE 12.7 Illustration of how memory tiles reserve spaces in cache.

12.5.2 Data-Layout Transformations for Conflict Miss Reduction

To maintain the decisions taken during the MHLA step, we must ensure that the overall miss contribution follows the estimated number of CWs at that level. Therefore, it is needed to ensure that the data-layout conflict miss contribution can be minimized to a large extent. This means that for each array a certain amount of space should be reserved in the cache to hold all data of the the selected CC without this being flushed because of conflicts in the cache lines.

The necessary spaces can be reserved in the (lockable) cache by using the array-interleaved data-layout approach for data-layout conflict misses presented in Kulkarni [38]. In this chapter the original approach is improved further by considering not only the inter-array data-layout conflict misses but also the intra-array component and the influence of the MHLA decisions in minimal capacity misses.

Our data-layout approach aims at an interleaved placement of arrays in main memory such that every array is distributed in memory as data "tiles." It assumes a cache with fixed mapping organization as the one found in DM-caches. For explanation purposes, we will also consider (without loss of generality) cache lines of the same size as each memory location containing at most one array element. The concept is illustrated in Figure 12.7. In this case, due to the fixed mapping organization of the DM-cache, every element of arrays A[], B[], or C[] gets a fixed cache location, which is defined by the fixed mapping organization of the DM-cache (e.g., $C_{addr} = M_{addr} \% C_{size}$, where C_{addr} and M_{addr} are the cache and memory locations, and C_{size} is the size of the cache). The interleaved nature of our data-layout approach ensures that the tiles of the same array are evenly distributed in memory at equal address distances. This is achieved by an index transformation (e.g., $index\%TS + index \div TS \times C_{size} + O_{ffset}$, with *index* being the index expression of the array reference, *TS* the tile size, and O_{ffset} the offset of the original array placement). For this data layout, the cache controller will always assign the same cache space to elements of the same array (see Figure 12.7). As a consequence, each array will always get disjoint address spaces in the cache, thus elements of different arrays will never conflict in the same cache line, thus eliminating any possible inter-array data-layout conflict miss effect.

In addition, this data organization allows for an address range of the cache that is always reserved to the elements of a particular array. We can use this property to ensure that a fixed amount of the cache size is always devoted to hold data of that array. Therefore we can ensure that the optimal MHLA decisions, relating the amount of memory space reserved for a particular CC, are also guaranteed even when the control is done in HW and not only in SW. This is done by choosing the tile size of each array to be equal to the size of the selected CC (see Section 12.3).

We must also consider the intra-array effect, however, because elements from the same array may still conflict in the same cache line. This situation happens because different elements of a working set (e.g., the elements of an array belonging to the copy selected by the cache controller) could be located within

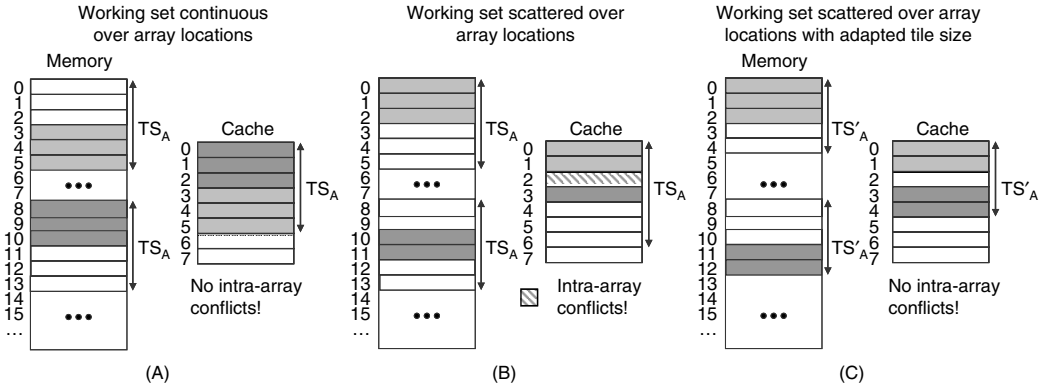


FIGURE 12.8 Illustration of how intra-array, data-layout conflict misses occur in an array-interleaved data-layout: (A) situation without intra-array conflict misses for a Working Set (WS) over continuous array locations; (B) situation with (potential) intra-array conflict misses for a WS with locations scattered all over the array; (C) adaptation of the tile size to avoid intra-array conflict misses for situation (B).

the array at distances larger than the actual tile size, thus leading to cache mapping situations with potentially conflicting cache lines. On the other hand, there are also situations when the mapping organization of the cache causes no conflict misses. This is typically the case when the different elements of the working set are accessed at consecutive locations within the array, thus they will never overlap in cache. Figures 12.8(A) and 12.8(B) illustrate both concepts.

These data-layout conflict situations due to intra-array effects can be avoided by ensuring that the elements of the working sets within a memory tile are evenly distributed over the available cache locations. In fact, when studying the miss contributions for both minimal capacity and data-layout conflict misses for different tile-sizes, a seemingly random behavior is observed: very high and very low miss counts are possible for small variations of the tile size. Indeed, changes in the tile size can modify the cache mapping decisions for the affected array, thus some sizes could lead to more conflict situations that others and vice versa. In general, the larger the tile size, the less the probability to have conflicts in the cache lines. Still, it is possible to find small enough tiles for which the amount of conflicts is still minimized. Moreover, these variations can be made small enough such as it is possible to trade few minimal capacity misses for a large amount of data-layout conflict misses. This is done by adjusting the tile size, thus avoiding a "peak" and selecting one of the neighborhood "valleys." Figure 12.9(A) illustrates the effect that the tile

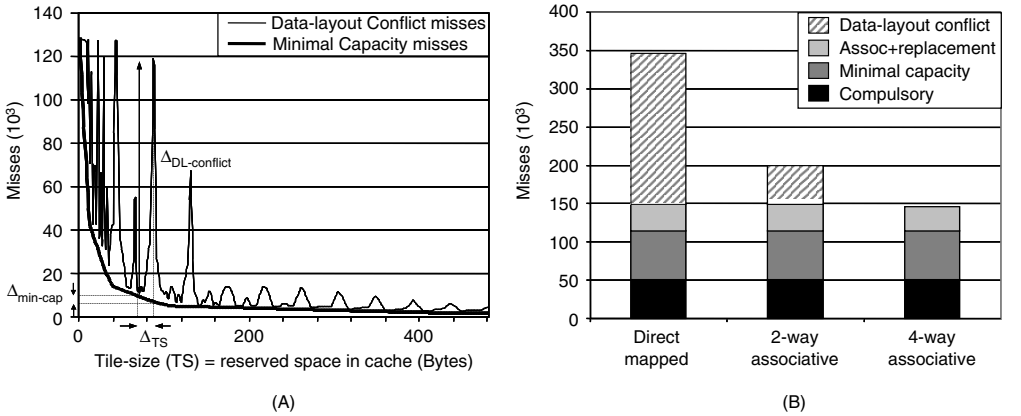


FIGURE 12.9 Cache miss simulations for QSDPCM: (A) miss count contributions due to minimal capacity and data-layout conflict misses in a DM-cache associated to one of the arrays in the QSDPCM when changing the tile size; (B) simulation results for the complete application with breakdown on different miss type contributions.

size has in miss rate for one of the arrays of the QSDPCM driver. The locations of these "valleys" can be avoided by ensuring that cache-lines are evenly used by the elements of the working set. This situation happens when the combination between tile size and the distances between elements of the working sets are co-prime numbers (e.g., numbers not having a greatest common divisor other than 1) as illustrated in Figure 12.8(C).

Finally, the mapping between elements of the working set and the cache lines can be influenced not only by adjusting the tile sizes but also by changing the size of the dimensions of the arrays (e.g., by using array padding techniques [39]). Globally considered, changing the tile size affects the mapping organization by modifying the space reserved in cache, while changing the padding decision also affects the mapping organization by modifying the memory address. However, a bigger tile size for one array decreases the amount of cache lines available for other arrays in the cache. Thus, these decisions must be globally considered during the MHLA step. A drawback to the use of array padding techniques is that they would definitely introduce holes in the array and those holes will become unused locations in main memory, thus the memory occupation would be far less efficient than adjusting the tile size. In fact, a trade-off exists in sacrificing cache or memory space for reducing the conflict misses. It is desirable that both memory and cache costs are globally minimized though. A global exploration is needed because the choice for a tile size for one array influences the choice for other arrays. Such exploration can be performed for the complete application in which the additional space in either cache or main memory is expressed; however, the details for that extension fall outside the scope of this chapter.

12.5.3 Case Study for Data-Layout Transformations

To evaluate the impact of data-layout transformations in cache miss rate, we have used the QSDPCM driver introduced in Section 12.3.

During the memory-hierarchy layer assignment step (see Section 12.3) an optimal L1 layer of 1KB has been selected based on the estimated amount of accesses to the different layers and the energy-consumption per memory access. During that exploration, the sizes of the arrays and CCs are used as initial tile sizes within the cache, and these are passed to the data-layout phase as first estimation. In the fine-tune phase, the tile sizes are slightly changed and some array dimensions padded marginally (e.g., by extending their declaration size). This is done to optimize the data-layout for intra-array conflict-misses.

Note there is no need to optimize those signals whose complete top-level array is selected to be stored in the cache. In this case, each element has its own location in the cache, elements of this signal will never compete for the same location. In this case, there are four signals (prev_frame, frame, prev_sub2_frame, and prev_sub4_frame) with a selected copy-candidate smaller than the top-level array (see Figure 12.5). This potentially results in conflict misses that should be avoided. The refinement of the initial tile size as indicated by MHLA allows avoidance of the data-layout conflict miss contributions and retention of the miss contribution due to minimal capacity misses decided by the precedent MHLA step.

Simulations have been performed to measure the impact of the data-layout optimizations for the complete QSDPCM application. The main results are given in Figure 12.9(B). The two bottom bars of the graph represent the (unavoidable) miss contribution due to compulsory and minimal capacity misses. To have a reference of how good the performance is when an optimal data-layout is applied, the miss rates of the DM-cache are compared to a two-way and a four-way associative cache with the least recently used replacement policy. Those caches have a more "intelligent" controller, and they should be able to resolve (part of) the conflict misses; however, the DM-cache with an optimized data-layout can still outperform the two-way associative cache (without data-layout optimizations) and become as efficient as a four-way associative cache. Note that a four way associative cache is about two to three times less energy efficient than the DM-cache of the same size (see Section 12.1). Thus, the combination of MHLA techniques (for an optimal sizing of the local memory layer) together with the use of data-layout optimization techniques (for the efficient use of DM-caches) provides a low-power alternative for platform architects and application designers of application domain specific solutions.

12.6 Conclusions

In this chapter, we have presented design time optimization techniques for an optimal dimensioning of the memory hierarchy for low-power operation. This is based on data-to-memory hierarchy layer assignment decisions that consider information resulting from the temporal locality and lifetime characteristics of the application data. This is true both for HW and SW controlled memories (e.g., scratch-pad and cache memories). For HW-controlled caches, however, because of the mapping organization of the hardware controller, additional data-layout organization techniques are needed to avoid the overhead in additional data transfer resulting from conflicts in the cache lines.

References

[1] T. Achteren, R. Lauwereins, and F. Catthoor. Systematic data reuse exploration techniques for non-homogeneous access patterns. *Proc. 5th ACM/IEEE Design and Test in Europe Conf. (DATE),* pp. 428–435, Paris, France, Apr. 2002.

[2] I. Issenin, E. Brockmeyer, M. Miranda, N. Dutt, Data reuse analysis technique for software-controlled memory hierarchies, *Proc. 7th ACM/IEEE Design and Test in Europe Conf. (DATE),* pp. 202–207, Paris, France, Feb. 2004.

[3] S.J.E. Wilton and N.P. Jouppi, CACTI: an enhanced cache access and cycle time model, *IEEE J. of Solid-State Circuits,* Vol. 31, No. 5, pp. 677–688, May 1996.

[4] P. Marchal, J.I. Gomez, D. Bruni, F. Catthoor, M. Prieto, L. Benini, and H. Corporaal, SDRAM-energy-aware memory allocation for dynamic multi-media applications on multi-processor platforms. *Proc. 6th ACM/IEEE Design and Test in Europe Conf. (DATE),* pp. 516–521, Munich, Germany, March 2003.

[5] C. Kulkarni, M. Miranda, C. Ghez, F. Catthoor, and H. De Man, Cache-conscious data layout organization for embedded multimedia applications, *Proc. 4th ACM/IEEE Design and Test in Europe Conf. (DATE),* Munich, Germany, pp. 686–691, March 2001.

[6] C. Ancourt et al. Automatic data mapping of signal processing applications. *Proc. Int. Conf. on Application-Specific Array Processors,* Zurich, Switzerland, pp. 350–362, July 1997.

[7] J. Anderson, S. Amarasinghe, and M. Lam. Data and computation transformations for multiprocessors. *5th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming,* pp. 39–50, Aug. 1995.

[8] M. Kampe and F. Dahlgren. Exploration of spatial locality on emerging applications and the consequences for cache performance. *Proc. Int. Parallel and Distributed Processing Symp. (IPDPS),* pp. 163–170, Cancun, Mexico, May 2000.

[9] H.-B. Lim and P.-C. Yew. Efficient integration of compiler-directed cache coherence and data prefetching. *Proc. Int. Parallel and Distributed Processing Symp. (IPDPS),* pp. 331–339, Cancun, Mexico, May 2000.

[10] L. Benini and G. De Micheli, System-level power optimization techniques and tools, *ACM Trans. on Design Automation for Embedded Syst. (TODAES),* Vol. 5, No. 2, pp. 115–192, Apr. 2000.

[11] L. Benini, A. Bogliolo, and G. Micheli. A survey of design techniques for system-level dynamic power management, *IEEE Trans. on VLSI Syst.,* pp. 299–316, 2000.

[12] P. Panda, F. Catthoor, N. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandecappelle, and P.G. Kjeldsberg, Data and memory optimizations for embedded systems, *ACM Trans. on Design Automation for Embedded Syst. (TODAES),* Vol. 6, No. 2, pp. 142–206, Apr. 2001.

[13] M. Kandemir and A. Choudhary. Compiler-directed scratch-pad memory hierarchy design and management. *39th ACM/IEEE Design Automation Conf.,* pp. 690–695, Las Vegas, NV, June 2002.

[14] P. Grun, N. Dutt, and A. Nicolau. Apex: access pattern based memory architecture exploration. *14th Int. Symp. on Syst. Synthesis,* pp. 25–32, Montreal, Canada, Oct. 2001.

[15] P.R. Panda, N.D. Dutt, and A. Nicolau. Data cache sizing for embedded processor applications. *Proc. 1st ACM/IEEE Design and Test in Europe Conf. (DATE),* pp. 925–926, Paris, France, Feb. 1998.

[16] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel. Assigning program and data objects to scratch-pad for energy reduction. *Proc. 5th ACM/IEEE Design and Test in Europe Conf. (DATE),* pp. 409–415, Paris, France, Apr. 2002.

[17] K. Masselos et al. Memory hierarchy layer assignment for data re-use exploitation in multimedia algorithms realized on predefined processor architectures. *8th IEEE Int. Conf. on Electron., Circuits and Syst. (ICECS),* pp. 285–288, Oct. 2001.

[18] L. Benini, L. Macchiarulo, A. Macii, and M. Poncino. Layout-driven memory synthesis for embedded system-on-chip. *IEEE Trans. on VLSI,* pp. 96–105, Sep. 2002.

[19] E. de Greef, Storage size reduction for multimedia applications. Doctoral dissertation, ESAT/KUL, Belgium, Jan. 1998.

[20] M. Kandemir, J. Ramanujam, and A. Choudhary, Improving cache locality by a combination of loop and data transformations, *IEEE Trans. on Comput.,* Vol. 48, No. 2, pp. 159–167, Feb. 1999.

[21] C. Kulkarni, F. Catthoor, and H. De Man, Cache transformations for low power caching in embedded multimedia processors, *Proc. Int. Parallel Processing Symp. (IPPS),* pp. 292–297, Orlando, FL, Apr. 1998.

[22] M. Lam, E. Rothberg, and M. Wolf, The cache performance and optimizations of blocked algorithms, *Proc. 4th Int. Conf. on Architectural Support for Prog. Lang. and Operating Syst. (ASPLOS),* pp. 63–74, Santa Clara, CA, Apr. 1991.

[23] F. Catthoor, S. Wuytack, E. de Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology — Exploration of Memory Organisation for Embedded Multimedia System Design,* Kluwer Academic Publishers, Boston, 1998.

[24] N. Manjiakian and T. Abdelrahman, Fusion of loops for parallelism and locality. Technical report CSRI-315, Computer Systems Research Institute, University of Toronto, Ontario, Canada, Feb. 1995.

[25] P. Panda and N. Dutt, Low-power mapping of behavioral arrays to multiple memories, *Proc. IEEE Int. Symp. on Low Power Design,* pp. 289–292, Monterey, CA, Aug. 1996.

[26] P.R. Panda, N.D. Dutt, and A. Nicolau, Efficient utilization of scratch-pad memory in embedded processor applications, *Proc. 5th ACM/IEEE Design and Test in Europe Conf. (DATE),* Paris, France, Mar. 1997.

[27] D.C. Burger, J.R. Goodman, and A. Kagi, The declining effectiveness of dynamic caching for general purpose multiprocessor. University of Wisconsin Computer Sciences Tech. Report CS-TR-95-1261—Madison, WI, 1995.

[28] E. Brockmeyer, M. Miranda, F. Catthoor, and H. Corporaal, Layer assignment techniques for low-power in multi-layered memory organisations, *Proc. 6th ACM/IEEE Design and Test in Europe Conf. (DATE),* pp. 1070–1075, Munich, Germany, March 2003.

[29] F. Catthoor, K. Danckaert, C. Kulkarni, E. Brockmeyer, P.G. Kjeldsberg, T. Van Achteren, and T. Omnes, *Data Access and Storage Management for Embedded Programmable Processors,* Kluwer Academic Publishers, Boston, 2002.

[30] F. Catthoor, K. Danckaert, S. Wuytack, and N. Dutt, Code transformations for data transfer and storage exploration preprocessing in multimedia processors, *IEEE Design and Test of Comput.,* Vol. 18, No. 2, pp. 70–82, June 2001.

[31] S. Wuytack, F. Catthoor, G. Jong, B. Lin, and H. Man. Flow graph balancing for minimizing the required memory bandwidth. *Proc. 9th ACM/IEEE Int. Symp. on System-Level Synthesis (ISSS),* pp. 127–132, La Jolla, CA, Nov. 1996.

[32] P. Grun, N. Dutt, and A. Nicolau. Mist: an algorithm for memory miss traffic management. *Proc. IEEE Int. Conf. on CAD,* pp. 431–437, Santa Clara, CA, Nov. 2000.

[33] P. Slock, S. Wuytack, F. Catthoor, and G. Jong. Fast and extensive system-level memory exploration for ATM applications. *Proc. 10th ACM/IEEE Int. Symp. on System-Level Synthesis (ISSS),* pp. 74–81, Antwerp, Belgium, Sep. 1997.

[34] P. Strobach. QSDPCM — a new technique in scene adaptive coding. *Proc. 4th Eur. Signal Processing Conf. (EUSIPCO),* pp. 1141–1144, Grenoble, France, Sep. 1988.

[35] Radio broadcasting systems; digital audio broadcasting to mobile, portable and fixed receivers. Standard RE/JTC-00DAB-4, ETSI, ETS 300401, May 1997.

[36] K.C. Shashidar, A. Vandecappelle, and F. Catthoor, Low-power design of turbo decoder module with exploration of energy-performance trade-offs, *Workshop on Compilers and Operating Systems for Low Power (COLP '01)* in conjunction with *Int. Conf. on Parallel Architecture and Compilation Techniques (PACT),* pp. 10.1–10.6, Barcelona, Spain, Sep. 2001.

[37] D. Patterson and J. Hennessey, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann, San Francisco, CA, 1996.

[38] C. Kulkarni. Cache optimization for multimedia applications. Doctoral dissertation, ESAT/KUL, Belgium, 2001.

[39] P.R. Panda, H. Nakamura, N.D. Dutt, and A. Nicolau, A data alignment technique for improving cache performance, *Proc. IEEE Int. Conf. on Comput. Design,* pp. 587–592, Santa Clara, CA, Oct. 1997.

13

Networks on Chips: Energy-Efficient Design of SoC Interconnect

Luca Benini
University of Bologna

Terry Tao Ye
Giovanni de Micheli
Stanford University

13.1 Introduction

The challenge of designing systems on chip (SoCs) is related to both the large scale of integration and the small transistor features in the upcoming silicon technologies. Moreover, SoCs will be applied in many embedded systems, where reliability of operation and low-energy consumptions are key figures of merit.

It is a common belief that SoCs are designed using preexisting components, such as processors, controllers, and memory arrays. Design methodologies have to support component reuse in a plug-and-play fashion to be effective.

We think that the most critical factor in system integration will be related to the communication scheme among components. The implementation of on-chip communication largely affects the system correctness, reliability, and energy consumption. Indeed, technology trends foresee an increase in device density and frequency of operation, which both correlate to higher power consumption. Voltage down-scaling will mitigate the energy cost at the expenses of reduced signal integrity. Thus, future system designs will be based on a balancing act between performance, reliability, and energy consumption. This chapter will analyze this trade-off in the domain of on-chip component interconnection.

The challenges for on-chip interconnect stem from the physical properties of the interconnection wires. Propagation delays on global wires — spanning a significant fraction of the chip size — will carry signals whose propagation delay will exceed the clock period. Thus, signals on global wires will be pipelined. At the same time, the switched capacitance on global wires will constitute a significant fraction of the dynamic power dissipation. Moreover, estimating delays accurately will become increasingly harder, as wire geometries may be determined late in the design flow. Thus, the need for latency insensitive design is critical. The most likely synchronization paradigm for future chips is globally asynchronous locally synchronous (GALS), with many different clocks.

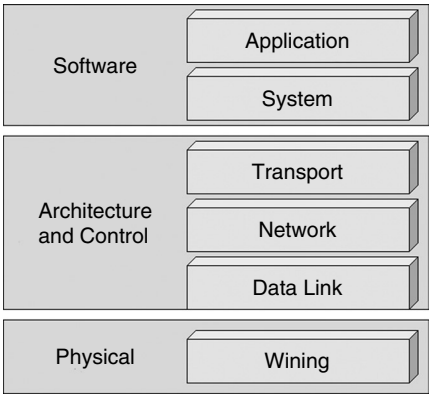


FIGURE 13.1 Micro-network stack.

SoC design will be guided by the principle of consuming the least possible power. This requirement matches the need of using SoCs in portable battery-powered electronic devices and of curtailing thermal dissipation, which can make chip operation infeasible or impractical. Energy considerations will impose small logic swings and power supplies, most likely below 1 V. Electrical noise due to crosstalk, electromagnetic interference (EMI), and radiation-induced charge injection (soft errors) will be likely to produce data upsets. Thus, the mere transmission of digital values on wires will be inherently unreliable.

To cope with these problems, we use network design technology to analyze and design SoCs modeled as micro-networks of components. The SoC interconnect design analysis and synthesis is based upon the micro-network stack paradigm, which is an adaptation of the protocol stack [30] (Figure 13.1) used in networking. This abstraction is useful for layering micro-network protocols and separating design issues belonging to different domains.

SoCs differ from wide-area networks because of local proximity and because they exhibit much less nondeterminism. In particular, micronetworks have a few distinctive characteristics, namely, energy constraints, design-time specialization, and low communication latency. This chapter addresses specifically the first problem.

Whereas computation and storage energy greatly benefits from device scaling (smaller gates, smaller memory cells), the energy for global communication does not scale down. On the contrary, projections based on current delay optimization techniques for global wires [15,28,29] demonstrate that global communication on chip will require increasingly higher energy consumption. Thus, communication-energy minimization will be a growing concern in future technologies. Furthermore, network traffic control and monitoring can help in better managing the power consumed by networked computational resources. For instance, clock speed and voltage of end nodes can be varied according to available network bandwidth. The emphasis on energy minimization creates a sleuth of novel challenges that have not been addressed by traditional high-performance network designers.

Design-time specialization is another facet of the SoC network design. Whereas macroscopic networks emphasize general-purpose communication and modularity, in SoCs networks, these constraints are less restrictive, because most on-chip solutions are proprietary. This degree of freedom can be used effectively to design low-energy communication schemes.

13.2 Micro-Networks: Architectures and Protocols

Much literature is available about architectures for macroscopic networks and, more specifically, for single-chip multi-processors [6,12,17]. These architectures can be classified by their topology, structure, and parameters. The most common on-chip communication architecture is the shared medium architecture, as exemplified by the shared bus. Unfortunately, bus performance and energy consumption are deeply penalized by the scaling up of the number of end nodes. Point-to-point architectures, such as

mesh, torus, and hypercube, have been demonstrated to scale up despite a higher complexity in their design. Examples of recent micro-network architectures include Octagon [17], which is a direct network (i.e., with routers attached to the end node) consisting of eight nodes arranged as the vertices of an octagon and connected via the octagon sides and diameters. Octagon has the properties that any two nodes can be reached in two hops, and that more octagons can be connected by sharing an end node. The Nostrum network is a two-dimensional indirect mesh network (i.e., with routers separate from end nodes) [19]. The network performs the routing function and acts as the network interface for each node processor as well. Another example of a recent micro-network is SPIN [12], which is also an indirect network, and is built with a 4-ary fat-tree topology.

Communication in any given architecture is regulated by protocols, which are designed in layers. We analyze next specific issues related to the different layers of abstraction outlined in the micro-network stack in a bottom-up way.

13.2.1 Physical Layer

Global wires are the physical implementation of the communication channels. Physical layer signaling techniques for lossy transmission lines have been studied for a long time by high-speed board designers and microwave engineers [2,10].

Traditional rail-to-rail voltage signaling with capacitive termination, as used today for on-chip communication, is definitely not well suited for high-speed, low-energy communication on future global interconnects [10]. Reduced swing, current-mode transmission, as used in some processor-memory systems, can significantly reduce communication power dissipation while preserving speed of data communication.

Nevertheless, as the technology trends lead us to use smaller voltage swings and capacitances, the upset probabilities will rise. Thus, the trend toward faster and lower-power communication may decrease reliability as an unfortunate side effect. Reliability bounds as voltages scale can be derived from theoretical (entropic) considerations [14] and can be measured by experiments on real circuits.

We conjecture that a paradigm shift is needed to address the aforementioned challenges. Current design styles consider wiring-related effects as undesirable parasitics, and try to reduce or cancel them by specific and detailed physical design techniques. It is important to realize that a well-balanced design should not over-design wires so that their behavior approaches an ideal one, because the corresponding cost in performance, energy-efficiency, and modularity may be too high. Physical layer design should find a compromise between competing quality metrics and provide a clean and complete abstraction of channel characteristics to micro-network layers above.

13.2.2 Data Link, Network, and Transport Layers

The data-link layer abstracts the physical layer as an unreliable digital link, where the probability of bit upsets is nonnull (and increasing as technology scales down). Furthermore, reliability can be traded off for energy [14]. The main purpose of data-link protocols is to increase the reliability of the link up to a minimum required level, under the assumption that the physical layer by itself is not sufficiently reliable.

An additional source of errors is contention in shared-medium networks. Contention resolution is fundamentally a nondeterministic process, because it requires synchronization of a distributed system, and for this reason it can be considered as an additional noise source. Generally, nondeterminism can be virtually eliminated at the price of some performance penalty. For instance, centralized bus arbitration in a synchronous bus eliminates contention-induced errors, at the price of a substantial performance penalty caused by the slow bus clock and by bus request/release cycles.

Future high-performance, shared-medium on-chip micro-networks may evolve in the same direction as high-speed local area networks, where contention for a shared communication channel can cause errors, because two or more transmitters are allowed to concurrently send data on a shared medium. In this case, provisions must be made for dealing with contention-induced errors.

An effective way to deal with errors in communication is to packetize data. If data is sent on an unreliable channel in packets, error containment and recovery is easier, because the effect of errors is contained by packet boundaries, and error recovery can be carried out on a packet-by-packet basis. At the data link layer, error correction can be achieved by using standard error correcting codes (ECC) that add redundancy to the transferred information. Error correction can be complemented by several packet-based error detection and recovery protocols. Several parameters in these protocols (e.g., packet size and number of outstanding packets) can be adjusted depending on the goal to achieve maximum performance at a specified residual error probability and/or within given energy consumption bounds.

At the network layer, packetized data transmission can be customized by the choice of switching and routing algorithms. The former establishes the type of connection while the latter determines the path followed by a message through the network to its final destination. Popular packet switching techniques include store-and-forward, virtual cut-through, and wormhole. When these switching techniques are implemented in on-chip networks, they will have different performance metrics along with different requirements on hardware resources.

- Store-and-forward (SAF) routing inspects each packet's content before forwarding it to the next stage. While SAF enables more elaborated routing algorithms, (e.g., content-aware packet routing), it introduces extra packet delay at every router stage. Furthermore, SAF also requires a substantial amount of buffer spaces because the switches need to store multiple complete packets at the same time. Because on-chip storage resources (i.e., static random access memory (SRAMs) and dynamic random access memory (DRAMs)) are very expensive in terms of area and energy consumption, SAF approaches are not appropriate for on-chip communications.
- Virtual cut-through (VCT) routing can forward a packet to the next stage before its entirety is received by the current switch. Therefore, VCT switching reduces the store-and-forward delays. When the next stage switch is not available, however, the entire packet still needs to be stored in the buffers of the current switch.
- Wormhole routing was originally designed for parallel computer clusters [11] because it achieves the minimal network delay and requires fewer buffers. In wormhole routing, each packet is further segmented into flits (flow control unit). The header flit reserves the routing channel of each switch, the body flits will then follow the reserved channel, and the tail flit will later release the channel reservation.

One major advantage of wormhole routing is that it does not require the complete packet to be stored in the switch while waiting for the header flit to route to the next stages. Wormhole routing not only reduces the store-and-forward delay at each switch, but it also requires much less buffer space. Because of these advantages, wormhole routing is an ideal candidate switching technique for on-chip interconnect networks [7].

In wormhole routing, one packet may occupy several intermediate switches at the same time. Thus, it may block the transmission of other packets. Deadlock and livelock are the potential problems in wormhole routing schemes [9,11].

At the transport layer, algorithms deal with the decomposition of messages into packets at the source and their assembly at destination. Packetization granularity is a critical design decision, because the behavior of most network control algorithms is very sensitive to packet size. Packet size can be application-specific in SoCs, as opposed to general networks. In general, flow control and negotiation can be based on either deterministic or statistical procedures. Deterministic approaches ensure that traffic meets specifications, and provide hard bounds on delays or message losses. The main disadvantage of deterministic techniques is that they are based on worst cases, and they generally lead to significant under-utilization of network resources. Statistical techniques are more efficient in terms of utilization, but they cannot provide worst-case guarantees. Similarly, from an energy viewpoint, we expect deterministic schemes to be more inefficient than statistical schemes, because of their implicit worst-case assumptions.

13.2.3 Software Layers

Current and future systems on chip will be highly programmable, and therefore their power consumption will critically depend on software aspects. Software layers comprise system and application software. The system software provides us with an abstraction of the underlying hardware platform, which can be leveraged by the application developer to exploit the hardware's capabilities safely and effectively.

Current SoC software development platforms are mostly geared toward single microcontroller with multiple coprocessor architectures. Most of the system software runs on the control processor, which orchestrates the system activity and farms off computationally intensive tasks to domain-specific coprocessors. Micro-controller–coprocessor communication is usually not data-intensive (e.g., synchronization and reconfiguration information), and most high-bandwidth data communication (e.g., coprocessor–coprocessor and coprocessor–IO) is performed via shared memories and direct memory access (DMA) transfers. The orchestration activities in the micro-controller are performed via runtime services provided by single-processor real time operating systems (RTOSs) (e.g., VxWorks, Micro-OS, and Embedded Linuxes), which differentiate from standard operating systems in their enhanced modularity, reduced memory footprint, and support for real-time-scheduling and bounded time-interrupt service times.

Application programming is mostly based on manual partitioning and distribution of the most computationally intensive kernels to data coprocessors (e.g., very long instruction word (VLIW) multimedia engines, digital signal processors, etc.). After partitioning, different code generation and optimization toolchains are used for each target coprocessor and the control processor. Hand-optimization at the assembly level is still quite common for highly irregular signal processors, while advanced optimizing compilers are often used for VLIW engines and fine-grained reconfigurable fabrics. Explicit communication via shared memory is usually supported via storage classes declarations (e.g., noncacheable memory pages) and DMA transfers from and to shared memories are usually set up via specialized system calls which access the memory-mapped control registers of the DMA engines.

Even from this cursory analysis, the poor scalability in a network on chip (NoC) setting of current software abstractions and runtime environments is evident. In our view, the most critical issues are the following:

- Confining the OS onto a single centralized micro-controller is a sensible choice for small-to-medium scale and asymmetric multi-processing architectures, but this choice is bound to create a performance bottleneck and significant power overhead as architectures become more symmetric and scale up in complexity and parallelism, resulting in a significant energy inefficiency. This is because all centralized control functions and policies will require communication (often under tight real-time constrains) to all peripheral processors. Even worse, a centralized OS would need to continuously collect information on all system components to maintain an updated system state snapshot. The cost in performance and power of system control and monitoring is significant and could either lead to an over budgeting of NoC resources (e.g., dedicated control channels) if quality-of-service guarantees (e.g., bounded control message delivery delay) must be provided, or to uncertain and unreliable operation in case of a best-effort network service.
- The manual and ad hoc partitioning and workload distribution procedure is too slow and error-prone in parallel, large-scale applications and target architectures. Furthermore, the lack of communication analysis tools may lead to highly inefficient task mappings. From a performance viewpoint, a communication suboptimal task mapping leads to reduced throughput and/or high latency. The energy implications can be even more serious, because in many cases reduced performance is caused by local congestion, which is a high-occupancy condition for network resources and implies high power consumption. Thus, energy efficiency decreases quadratically (high power and low performance).
- Current programming styles are based on a shared memory paradigm, which is quite natural and well suited for tightly coupled, small-scale clusters. Unfortunately, shared memory abstraction tends to hide the cost and unpredictability of communication, which are destined to grow in an

NoC setting. Furthermore, DMA burst transfers, often advocated as a mean to increase throughput in memory transfers, do increase the risk of starvation and the variance in delivery time of short, sporadic messages. From an energy viewpoint, we need to raise the level of awareness of programmers on the energy cost in accessing shared memories, especially when a single memory is shared among many multiple processors. Such a cost is only in part due to pure memory array access. Significant overheads are associated with communication and contention resolution.

In our view, software issues are among the most critical and less understood in NoC. We believe that the full potential of on-chip networks can be effectively exploited only if adequate software abstractions and programming aids are developed to support them.

13.3 Energy-Efficient Micro-Network Design

This section delves into a few specific instances of energy-efficient, micro-network design problems. In most cases, we also outline specific solutions that have been proposed in the literature, even though it should be clear that many design issues are open and significant progress in this area is expected in the near future.

13.3.1 Physical Layer

At the physical layer, low-swing signaling is actively investigated to reduce communication energy on global interconnects [36]. In the case of a simple CMOS driver, low-swing signaling is achieved by lowering the driver's supply voltage V_{dd}. This implies a quadratic dynamic power reduction (because $P_{dyn} = KV_{dd}^2$). Unfortunately, swing reduction at the transmitter complicates the receiver's design. Increased sensitivity and noise immunity are required to guarantee reliable data reception. Differential receivers have superior sensitivity and robustness, but they require doubling the bus width. To reduce the overhead, pseudo-differential schemes have been proposed, where a reference signal is shared among several bus lines and receivers, and incoming data is compared against the reference in each receiver. Pseudo-differential signaling reduces the number of signal transitions, but it has reduced noise margins with respect to fully differential signaling. Thus, reduced switching activity is counterbalanced by higher swings and determining the minimum-energy solution requires careful circuit-level analysis.

Dynamic voltage scaling has been recently applied to busses [26,33]. In Worm et al. [33], the voltage swing on communication busses is reduced, even though signal integrity is partially compromised. Encoding techniques are used to detect corrupted data that is retransmitted. The retransimission rate is an input to a closed-loop DVS control scheme, which sets the voltage swing at a trade-off point between energy saving and latency penalty (due to data retransmission).

Another key physical-layer issue is synchronization. Traditional on-chip communication has been based on the synchronous assumption, which implies the presence of global synchronization signals (i.e., clocks) that define data sampling instants throughout the chip. Unfortunately, clocks are extremely energy-inefficient, and it is a well-known fact that they are responsible for a significant fraction of the power budget in digital integrated systems. Thus, postulating global synchronization when designing on-chip micronetworks is not an optimal choice from the energy viewpoint. Alternative on-chip synchronization protocols that do not require the presence of a global clock have been proposed in the past [3,37], but their effectiveness has not been studied in detail from the energy viewpoint.

13.3.2 Data-Link Layer

At the data-link layer, a key challenge is to achieve the specified communication reliability level with minimum energy expense. Several error recovery mechanisms developed for macroscopic networks can be deployed in on-chip micronetworks, but their energy efficiency should be carefully assessed in this context. As a practical example, consider two alternative reliability-enhancement techniques: error-cor-

recting codes and error-detecting codes with retransmission. A set of experiments involved applying error correcting and detecting codes to an AMBA bus and comparing the energy consumption in four cases [4]:

1. Original unencoded data
2. Single-error correction
3. Single-error correction and double-error detection
4. Multiple-error detection

Hamming codes were used. Note that in case 3, a detected double error requires retransmission. In case 4, using (n,k) linear codes, there are $(2^n - 2^k)$ error patterns of length n that can be detected. In all cases, some errors may go undetected and be catastrophic. Using the property of the codes, it is possible to map the mean time to failure (MTTF) requirement into bit upset probabilities, and thus comparing the effectiveness of the encoding scheme in a given noisy channel (characterized by the upset probability) in meeting the MTTF target.

The energy efficiency of various encoding schemes varies: we summarize here one interesting case, where three assumptions apply. First, wires are long enough so that the corresponding energy dissipation dominates encoding/decoding energy. Second, voltage swing can be lowered until the MTTF target is met. Third, upset probabilities are computed using a white Gaussian noise model [13]. Figure 13.2 gives the average energy per useful bit as a function of the MTTF (which is the inverse of the residual word error probability). In particular, for reliable SoCs (i.e., for MTTF = 1 year), multiple-error detection with retransmission is demonstrated as more efficient than error-correcting schemes. We refer the reader to Bertozzi et al. [4] for results under different assumptions.

Another important aspect affecting the energy consumption is the media access control (MAC) function. Currently, centralized time-division multiplexing schemes (also called centralized arbitration) are widely adopted [1,8,32]. In these schemes, a single arbiter circuit decides which transmitter accesses to the bus for every time slot. Unfortunately, the poor scalability of centralized arbitration indicates that this approach is likely to be energy-inefficient as micronetwork complexity scales up. In fact, the energy cost of communicating with the arbiter, and the hardware complexity of the arbiter itself scales up more than linearly with the number of bus masters.

Distributed arbitration schemes as well as alternative multiplexing approaches, such as code division multiplexing, have been extensively adopted in shared-medium macroscopic network, and are actively investigated for on-chip communication [34]. Research in this area is just burgeoning, however, and significant work is needed to develop energy-aware media-access-control for future micronetworks.

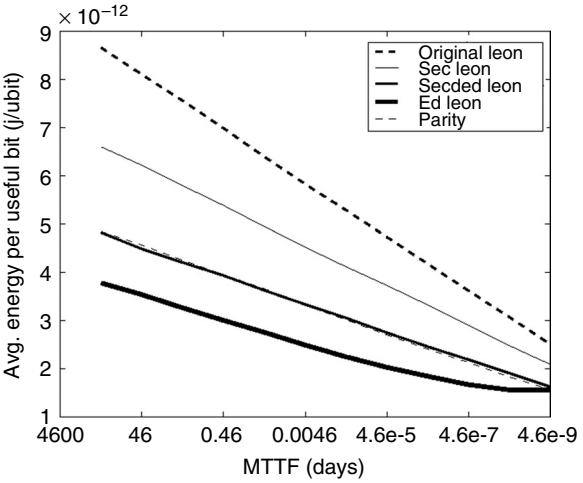


FIGURE 13.2 Energy efficiency for various encoding schemes.

13.3.3 Network Layer

Switching and routing for on-chip micro-networks affect heavily performance and energy consumption, whereas contention plays an important role. On one hand, contention delays packet transmission. On the other hand, resolving contention requires packets to be stored temporarily on the storage elements (on-chip SRAMs or DRAMs), which will increase power consumption significantly.

13.3.3.1 Contention-Look-Ahead Routing

A contention-look-ahead routing scheme is the one where the current routing decision is helped by monitoring the adjacent switches, thus possibly avoiding or reducing blockages and contention in the coming stages.

A contention-aware routing scheme is described in Nilsson [23]. The routing decision at every node is based on the "stress values" (the traffic loads of the neighbors) that are propagated between neighboring nodes. This scheme is effective in avoiding "hot spots" in the network. The routing decision steers the packets to less congested nodes.

To solve the contention problems in the wormhole routing schemes, we propose a contention-look-ahead routing algorithm that can "foresee" the contention and delays in the coming stages using a direct connection from the neighboring nodes. We use a two-dimensional mesh on-chip multiprocessor network to further explain and implement this routing algorithm. The processors are connected directly to each other in a tile-array formation, similar to that proposed by Dally and Toles [7]. Each processor tile performs packet routing and arbitration independently. The major difference from Nilsson [23] is that information is handled in flits, and thus packets with large or variable sizes can be handled with limited input buffers. Furthermore, because it avoids contention between packets and requires much less buffer usage, the proposed contention-look-ahead routing scheme can greatly reduce the network power consumption.

13.3.3.2 Wormhole Contention-Look-Ahead Algorithm

At every intermediate stage, there may be many alternate routes to go to the next stage. We call the route that always leads the packet closer to the destination a profitable route. Conversely, a route that leads the packet away from the destination is called misroute [11] (Figure 13.3). In mesh networks, profitable routes and misroutes can be distinguished by comparing the current node ID with the destination node ID.

Profitable routes will guarantee a shortest path from source to destination. Nevertheless, misroutes do not necessarily need to be avoided. Occasionally, the buffer queues in all available profitable routes are full, or the queues are too long. Thus, detouring to a misroute may lead to a shorter delay time. Under these circumstances, a misroute may be more desirable.

Any packet entering an intermediate switch along a path finds a set C of output channels to exit. As an example, for a two-dimensional mesh, C = {North, South, East, West}. We further partition C into profitable routes P and misroutes M. We define the buffer queue length of every profitable route $p \in P$ as Q_p. Similarly, we define the buffer queue length of every misroute $m \in M$ as Q_m.

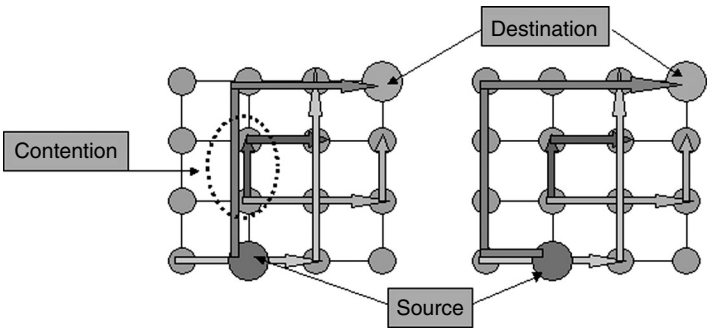


FIGURE 13.3 Profitable route and misroute.

Assume the flit delay of one buffer stage is D_B, and the flit delay of one switch stage is DS. The delay penalty to take a profitable and a misroute is defined as D_{profit} and $D_{misroute}$, respectively, in the following equation (Equation 13.1).

$$D_{profit} = min(D_B \times Q_p) \forall p \in P \tag{13.1}$$

$$D_{misroute} = min(D_B \times Q_m + 2D_S) \forall m \in M \tag{13.2}$$

In a mesh network, when a switch routes a packet to a misroute, the packet moves away from its destination by one switch stage. In the subsequent routing steps, this packet needs to get back on track and route one more stage back toward its destination. Therefore, the delay penalty for a misroute is $2 \times D_S$. The delay D_S can be estimated beforehand, and, without loss of generality, we assume the same D_S value for all switches in the network.

If all profitable routes are available and waiting queues are free, the packet will use profitable routing decision. If the buffer queues on all of the profitable routes are full or the minimum delay penalty of all the profitable routes is larger than the minimum penalty of the misroutes, it is more desirable to take the misroute (Equation 13.3):

$$(D_{profit} \le D_{misroute})(Q_p \le Q_{p_{max}} \forall p \in P)?ProRoute:Misroute \tag{13.3}$$

where Q_{pmax} is the maximum buffer queue length (buffer limit). This routing algorithm is heuristic, because it can only "foresee" one step ahead of the network. It provides a local best solution but does not guarantee the global optimum.

13.3.3.3 Network Power Consumption

This routing scheme was simulated with RSIM, a multiprocessor instruction level simulator, using 16 reduced instruction set computer (RISC) processors connected in a 4×4 (4-ary 2-cube) mesh network. Control wires that deliver the input queue information to the adjacent switches also connect adjacent processors.

The contention-look-ahead routing algorithm is compared with dimension-ordered routing — a routing scheme that always routes the packets on one dimension first, upon reaching the destination row or column, then switch to the other dimension until reaching the destination. Dimension-ordered routing is deterministic and guarantees shortest path, but it cannot avoid contention. The comparison is performed on four benchmarks: quicksort, fft, lu, and sor. These benchmarks are ported from Stanford SPLASH suite [27] and running on the RSIM simulation platform.

On-chip network power consumption comes from three contributors:

1. The interconnect wires
2. The buffers
3. The switch logic circuits

A network power consumption estimation technique is proposed by Ye et al. [35], and we will use it in the experiments.

The contention-look-ahead routing will reduce the power consumption on the buffers because it can "foresee" the contention in the forthcoming stages and shorten the buffer queue length. Figure 13.4(a) presents the averaged buffer power reduction of different benchmarks. The reduction is more significant under larger buffer sizes. This is because larger buffers will consume more power, and the power consumption is more sensitive with contention occurrence.

The power consumption on the interconnect can be estimated by counting the average number of hops a packet travels from source to destination. Dimension-ordered routing always steers the packets along the shortest path. In comparison, our proposed routing scheme may choose the misroute when contention occurs. Therefore, the contention-look-ahead routing has larger average hop count per

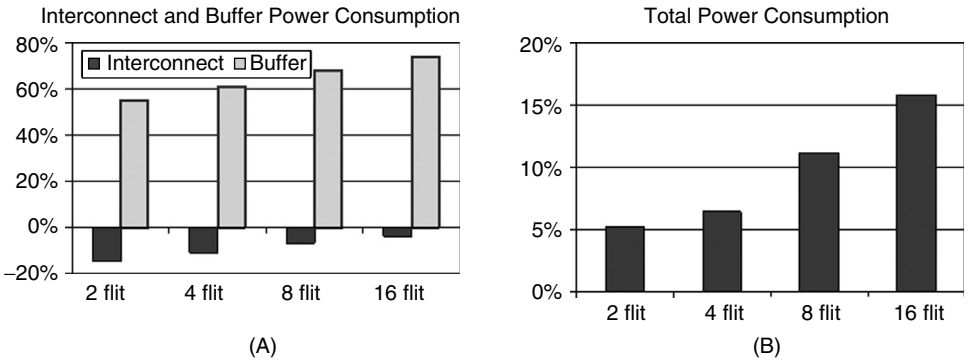


FIGURE 13.4 Power consumption comparison on interconnect wires and buffers.

packet than the dimension-ordered routing, and consequently consumes more power on the interconnects. Figure 13.4(A) depicts the proposed routing scheme consumes more power (presented as negative values) with smaller buffer size, this is because smaller buffer sizes will cause more contention and induce more misroutes.

The contention-look-ahead routing switch needs more logic gates than dimension-ordered routing. From synopsys power compiler simulation, the proposed switch circuit consumes about 4.8% more power than dimension-ordered switch. Combining the power consumption on the interconnects and buffers, the total network power consumption is depicted in Figure 13.4(B). It presents the total network power reduction compared with dimension-ordered routing. The reduction is more significant with larger buffer sizes (15.2% with 16-flit buffers).

13.3.3.3.1 Transport Layer

Above the network layer, the communication abstraction is an end-to-end connection. The transport layer is concerned with optimizing the usage of network resources and providing a requested quality of service. Clearly, energy can be considered as a network resource or a component in a quality-of-service metric. An example of transport-layer design issue is the choice of information decomposition into packets or flits, as well as the choice of packet size. Energy efficiency can be heavily impacted by this decision. Next, we will use the shared-memory multi-processor system on chip (MPSoC) as a case study to analysis the packet size trade-offs both qualitatively and quantitatively.

A typical shared-memory MPSoC architecture is illustrated in Figure 13.5. The MPSoC power consumption originates from three sources:

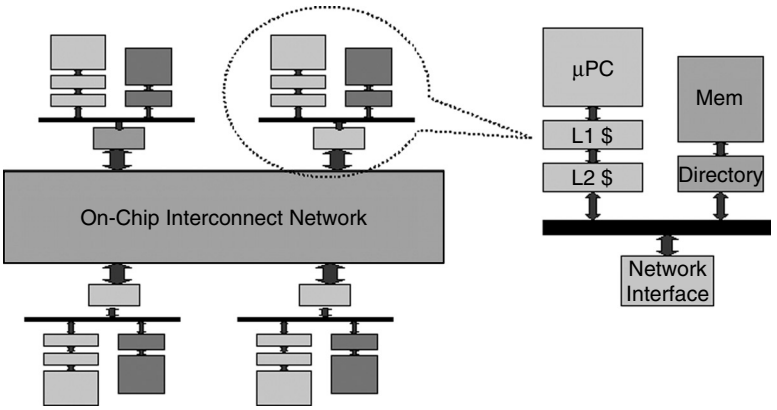


FIGURE 13.5 MPSoC architecture.

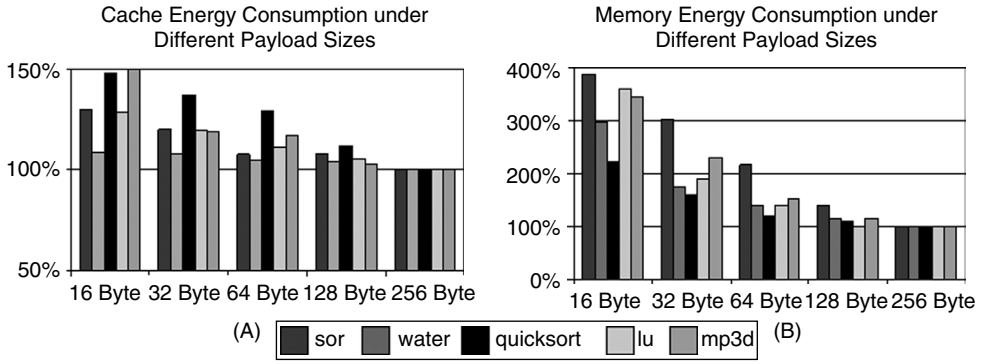


FIGURE 13.6 Cache and memory energy decrease as packet payload size increases.

1. The node processor power consumption
2. The cache and shared memory power consumption
3. The interconnect network power consumption

We will start first from the cache and memory analysis.

13.3.3.4 Cache and Memory Power Consumption

Whenever there is a cache miss, the cache block content needs to be encapsulated inside the packet payload and sent across the network. In shared-memory MPSoC, the cache block size correlates with the packet payload size. Larger packet sizes will decrease the cache miss rate, because more cache content can be updated in one memory access. Consequently, both cache energy consumption and memory energy consumption will be reduced. This relationship can be observed in Figure 13.6. It depicts the energy consumption by cache and memory under different packet sizes. The energy in the figure is normalized to the value of 256 Bytes, which achieves the minimum energy consumption.

13.3.3.5 Interconnect Network Power Consumption

The power consumption of packetized dataflow on MPSoC network is determined by three factors. The effects of these factors are summarized and listed next:

1. The number of packets on network. Packets with larger payload size will decrease the cache miss rate and consequently decrease the number of packets on the network. This effect can be observed in Figure 13.7(A). It gives the average number of packets on the network (traffic density) at one clock cycle. As the packet size increases, the number of packets decreases accordingly.

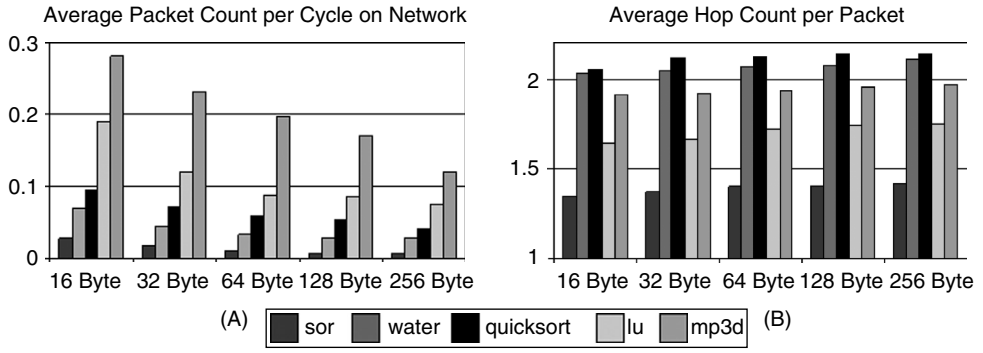


FIGURE 13.7 Packet count and hop count per packet under different payload sizes.

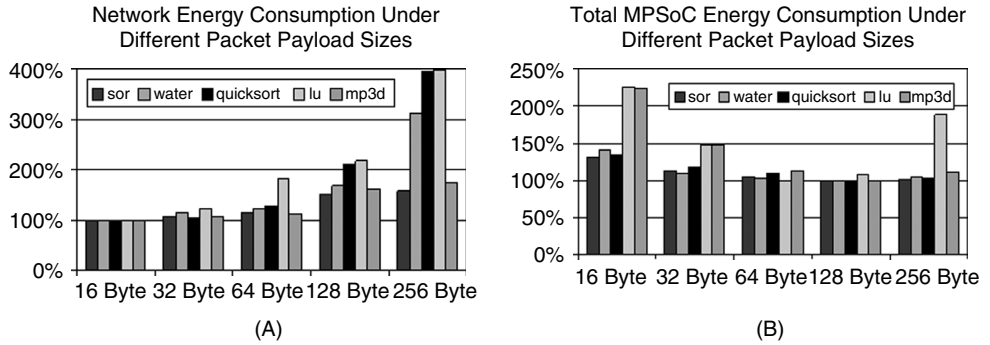


FIGURE 13.8 Network and total MPSoC energy consumption under different packet payload sizes.

2. The energy consumed by each packet on one hop. Larger packet size will increase the energy consumed per packet, because there are more bits in the payload.
3. The number of hops each packet travels. Larger packets will occupy the intermediate node switches for a longer time, and cause other packets to be rerouted to longer paths. This leads to more contention that will increase the total number of hops needed for packets traveling from source to destination. Figure 13.7(B) illustrates the effect. As packet size (payload size) increases, average hop count per packet increases as well.

Actually, increasing the cache block size will not decrease the cache miss rate proportionally. Therefore, the decrease of packet count cannot compensate for the increase of energy consumed per packet caused by the increase of packet length. Larger packet size also increases the hop counts on the datapath. Figure 13.8(A) presents the combined effects of these factors. The values are normalized to the measurement of 16 Bytes. As packet size increases, energy consumption on the interconnect network will increase.

The total energy dissipated on MPSoC comes from noncache instructions (instructions that do not involve cache access) of each node processors, the caches and the shared memories as well as the interconnect network. The overall results are given in Figure 13.8(B). From this figure, we can see that the total MPSoC energy will decrease as packet size increases. When the packets are too large, however, as in the case of 256 Bytes in the figure, the total MPSoC energy will increase. This is because when the packet is too large, the increase of interconnect network energy will outgrow the decrease of energy on cache and memories. In our simulation, the noncache instruction energy consumption does not change significantly under different packet sizes.

13.3.3.5.1 Application and System Layer

As hinted in Section 13.2, software layers are critical for the NoC paradigm shift, especially when energy efficiency is a requirement. As outlined in the previous sections, NoCs have the potential for overcoming many of the energy bottlenecks of current integrated architectures (i.e., globally shared communication and storage blocks), but only if programming abstractions, development tools, and system software help programmers understand communication-related costs and how to cope with them.

From a high-level application viewpoint, multi-processor SoC platforms can be viewed as networks of computing nodes equipped with local storage. Computation and storage are highly energy efficient if confined to the local resources within a node. Communication cost should be made explicit throughout all steps of the code development flow. Software analysis tools should help designers in identifying communication bottlenecks and code optimizers should heavily emphasize communication cost reduction. Many effective techniques have been devised in the area of parallel programming for large-scale supercomputers, and there is good potential for leveraging these experiences. It is important, however, to point out three key differences:

1. Target MPSoC architectures are much more heterogeneous than general-purpose parallel computers.

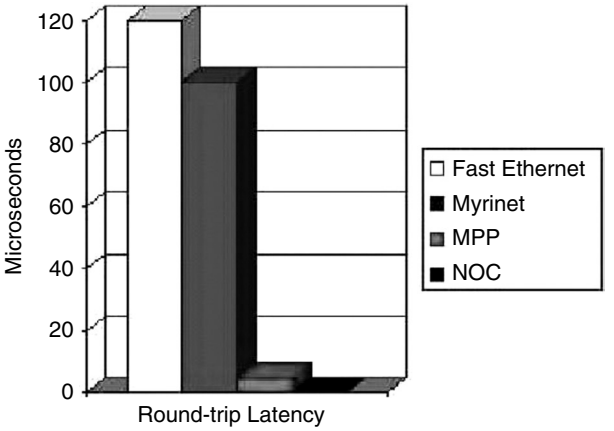


FIGURE 13.9 Interconnect latency for different parallel machines.

2. Physical link latency, albeit significant, is not nearly as dominant in on-chip networks as it is in macroscopic parallel computers (see Figure 13.9).
3. Energy constraints are extremely tight, while they have never been significant in traditional parallel machines.

The following paragraphs outline four critical areas for the evolution of energy efficient software layers in current and future NoCs. We survey seminal contributions and identify critical needs.

1. Programming abstractions. Developing adequate abstractions for NoC programming is a critical objective. Dataflow programming abstractions, such as streams [18] and Kahn networks [22], are based on a model of computation that matches very well NoC architectures. With these abstractions, communication is made explicit starting from the early steps of application development, because data flow is explicitly represented. At these levels, energy efficiency can be pursued by minimizing redundant communication, and by carefully balancing local computation and communication costs. A critical need in this area is the definition of hardware platform dependent high-level metrics, such as energy per local operation and energy per transmitted bit, which can help in first-cut exploration of the communication vs. computation trade-off during algorithm development. Unfortunately, even though many digital signal processing and multimedia applications are developed starting from dataflow models, numerous legacy applications use more traditional programming styles, where tasks are not clearly decoupled and communication is implicitly performed through memory. Leveraging the existing code basis, without compromising performance and energy efficiency, is today an open challenge.
2. Task-level analysis and optimization. A number of interesting opportunities are open for high-level optimization tools, which can help designers mapping data-flow specifications onto target hardware platforms. Consider, for instance, task splitting and merging (i.e., distributing the computation performed by a task among two or more computational nodes and collapsing two or more tasks onto the same node), task allocation, as well as communication splitting and merging over available physical NoC links. Even though a few of these problems have been explored in preliminary works [16,25], we critically need high-level energy models and analysis tools to explore techniques for increasing energy efficiency. It is important to acknowledge that energy-optimal solutions can differ significantly from performance-optimal ones in this context. Consider, for instance, a situation where available computational resources are used to achieve marginal performance benefits (e.g., a task is speculatively executed), at a price of significantly increased power consumption.

3. Code optimization. Classical code optimization, at the single task level, will still hold an important place in future NoC software development. In this area we view as critical further developments of two types of code optimizers: tools for parallelism extraction from a single task or a legacy applications [31]; tools that reduce the memory footprint and improve access locality for both code and data [24]. The first class of tools represents enabling technology that enables the reuse of legacy software as well as task splitting. The second class has a critical role in reducing "implicit" communication (as opposed to "explicit" data flow communication) from/to large background memories, which ensues because of large working sets that do not fit into local node memory. Another critical area in code optimization is the development of highly efficient communication primitives, possibly with significant dedicated hardware support. The latency and energy consumption associated with software handling of communication primitives (e.g., message send or receive) in traditional parallel programming libraries are simply unacceptable in an NoC setting [5].
4. Distributed operating systems. Intuitively, the operating system support NoC operation cannot be centralized. Truly distributed embedded OSes are required [5,6] to create a scalable runtime system. In addition to traditional functions (i.e., scheduling, interrupt handling), the NoC OS should natively support power management. End-nodes (processing elements) in SoC micronetworks will most likely be power-manageable "voltage islands" [20], with individually controllable clock speeds and supply voltages. One of the key tasks of the system software will be to control the voltage islands power states. We can envision a network-centric approach, where components send messages to neighbors to request state changes [21]. Such requests are originated and serviced at the system software levels. For example, an image processor can be required to raise its service levels before receiving a stream of data. In this case, the system software supports policies that accept requests from other components and perform transitions according to such requests.

13.4 Conclusions

The challenges of designing SoCs in 50- to 100-nm technologies available in the second part of this decade include coping with design complexity and providing reliable, high-performance operation and minimizing energy consumption. Starting from the observation that interconnect technology will be the limiting factor for achieving the operational goals, we envisioned a communication-centric view of design. We focused on energy efficiency issues in designing the communication infrastructure for future SoCs. We described several open problems at various layers of the communication stack, and we outlined basic strategies to effectively tackle the energy efficiency challenge for on-chip communication networks.

References

[1] P. Aldworth, System-on-a-chip bus architecture for embedded applications, *IEEE Int. Conference on Comput. Design,* pp. 297–298, 1999.
[2] H. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI,* Addison-Wesley, Reading, MA, 1990.
[3] W.J. Bainbridge and S.B. Furber, Delay insensitive system-on-chip interconnect using 1-of-4 data encoding, *IEEE Int. Symp. on Asynchronous Circuits and Syst.,* pp. 118–126, March 2001.
[4] D. Bertozzi, L. Benini, and G. De Micheli, Low power error resilient encoding for on-chip data busses, *Proc. Int. Conf. on Design and Test Europe,* Paris, France, pp. 102–109, March 2000.
[5] D. Bertozzi, F. Poletti, L. Benini, and A. Bogliolo, Performance analysis of arbitration policies for SoC communication architectures, *J. Design Automation for Embedded Sys.,* Vol. 8, pp. 189–210, June–Sept. 2003.
[6] W.O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, and A.A. Jerraya, Multiprocessor SoC platforms: a component-based design approach *IEEE Design andTest of Comput.,* Vol. 19, No. 6, Nov.–Dec., 2002
[7] W. Dally and B. Towles, Route packets, not wires: on-chip interconnection networks, *Proc. 38th Design Automation Conf.,* pp. 684–689, June 2001.

[8] B. Cordan, An efficient bus architecture for system on chip design, *IEEE Custom Integrated Circuits Conf.,* pp. 623–626, May 1999.
[9] W.J. Dally and H. Aoki, Deadlock-free adaptive routing in multicomputer networks using virtual channels *IEEE Trans. on Parallel and Distributed Syst.,* pp. 466–475, April 1993.
[10] W. Dally and J. Poulton, *Digital System Engineering,* Cambridge University Press, New York, 1998.
[11] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach,* IEEE Computer Society Press, Washington, D.C., 1997.
[12] P. Guerrier and A. Greiner, A generic architecture for on-chip packet-switched interconnections, *Proc. Int. Conf. on Design Automation and Test in Europe,* pp. 250–256, March 2000.
[13] R. Hegde and N. Shanbhag, Toward Achieving Energy Efficiency in Presence of Deep Submicron Noise, *IEEE Trans. on VLSI Syst.,* Vol. 8, No. 4, pp. 379–391, August 2000.
[14] R. Hegde and N. Shanbhag, Toward achieving energy efficiency in presence of deep submicron noise, *IEEE Trans. on VLSI Syst.,* Vol. 8, No. 4, pp. 379–391, August 2000.
[15] R. Ho, K. Mai, and M. Horowitz, The future of wires, *Proc. IEEE,* April 2001, pp. 490–504.
[16] J. Hu and R. Marculescu, Energy-aware mapping for tile-based NOC architectures under performance constraints, *Proc. ASP Design Automation Conf.,* pp. 233–239, Jan. 2003.
[17] F. Karim, A. Nguyen, and S. Dey, On-chip communication architecture for OC-768 network processors, *Proc. 38th Design Automation Conf.,* pp. 678–683, June 2001.
[18] B. Khailany et al. Imagine: Media Processing with Streams, IE*EE Micro,* pp. 35–46, Vol. 21, No. 2, 2001.
[19] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrij, and A. Hemani, A network on chip architecture and design methodology, *Proc. IEEE Computer Society Annual Symp. on VLSI,* pp. 105–112, April 2002.
[20] D. Lackey, P. Zuchowski, T. Bednar, D. Stout, S. Gould and J. Cohn, Managing power and performance for systems on chip design using voltage islands, *ICCAD – Int. Conf. on Computer-Aided Design,* pp. 195–202, Nov. 2002.
[21] A. Laffely, J. Liang, P. Jain, N. Weng, W. Burleson, and R. Tessier, Adaptive systems on a chip (aSoC) for low-power signal processing, *35th Asilomar Conf. on Signals, Syst., and Comput.,* pp. 1217–1221, Nov. 2001.
[22] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, A methodology for architecture exploration of heterogeneous signal processing systems *J. VLSI Signal Process. for Signal, Image and Video Technol.,* Vol. 29, No. 3, pp. 197–207, 2001.
[23] E. Nilsson Design and implementation of a hot-potato switch in a network on chip, M.S. thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm, Sweden, June 2002.
[24] P. R. Panda, N. D. Dutt, A. Nicolau, F. Catthoor, A. Vandecappelle, E. Brockmeyer, C. Kulkarni, and E. de Greef, Data memory organization and optimizations in application-specific systems, *IEEE Design and Test of Comput.,* Vol. 18, No. 3, May–June 2001.
[25] A. Pinto, L Carloni, and A. Sangiovanni-Vincentelli, Constraint-driven communication synthesis,' *Design Automation Conf.,* pp. 783–788, June 2002.
[26] L. Shang, L.-S. Peh, and N.K. Jha, Dynamic voltage scaling with links for power optimization of interconnection networks, *HPCA — Proc. Int. Symp. on High-Performance Computer Architecture,* Anaheim, CA, pp. 91–102, February 2003.
[27] J.P. Singh, W. Weber, and A. Gupta, SPLASH: Stanford parallel applications for shared-memory *Computer Architecture News,* Vol. 20, No. 1, pp. 5–44, March 1992.
[28] D. Sylvester and K. Keutzer, A global wiring paradigm for deep submicron design, *IEEE Trans. on CAD/ICAS,* Vol. 19, No. 2, pp. 242–252, February 2000.
[29] T. Theis, The future of Interconnection Technology, *IBM J. Res. and Dev.,* Vol. 44, No. 3, pp. 379–390, May 2000.
[30] J. Walrand and P. Varaiya, *High-Performance Communication Networks,* Morgan Kaufman, San Francisco, 2000.

[31] M. Wolfe, *High-Performance Compilers for Parallel Computing,* Addison-Wesley, Reading, MA, 1995.
[32] S. Winegarden, A bus architecture centric configurable processor system, *IEEE Custom Integrated Circuits Conf.,* pp. 627–630, May 1999.
[33] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, An Adaptive low-power transmission scheme for on-chip networks, *ISSS, Proc. Int. Symp. on System Synthesis,* Kyoto, Japan, pp. 92–100, October 2002.
[34] R. Yoshimura, T. Koat, S. Hatanaka, T. Matsuoka, and K. Taniguchi, DS-CDMA wired bus with simple interconnection topology for parallel processing system LSIs, *IEEE Solid-State Circuits Conf.,* p. 371, January 2000.
[35] T.T. Ye, L. Benini, and G. De Micheli, Packetized on-chip interconnect communication analysis for MPSoC, *Proc. on Design Automation and Test in Europe,* pp. 344–349, March 2003.
[36] H. Zhang, V. George, and J. Rabaey, Low-swing on-chip signaling techniques: effectiveness and robustness, *IEEE Trans. on VLSI Syst.,* Vol. 8, No. 3, pp. 264–272, June 2000.
[37] H. Zhang, M. Wan, V. George, and J. Rabaey, Interconnect architecture exploration for low-energy configurable single-chip DSPs, *IEEE Computer Society Workshop on VLSI,* pp. 2–8, April 1999.

14

Highly Integrated Ultra-Low Power RF Transceivers for Wireless Sensor Networks

Brian P. Otis
Yuen Hui Chee
Richard Lu
Nathan M. Pletcher
Jan M. Rabaey
University of California—Berkeley

Simone Gambini
Universita di Pisa

14.1 Introduction

14.1.1 Motivation

Technological advances have made it conceivable to build and deploy dense wireless networks of heterogeneous nodes collecting and disseminating wide ranges of environmental data [1]. An inspired reader can easily imagine a multiplicity of scenarios in which these sensor and actuator networks might excel. To mention just a few: environmental control in office buildings, robot control and guidance in automatic manufacturing environments, warehouse inventory, integrated patient monitoring, diagnostics and drug administration in hospitals, interactive toys, the smart home providing security, identification and personalization, and interactive museums. The overwhelming opportunities emerging from this technology

TABLE 14.1 Power Density of Energy Scavenging Sources

Power Source	Power Density ($\mu W/cm^2$)	Lifetime
Lithium battery	100	1 year
Solar cell	10–15,000 (in $\mu W/cm^2$)	∞
Vibrational converter	300	∞

indeed give rise to new definitions of distributed computing and user interface. Regardless of the specific application, however, they all rely on a network of ubiquitously distributed sensor, compute, and actuation nodes, which are integrated and embedded into the fabrics of our daily living environment. This explains why the name "ambient intelligence" is often attributed to such environments [2].

Widespread deployment of wireless sensor networks requires that some economic and physical realities be met. More precisely, the physical implementation of an individual network node is constrained by three important metrics: power, cost, and size. Of these three, power (or energy, depending on how the node is powered) turns out to be the most fundamental metric. To keep cost down and to allow for a flexible deployment, most nodes must be untethered. Cost considerations also dictate that frequent replacement of the energy source of the node (especially in a ubiquitous deployment scenario) is out of the question. This leads to the general guideline that a network node must be self-sufficient from an energy perspective for the lifetime of the product. This could be multiple years for applications such as smart homes. The energy storage capability of a node is limited by the storage medium (battery or capacitor) and the size constraints [3]. Although a single-time charge could work for applications with life cycles below one year, replenishment of the energy supply using energy scavenging is often a necessity. As a result, the average power dissipation of a node is firmly capped at 1 mW. More realistically, average power dissipation levels around 100 μW are necessary given today's energy generation technologies. Table 14.1 illustrates the finite power density of state-of-the-art energy sources [3].

As listed in the table, the average power consumption of the sensor node must be very low if the energy scavenging volume is limited. From a volume of 1 cm^3, one or a combination of these power sources could supply an average continuous power output of 100 μW. Although this severely restricts the amount of processing that can be done within a node, it also determines the type of wireless connectivity that can be obtained between the nodes.

Ubiquitous deployment of these nodes is only economically feasible if the cost of the individual elements is ignorable, or, in other words, the electronics have become disposable. This translates to price points per node of less than $1. Achieving a node cost this low requires a minimal number of components, a high level of integration, simple and cheap packaging and assembly, and avoidance of any expensive components and/or technologies.

Finally, embedding the components into the daily environment (walls, furniture, clothing, etc.) further requires that the form factor of the entire sensor node must be very small. Typically, sizes smaller than $1cm^3$ are necessary. Again, a very high level of integration is mandatory if such small dimensions are to be achieved.

In the design of these sensor nodes, we have experienced that the wireless interface takes up the largest fraction of the power and size budget of the node. Although the demands of the sensing and digital processing components cannot be ignored, their duty cycle is typically very low. Exploitation of advanced sleep and power-down techniques makes it possible to make their average power dissipation virtually ignorable. Thus, in the remainder of this chapter we will focus our discussion on the design of ultra-low power wireless interfaces for wireless sensor networks. Although optical communication approaches offer the potential for very low power and small size, line-of-sight and directivity considerations make them less attractive [4]. Thus, we will limit our discussion to radio-frequency (RF) interfaces.

The previous observations demonstrate that wireless sensor nodes occupy a unique corner of the semiconductor and embedded system design space, and, in a way, push against many traditional design boundaries.

14.1.2 Characteristics of Wireless Sensor Networks

One may wonder if and why an RF interface for a sensor network should be substantially different than the one used, for instance, in a wireless data network (local area network [LAN]). In fact, it turns out that the operation mode of the sensor node is so fundamentally dissimilar that completely different optimization criteria apply. This results mostly from the traffic patterns that affect the power dissipation profile of a node. Data packets in sensor networks tend to be relatively rare and unpredictable events. In most application scenarios, each node in the network sees at most a couple of packets/sec. In addition, the packets are relatively short (typically less than 200 bits/packet), as the payloads normally represent data measurements, which typically require a resolution of less than 24 bit/measurement. Combined, this means that the average data rate of a single node rarely exceeds 1 kbit/sec. These observations are of foremost importance when designing the wireless transceiver, as we will highlight in the following sections.

In the rest of the discussion, we will assume that the sensor networks of interest are dense, which means that the nodes in the network are placed relatively closely (i.e., the average distance between nodes is less than or equal to 10 m).*

14.1.3 Performance Metrics for Sensor Node RF Transceivers

14.1.3.1 Average Power Dissipation

Traditional quality metrics for radios used in wireless LANs are the data throughput (bit/sec), spectral efficiency (bit/sec/Hz), and energy-efficiency (nJ/bit). None of these is truly important for a wireless sensor node because the required average data rates are very low. Because the bits are few and the nodes are closely spaced, the energy/bit is not an important metric either. In fact, the power used for the actual data transmission and reception is only a fraction of the total power dissipated in the front-end. This is best illustrated with a statistical power model of the transceiver [5]. At any point in time, the transceiver is in one of the following states:

- Transmitting state (TX) during transmission of data.
- Receiving state (RX) during reception of data.
- Acquiring state (AQ) while acquiring synchronization at the start of the packet.
- Monitoring state (MN) when the transceiver is monitoring the channel (carrier sense).
- The idle state (IL) when the majority of the transceiver is turned off, and it is considered to be sleeping; it may be assumed that the power dissipation in this state is zero.

The average power dissipation of the transceiver is then expressed as

$$P_{av} = p_{TX}P_{TX} + p_{RX}P_{RX} + p_{AQ}P_{AQ} + p_{MN}P_{MN} + p_{IL}P_{IL}$$

where P_x is the average power dissipation in state x and p_x the probability the transceiver is in that particular state. The power dissipation in the TX state is determined mostly by the dissipation in the power amplifier. The average power dissipation of the RF front end in the three other modes (RX, AQ, and MN) is approximately equal, although P_{AQ} may dominate slightly.

Given the low data rates and duty cycles, the transceiver should be in the idle state for most of the time, given proper sleep disciplines. Among the four other states, the monitoring state (MN) is the most probable, as became apparent from simulations based on this model. Assuming a dense network and sparse traffic, the average power of the transceiver is well approximated as

$$P_{av} = p_{TX}P_{TX} + (p_{RX} + p_{AQ} + p_{MN})P_{RXon} \cong p_{MN}P_{RXon}$$

*In networks with a lower density, TX power rapidly becomes the dominant power factor.

where P_{RXon} is the dissipation when the receiver is on. It is thus fair to state that the average power is dominated by the power of having just the RF receiver turned on (e.g., low-noise amplifier [LNA], down-converter, and synthesizer), independent of the data activity. Minimizing the average power then translates into minimizing the active current draw of the RF front end, and, obviously, the time that the transceiver is turned on. This leads to the important conclusion that simple RF transceivers with a minimal number of active components are the preferred option for use in wireless sensor networks.

If we assume a power budget of 100 μW and the RF module is allotted 20% of this power budget, at a 1% radio duty cycle, this provides the on-state power consumption goal of 2 mW for the entire RF transceiver.

14.1.3.2 Turn-On and Acquisition Time

In an environment where the radio is in idle or off mode most of the time, and where data communications are rare and packets short, it is essential that the radio can start up very quickly. For instance, a 1-Mbps radio with a 500-μs turn-on time would be poorly suited for the transmission of short packets. The on-time to send a 200-bit packet would be only 200 μs. Start-up and acquisition thus represent an overhead that is larger than the actual payload cost, and may very well dominate the power budget (given that channel acquisition is typically the most power-hungry operation).

Thus, fast start-up and acquisition is essential. An agile radio architecture that allows for quick and efficient channel acquisition and synchronization is therefore desirable. Complex wireless transceivers tend to use sophisticated algorithms such as interference cancellation and complex modulation schemes to improve bandwidth efficiency. These techniques translate into complex and lengthy synchronization procedures and may require accurate channel estimations. Packets are spaced almost seconds apart, which is beyond the coherence time of the channel. This means that these procedures have to be repeated for every packet, resulting into major overhead. Simple modulation and communication schemes are thus the desirable solution if agility is a prime requirement.

14.1.3.3 Integration and Cost

In RF circuit design, the term "fully integrated" typically refers to a transceiver that still requires an off-chip quartz crystal and a few assorted passive components. To meet the cost and form-factor requirements of this application, a true fully integrated transceiver is mandatory. In addition to increasing the size, off-chip passives add to the complexity and cost of the board manufacturing and package design.

One method that can be used to achieve a high level of integration is the use of a relatively high carrier frequency. Currently available simple low-power radios, as used in control applications, typically operate at low carrier frequencies between 100 and 800 MHz. A high carrier frequency has the distinct advantage of reducing the required values of the passive components, making integration easier. For example, a 2.53-μH inductance is needed to tune out a 1-pF capacitor in a narrow-band system at 100 MHz, requiring a surface-mount inductor. For a 2-GHz carrier frequency, the inductance needed is only 6.33 nH, which can easily be integrated on-chip using interconnect metallization layers. In addition, the antenna form-factor is very dependent upon carrier frequency. For a given antenna gain, a higher carrier frequency allows for a much smaller antenna. A quarter-wavelength monopole antenna at 100 MHz would be 0.75-m long. At 2 GHz, the size shrinks to 37.5 mm, making board-level integration or use of small chip-antennas possible. The drive to higher carrier frequencies to achieve high integration is in direct conflict with the need for low-power consumption. As the carrier frequency increases, the active devices in the RF signal path must be biased at higher cutoff frequencies, increasing the bias current and decreasing the transconductance-to-current (g_m/I_d) ratio. This results in increased power dissipation at higher carrier frequencies. Thus, an inherent integration/power consumption trade-off must be dealt with through architectural decisions and the use of new technologies.

In conclusion, RF transceivers for wireless sensor networks should be simple, consume a minimum amount of on-current, and operate at higher carrier frequencies. In the rest of the chapter, we explore how these goals can be simultaneously accomplished. The emerging technology of RF microelectromechanical systems (MEMS), which promises the availability of small highly tuned high-frequency

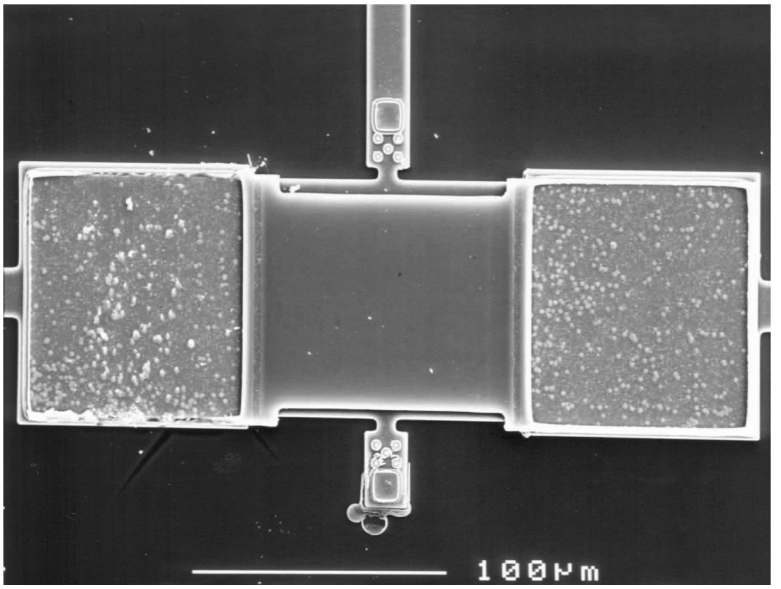


FIGURE 14.1 50-MHz capacitively driven/sensed resonator. (B. Bircumshaw, G. Liu, H. Takeuchi, T.-J. King, R. Howe, O. O'Reilly, and A. Pisano, *Tech. Dig., 12th Int. Conf. on Solid-State Sensors, Actuators, and Microsystems,* Boston, MA, pp. 875–878, June 8–12, 2003. With permission.)

passive components, offers an excellent opportunity of doing so. We commence our discussion with a description of this exciting technology. The rest of the chapter then describes how these components can be used to build power-efficient receivers and transmitters. Next, a number of low-power circuit techniques used in the implementation of these modules are described, followed by a discussion of some system integration techniques and some realized prototypes.

14.2 RF MEMS in Low-Power Radios

The relatively new field of RF MEMS provides unique opportunities to RF transceiver designers. This section provides background on RF MEMS and provides insight into the opportunities presented by these new technologies.

14.2.1 Introduction to RF MEMS

The field of RF MEMS includes the design and utilization of RF filters, resonators, switches, and other passive mechanical structures constructed using integrated circuit fabrication techniques. To date, these devices have been used as discrete board-mounted components, primarily used to enhance the miniaturization of mobile phones [6]; however, RF MEMS components have the potential to be batch fabricated using existing integrated circuit fabrication techniques. New capacitively driven and sensed structures offer the potential of integration on the same substrate as the CMOS circuitry. In addition, because the resonant frequency is set lithographically and not by a deposition layer thickness, it is possible to fabricate devices with many unique resonant frequencies on the same wafer [7]. See Figure 14.1 for an example of this technology.

The structure in Figure 14.1 was constructed of micromachined polysilicon on top of a silicon wafer. The continued improvement in the performance, reliability, and manufacturability of these structures will greatly change the performance and form-factor of RF transceivers. As discussed in this chapter, however, even in their current state, these devices hold the potential to enable new circuit blocks and architectures.

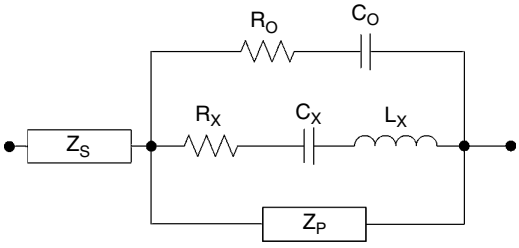


FIGURE 14.2 Simplified circuit equivalent model of MEMS resonator.

14.2.2 Opportunities Offered by RF-MEMS

14.2.2.1 Passives with High-Quality Factor

One often-cited benefit of RF MEMS structures is the ability to design resonators with very high-quality factors. As compared with integrated LC tank structures, which typically achieve Q factors of 5 to 10, RF MEMS resonators can achieve Q factors two orders of magnitude higher [8]. See Figure 14.2 for a simplified circuit equivalent model of a MEMS resonator.

In the equivalent schematic, R_x, C_x, and L_x correspond to the motional impedances of the resonator. R_o and C_o represent the finite quality factor of the feed-through capacitance, which affects the Q of the parallel resonance. Finally, Z_s and Z_p represent the impedance load on the resonator due to the CMOS circuitry. It is important to note that these additional impedances have a strong influence on the resonator characteristics, as they affect the loaded resonator quality factor, resonant frequencies, and frequency tolerance.

When used in the design of bandpass filters and duplexers, high Q resonators help to realize the steep skirts necessary to meet cell phone specifications [8]. High Q resonators are further useful in a variety of other transceiver blocks. For example, high Q resonators provide the potential for radio frequency channel select filtering, as their bandwidth is much narrower than what can be obtained from integrated LC filters. This passive channel select filtering can be exploited to simplify the receiver architecture and to reduce the number of active components [9]. In addition, when used in an RF oscillator, RF MEMS resonators provide a vastly improved phase-noise compared with a standard, low Q LC resonator [10]. A design example based on these principles is explored in Section 14.5.

14.2.2.2 Passive Frequency Reference

For all narrowband communication systems, an RF carrier frequency generator is necessary. The absolute frequency reference used is typically a low-frequency quartz crystal oscillator. A frequency synthesizer then multiplies the low-frequency sinusoid up to radio frequencies. This technique has a few disadvantages for low-power radio design. First, even for a fully integrated frequency synthesizer, an off-chip quartz crystal is always necessary, making true full integration impossible. In addition, frequency synthesizers are a huge source of power dissipation in low-power radios [11]. The VCO and frequency dividers tend to dominate the power consumption of frequency synthesizers. Radio frequency MEMS components provide an inherent high-frequency reference without the need for a power hungry frequency synthesizer.

14.2.2.3 MEMS/CMOS Codesign

One of the most exciting aspects of RF micromachined components is the potential for codesigning the MEMS devices with the CMOS circuitry. Until now, passive components included low-quality, on-chip devices (e.g., inductors, capacitors) or high-quality, off-chip components (e.g., inductors, surface acoustic wave (SAW) filters, quartz crystals, duplexers). The on-chip components allow customization to meet the requirements of the circuitry, but their performance is normally poor. High-quality, off-chip passives offer few design degrees of freedom. For example, most filters and duplexers are designed for 50-Ω input and output impedances. This rigid impedance level is very detrimental from a low-power point of view, and has been very troublesome in past receiver implementations [12]. The potential of integrating RF MEMS components and circuitry on the same die or on the same substrate using, for instance, fluidic

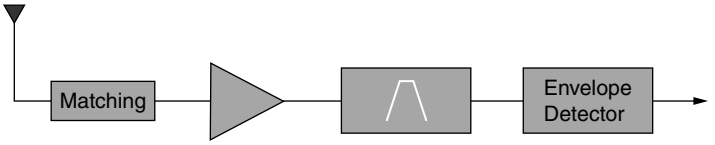


FIGURE 14.3 TRF architecture with envelope detector.

self-assembly (FSA) could allow the circuit designer to size the MEMS components and the circuitry simultaneously [13]. The ability to design these devices alongside the circuitry provides increased system performance and additional designer degrees of freedom.

Overall, the availability of high-quality passive RF components makes it possible to realize transceiver architectures with a minimal number of active components and with a minimum on-current. The following sections evaluate a number of receiver and transmitter architectures that exploit this concept.

14.3 Receivers for Ad Hoc Wireless Sensor Networks

As mentioned earlier, two main considerations in the design of the receiver are ultra-low power consumption and ultra-high integration. The choice of receiver topology has huge implications on the ability to meet these two goals. For example, although aggressive and carefully optimized circuit design can provide low-power consumption, some radio architectures inherently require more active devices biased at higher cutoff frequencies and more off-chip components than other architectures. This section discusses various architectures that can be considered for the implementation of such a radio.

14.3.1 Heterodyne

The omnipresent heterodyne architecture is often the first one to be considered. The process of down-converting the signal allows high gain to be placed at the intermediate frequency, overcoming the noise of the detector and reducing the risk of instabilities. In addition, heterodyning allows channel select filtering to take place at low frequencies, easing the implementation complexity and increasing the potential to integrate these filters. Various flavors of this architecture include high-IF, low-IF, and direct conversion, where the signal is down-converted directly to DC. Regardless of the choice of intermediate frequency, however, an accurate RF frequency reference is needed to drive the mixer in this architecture. Most of the time, this local oscillator (LO) signal is generated from a reference crystal oscillator through a frequency synthesizer. Thus, architectures that eliminate this frequency synthesizer and reduce the number of active devices biased at a high f_t are more appropriate for ultra-low power design.

14.3.2 Tuned Radio Frequency

The tuned radio frequency (TRF) architecture, one of the simplest receiver architectures, eliminates the RF frequency synthesizer and mixers by filtering in the RF domain and directly detecting the RF signal [14]. This architecture relies on sharp RF filters with high-frequency stability — two requirements that are met by RF MEMS components described earlier. The filtered signal can be detected and down-converted in a variety of ways, including envelope detection and subsampling.

14.3.2.1 TRF Envelope Detection

This method of detection, also referred to as diode detection, performs a self-mixing operation on the signal. The RF signal drives a nonlinear element such as a diode or envelope detector, providing a DC component containing the signal spectrum of interest, as depicted in Figure 14.3.

After down-conversion via envelope detection, the signal is simply low-pass filtered to remove the fundamental and higher harmonics, leaving only the baseband signal. One main disadvantage of this approach is that a large RF input signal must be present to induce this self-mixing, as there is no separate

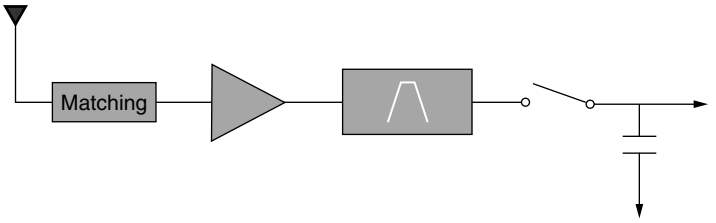


FIGURE 14.4 TRF architecture with subsampling detector.

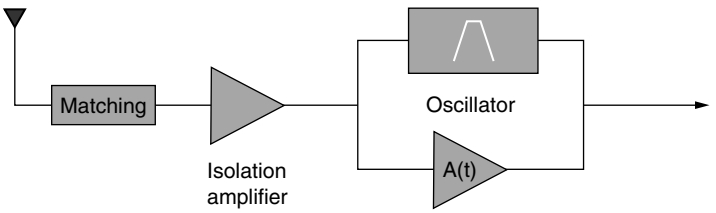


FIGURE 14.5 Super-regenerative detection.

local oscillator driving the system into nonlinearity. Thus, the sensitivity of the detector is poor, and a large RF signal is necessary to achieve detection. The inherently poor noise figure of this detector thus necessitates high RF gain to overcome the noise of the detector.

14.3.2.2 TRF Subsampling Detection

The concept of subsampling overcomes the need for the self-mixing operation needed for envelope detection. As depicted in Figure 14.4, a subsampling architecture samples the signal directly at RF frequencies.

As the name implies, the sampling rate of this detector does not satisfy the Nyquist criterion for sampling the RF signal, and aliasing of the signal occurs. This aliasing effect down-converts the signal to DC, where it is converted to the digital domain by an A/D converter. The sampling rate must only satisfy the Nyquist sampling criterion of the baseband signal. Unfortunately, all noise and sources of interference are also aliased into the baseband bandwidth. Even with very sharp RF filters, the noise of the sampler is problematic. Although the signal is sampled at a low rate, the sampler must track the RF signal, so its bandwidth must be high. This high-bandwidth requirement translates into a relatively small sampling capacitor, which means that the sampler itself is a source of high KT/C noise. Similar to the envelope-detector approach, overcoming the poor noise figure of the subsampler requires high RF gain.

In conclusion, the TRF architecture is promising in its simplicity and reduction of active circuit blocks. They have the advantage that all mixers and synthesizers have been eliminated. The challenge of integrating the steep RF filters is also well suited for RF MEMS technologies. The detection is noisy, however, requiring high gain in the RF amplification stages. This necessitates multiple gain stages biased at high cutoff frequencies (f_t), resulting in high power dissipation in the RF amplifiers.

14.3.3 Super-Regenerative

As discussed in the previous section, the TRF architecture takes advantage of RF MEMS technologies to perform channel selection without a need for mixers or frequency synthesizers. The RF gain needed, however, is very high due to the noisy detection circuitry. A super-regenerative front-end provides extremely high RF amplification and narrowband filtering at low bias-current levels. As depicted in Figure 14.5, the heart of a super-regenerative detector is an RF oscillator with a time-variant loop gain. The isolation amplifier between the antenna and the oscillator performs the following functions: it prevents radiation of the oscillation to the antenna, it provides an input match to the antenna, and it injects the RF input signal current into the oscillator tank without adding significant loading to the oscillator.

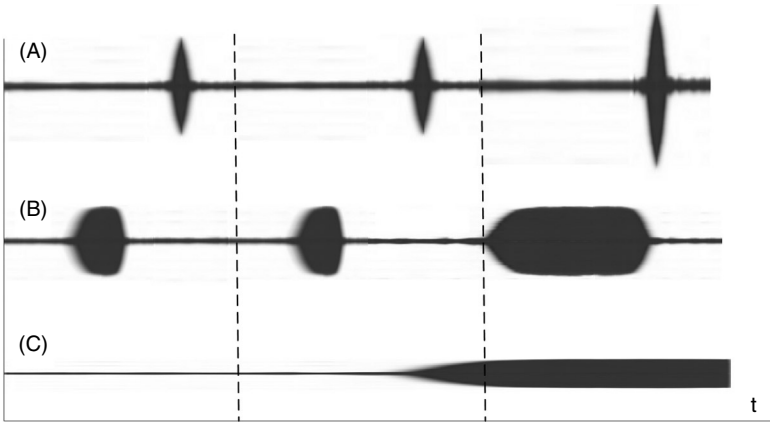


FIGURE 14.6 Super-regenerative detector waveforms.

The time-varying nature of the loop-gain is designed such that the oscillator transconductance periodically exceeds the critical g_m necessary to induce instability. Consequently, the oscillator periodically starts up and shuts off. The start-up time of an oscillator is:

$$t_{rise} = \tau_{rise} \ln\left[\frac{V_{osc}}{V_{initial}}\right]$$

where τ_{rise} is the time constant of the exponentially increasing oscillation envelope, V_{osc} is the zero-peak RF voltage of the saturated oscillator, and $V_{initial}$ is the zero-peak RF signal when the oscillator loop gain is unity (at the onset of oscillation). As this equation demonstrates, the start-up time of the oscillator is exponentially dependent upon the initial voltage in the oscillator tank. This dependency translates into the huge gain attainable by the super-regenerative receiver.

There exists two basic modes of operation that can be utilized: the logarithmic mode or the linear mode [15], as is illustrated in Figure 14.6.

Waveforms (a), (b), and (c) illustrate the detector output in the linear mode, the output in the logarithmic mode, and the RF input signal, respectively. In the linear mode, the level of oscillation is measured before the oscillator reaches saturation, providing a high signal independent gain. As illustrated in waveform (a), the sampled envelope is much larger in the presence of an RF input signal. In the logarithmic mode, detection circuitry senses the area under the oscillation envelope, providing signal dependent gain. Waveform (b) depicts the increased area under the saturated oscillation envelope in the presence of an RF input, resulting from the decreased oscillator start-up time in this condition. Due to the severe fading anticipated in dense indoor sensor networks, a very wide dynamic range is required from the receiver. The logarithmic mode provides an inherent automatic gain control, making its use preferable for this application.

The potential of the super-regenerative receiver to generate large signal gain at very low bias currents makes it the preferred architecture for integrated ultra-low power wireless receivers.

14.4 Transmitters for Ad Hoc Wireless Sensor Networks

As mentioned in Section 14.1, the environment of an ad hoc wireless sensor network differs significantly from those in a conventional wireless network (e.g., GSM, CDMA, Wireless LAN, and Bluetooth). In a typical sensor network, the transmitter sends out sporadic bursts of short data packets to neighboring sensor nodes (<10 m) [9]. This implies the need for:

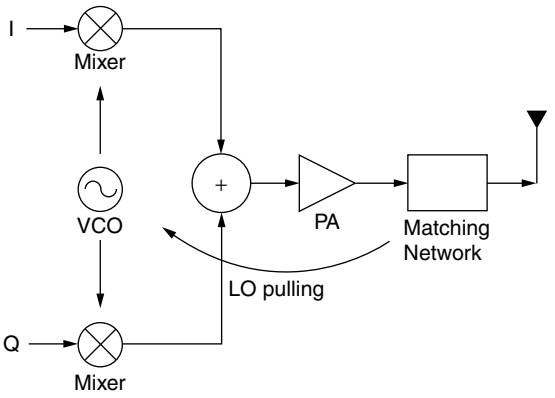


FIGURE 14.7 Direct conversion transmitter architecture.

1. Power shutdown of the transmitter when idle
2. Short turn-on time of the transmitter to minimize overhead power
3. A low transmit power of about 0 dBm (1 mW) for typical receiver sensitivity and indoor multipath fading conditions

In addition, all the energy dissipated by the transmitter is scavenged from the environment. As scavenged energy is the most precious resource in a sensor node, the efficiency of the transmitter when switched on must be maximized. To put this in into perspective, consider an RF transceiver with an on-state power consumption goal of 2 mW (see Section 14.1.3). An on–off keyed (50% duty cycled) transmitter with 25% efficiency will require 2 mW of power, consuming the entire power budget allocated for the transceiver. A more reasonable division of the power budget between transmitter and receiver is 70% and 30%, respectively. This translates to a required transmitter on-state efficiency of around 36%. Note that this is the required global efficiency of the transmitter, not just the power amplifier.

These requirements have a profound impact on the architecture and implementation of the transmitter for wireless sensor network. In this section, we compare various transmitter architectures and propose a transmitter amenable to low-power ad hoc sensor networks. The low-power design techniques employed in the implementation are discussed in Section 14.5.

14.4.1 Direct-Conversion Transmitter

The direct conversion and the two-step transmitter architectures are most commonly used in conventional radios (e.g., GSM, CDMA, Bluetooth, Wireless LAN) [16]. In the direct conversion transmitter, illustrated in Figure 14.7, the baseband signal is up-converted directly to RF in the modulator, and then efficiently boosted to the required power level by the power amplifier and matching network. The direct conversion transmitter suffers from local oscillator (LO) pulling, in which the output power from the power amplifier leaks into the local oscillator, corrupting the clean LO signal.

14.4.2 Two-Step Transmitter

The effect of LO pulling is alleviated if a two-step conversion as illustrated in Figure 14.8 is employed. The baseband signal is first up-converted to an IF signal, and then converted to the required RF signal in a second mixing step. In this case, the frequency of the oscillator and the RF carrier are different, and LO pulling is reduced.

A careful analysis of the preceding transmitters reveals that they are not ideally suited for sensor applications for two main reasons:

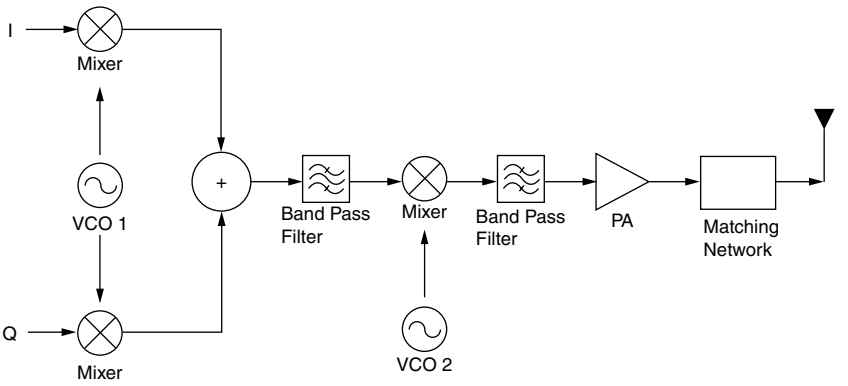


FIGURE 14.8 Two-step transmitter.

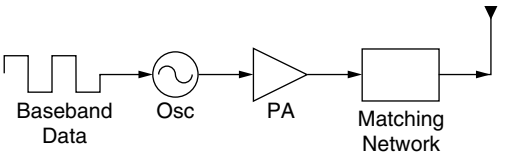


FIGURE 14.9 Direct modulation transmitter.

1. High data rate/bandwidth/spectral efficiency. Conventional radios employ complex modulation schemes (using both AM and PM) to maximize spectral efficiency. Given that the accumulated traffic is approximately 1 kbit/sec/node, maximizing spectral efficiency is not a primary goal. Advanced modulation schemes require the use of a linear power amplifier, which generally have low efficiency. The relaxed requirement on spectral efficiency and linearity means that constant envelope modulation schemes can be employed, which allow for the use of high-efficiency, nonlinear power amplifiers.
2. High transmit power. Conventional radios transmit hundreds of mW to 1W, and LO pulling is an important concern; however, the transmitted power from a sensor node is about 1 mW. With an isolation of 20 dB to 30 dB between the power amplifier (PA) and the local oscillator, the leakage power ranges between 10 μW to 1 μW, which is insignificant compared with the LO's output power. Thus, LO pulling is not a significant issue, and the two-step approach is definitely overkill.

14.4.3 Direct-Modulation Transmitter

Taking advantage of the unique characteristics of a wireless sensor network, the direct modulation transmitter in Figure 14.9 is very attractive.

In this architecture, the oscillator is directly modulated by the baseband data (on–off keying) and the nonlinear PA and matching network efficiently boosts the power of the RF signal. The direct-modulation transmitter architecture lends itself to an ultra-low power transmitter for the following reasons:

- Direct modulation of the oscillator eliminates power-hungry mixers. This is enabled by the use of a simple on–off keying modulation scheme. This is deemed acceptable due to the relaxed spectral efficiency requirements of the sensor network.
- On–off keying allows the use of nonlinear high efficiency power amplifiers.
- On–off keying eliminates the use of quadrature channels, thus reducing the number of active components.
- In on–off keying, the transmitter is only turned on when transmitting a one, resulting in a 50% energy savings if the long-term probability of sending a one and a zero is the same.

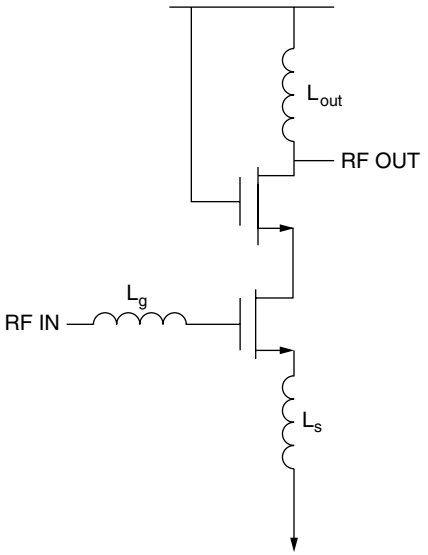


FIGURE 14.10 Inductively degenerated common source amplifier.

14.5 Low-Power Circuit Design Techniques

So far, we have introduced new technologies and promising transmit/receive (Tx/Rx) architectures, which may facilitate the implementation of ultra-low power, high-integration transceivers for sensor networks. Enabling the potential of these approaches requires the availability of the appropriate RF modules, which is the topic of this section. First, we explore the design of low-current front-end amplifiers. Then, an envelope detector implementation is explored, as it is another important component in the receiver. Next, we discuss the modules that compose the transmitter: the RF oscillator and the PA. Finally, we analyze the requirements of low-current proportional to absolute temperature (PTAT) bias circuits.

14.5.1 Low-Current RF Amplification

Whether used to overcome the noise of a subsequent stage, to provide isolation to the antenna, or to provide an input match, an input amplifier is a crucial part of virtually every RF receiver. The considerations for an input amplifier to be used in a sensor-network receiver are quite different from a traditional CMOS LNA. In traditional LNA design, the main goal is to minimize the noise figure while satisfying gain, linearity, and input matching constraints. As became clear in the receiver architecture section, the most important metric for the first stage in the receiver of an ultra-low power, gain-limited receiver is to obtain a large gain at a low current level. In such architectures, the noise figure of the front-end stage is typically swamped by the high noise figure of the detection circuitry. Thus, the goal is to maximize gain for a given power budget, while still maintaining reasonable noise figure, linearity and input matching performances.

An input impedance match is one of the primary architectural considerations for the input amplifier. Many circuit topologies can be used to achieve an input impedance match, including a simple resistive termination circuit. In terms of gain and noise performance, however, one promising topology is the inductively degenerated common source (IDCS) amplifier in Figure 14.10. The advantage of this topology is that it is able to provide high gain and a noiseless real impedance at the resonant frequency.

To explore the difficulty of obtaining sufficient gain at low bias currents in an IDCS amplifier, it is helpful to write down the equation for its overall transconductance G_m:

$$G_m \approx \left(\frac{f_t}{f_o}\right)\frac{1}{2R_S}$$

where R_s is the source impedance, f_t is the transistor cutoff frequency, and f_o is the operating frequency. It can be observed that G_m is not dependent on the g_m of the input device. Instead, it depends on the ratio of the operating frequency to the cutoff frequency and the value of the source resistance. Because the operating frequency and source resistance are set by system constraints, they are not available to the LNA designer as design variables. Consequently, to increase G_m, it is necessary to increase the f_t of the device.

Although it appears attractive that improved process technology (increased f_t) will result in higher gain, system-level integration considerations complicate the design. The problem results from the fact that increases in f_t will result in large values of L_g and very small values of L_s, both of which can pose problems. The small value of L_s is problematic because it is sensitive to parasitic inductances, reducing the accuracy of the input match impedance. The large value of L_g, where the size is determined by the operating frequency, makes it difficult to implement on-chip in modern CMOS processes while still maintaining a high-quality factor. Using an off-chip inductor also introduces a parasitic capacitance at the input, which alters the resonant frequency and decreases the G_m. When biasing low-current RF amplifiers to operate with a maximum f_t, it is essential to include the effect of this capacitance since it may be on the same order of magnitude as C_{gs}. Increasing the bias current allows for larger device widths for the same f_t, which makes the design less sensitive to parasitics and facilitate integration.

To achieve higher gains for a given bias current, it is necessary to deviate from traditional architectures. One example of a current-reuse (stacked) topology is depected in Figure 14.11. The idea is to recycle the

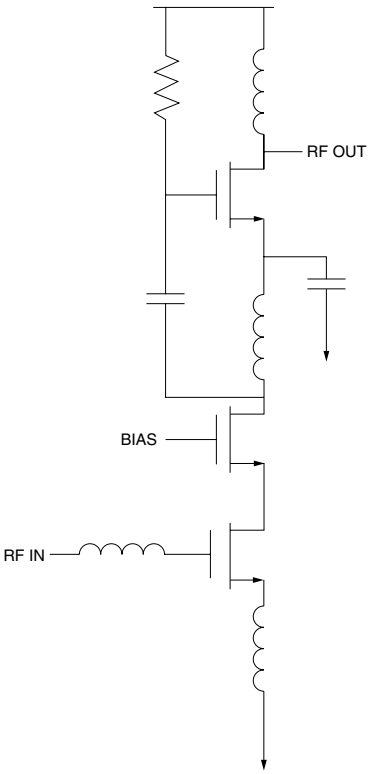


FIGURE 14.11 Current-reuse topology.

bias current so that it can be used by more than one stage. The circuit combines the inductive source degeneration architecture analyzed previously with a common-source amplification stage [17,18]. The gain is increased because the signal is coupled through a capacitor from the drain of the first stage to the gate of the second, providing two stages of amplification. The capacitor at the source of the second stage is made large so that it serves as an AC ground. A potential problem with this topology is the limited headroom and output swing because two stages are stacked on top of each other. In addition, the parasitic bottom plate capacitance of the coupling capacitor could reduce the achievable gain [19]. This topology also consumes more area due to the addition of large passives. Nevertheless, this topology is better suited for this design, and simulation has demonstrated that it is possible to achieve approximately 40 dB of voltage gain at <1mW of power consumption.

An important function of the receiver input stage is to provide an input impedance match. The input impedance of the IDCS amplifier is:

$$Z_{in}(\omega) = j\omega\left(L_s + L_g\right) + \frac{1}{j\omega C_{gs}} + \frac{g_m L_s}{C_{gs}}$$

In this equation, the nonquasi static gate resistance is ignored [20]. This assumption is acceptable in conventional LNA design, where the g_m of the device is high enough such that $r_{g,NQS}$ is approximately a few ohms, and input matching is not a problem. In low-power designs, however, g_m values are small, and the $r_{g,NQS}$ cannot be ignored making input matching a lot more complicated. The nonquasi static gate resistance term adds directly to the preceding input impedance equation. Input matching is further complicated by the parasitic capacitance of the bonding pad at the input, which alters the frequency at which an input match is achieved, as well as the value of the real impedance. Due to the small device sizes used in low-power design, this capacitance cannot be ignored; thus, accurate modeling of this parasitic is a necessity.

The noise figure of the IDCS amplifier is dominated by the drain noise and the induced gate noise of the input transistor. Because the drain noise is proportional to g_m, and the induced date noise is inversely proportional to g_m, an input transistor size would minimize the noise figure. In modern CMOS processes, the f_t is high enough that acceptable noise figures for low-power sensor networks can be obtained.

14.5.2 Envelope Detector

As discussed in Section 14.3, an envelope detector is a crucial component in a TRF architecture, and is useful to detect the oscillation envelope in amplitude control loops and/or a super-regenerative detector. The following CMOS envelope detector (see Figure 14.12) was found to be quite effective for its use in these applications.

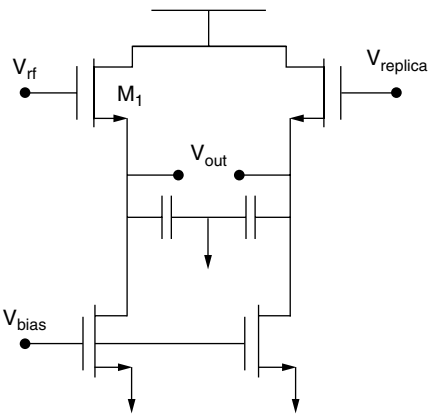


FIGURE 14.12 Simplified envelope detector schematic.

This circuit, which was previously used in bipolar applications, is well suited to CMOS implementations as well [21]. As depicted in the schematic, the single-ended circuit includes a replica half to produce a reference output voltage. The output is pseudo-differential, and the transconductance of M1 and the capacitive loading of the output determine the time constant. Translation of the RF-to-DC spectrum is accomplished though the nonlinearity of M_1. The fundamental and higher harmonics produced by this nonlinearity are filtered, leaving only the DC term. Because the conversion gain of the RF-to-DC spectrum is determined by the nonlinearity of the active devices, the CMOS transistors should be operated in the subthreshold regime, resulting in very low bias currents (<1 μA). Additionally, the efficiency of the circuit can be increased if transistor M_1 and its replica are designed in a DTMOS configuration (gate connected to body contact). This decreases the subthreshold slope to a value approaching 60 mV/decade, increasing the conversion gain of the detector [22].

14.5.3 RF Oscillator

As mentioned in Section 14.2, recent advancements in RF MEMS offer the potential increase the performance and decrease the power consumption of circuit blocks. The RF frequency synthesizer is an example of a transceiver block that can be greatly influenced by the availability of these new technologies. For example, the use of high-Q RF MEMS resonators can potentially create a very low-power, low-phase-noise VCO for use in a traditional frequency synthesizer. Additionally, although current RF MEMS resonators have poor fabrication tolerance compared with crystal resonators, they can be used to create a RF frequency reference in open-loop mode, without the need for a frequency synthesizer. Although the frequency stability would be worse than a traditional synthesizer, the power consumption and phase noise potentials are very promising. The following is a design example of a 1.9-GHz RF reference using the MEMS/CMOS codesign philosophies discussed in Section 14.2 [10].

The goals of the oscillator design were the following:

- Demonstrate the benefits of RF MEMS/CMOS codesign.
- Provide an ultra-low power, open-loop RF frequency reference without the need for a frequency synthesizer.
- Implement the solution in a very small, reproducible form factor.

The MEMS components used in the design were Agilent thin film bulk acoustic wave (FBAR) resonators [23]. Though typically used in the design of complex filter ladder networks for RF duplexers and transmit filters, they were found to be very well suited for use in RF oscillators. Because it is possible to customize the dimensions and properties of the MEMS resonator, accurate models are necessary to jointly optimize the CMOS and MEMS components. (See Figure 14.2 for a simplified model of the RF MEMS resonator.) A Pierce oscillator topology was used to provide low-phase-noise and low power consumption. Figure 14.13 is a simplified schematic of the oscillator core.

A complete model was developed, including the CMOS transistor models and the MEMS resonator models. The design was optimized for minimum power consumption and a 100-mV output voltage swing. A custom RF MEMS resonator chip and a 0.18-μm CMOS chip were fabricated. The system integration was accomplished using chip-on-board (CoB) technology. See Figure 14.14 for the completed oscillator subsystem.

As discussed in the preceding photograph, the system is easily bonded together with standard CoB technology, resulting in a very small form factor. The oscillator consumes 300 μA from a 1-V supply, and the 100-mV output swing exhibits a phase noise of −120 dBc/Hz at a 100-kHz offset. The phase noise and frequency stability is much better than an integrated LC oscillator, making the oscillator suitable as an RF frequency reference in low-power, low-datarate transceivers. In addition, the start-up time of this frequency reference is approximately 1 μs, providing a very fast turn-on time for the transceiver.

14.5.4 Nonlinear Power Amplifiers

One of the key factors determining the efficiency of the transmitter during transmission is the efficiency of the power amplifier (PA). Power amplifiers can be classified as linear PAs (Class A or AB) or nonlinear

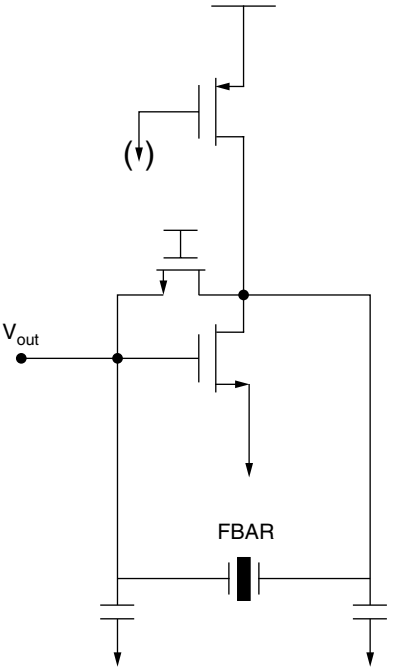


FIGURE 14.13 Simplified oscillator schematic.

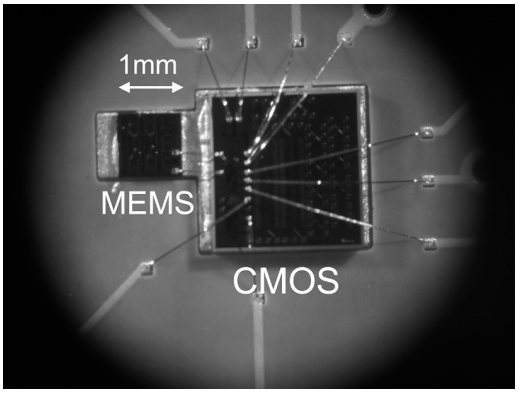


FIGURE 14.14 RF MEMS/CMOS chip-on-board implementation.

PAs (Class C, D, E, or F) [24]. Fully integrated CMOS linear PAs generally suffer from lower efficiency (30 to 40%) as compared with their nonlinear counterparts (40 to 50%) [25]. Thus, linear PAs are generally not suitable for wireless sensor networks. This is evident by considering, as an example, the goal of achieving a global 36% transmitter efficiency (see Section 14.4). If all the transmitter circuits excluding the PA consume 20% of the transmitter power, the efficiency of the PA must be 45%. This exceeds the typical achievable efficiency of fully integrated CMOS linear PAs and is only achievable by nonlinear PAs.

One possible implementation is the Class C PA depicted in Figure 14.15, in which the transistor operates as a current source modulated by the input signal. The high-Q tank filters out the harmonics and ensures that the transmitted signal is sinusoidal. High efficiency is obtained by reducing the conduction angle ($< 180°$) to minimize the product of the drain current and drain voltage; however, decreasing the conduction angle also reduces the output power for a given input drive amplitude.

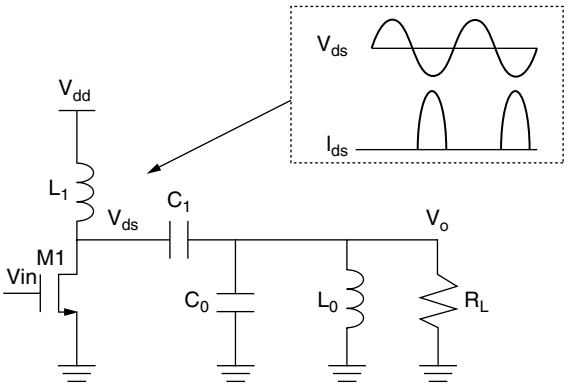


FIGURE 14.15 Class C amplifier. Inset depicts the drain voltage and drain current waveforms.

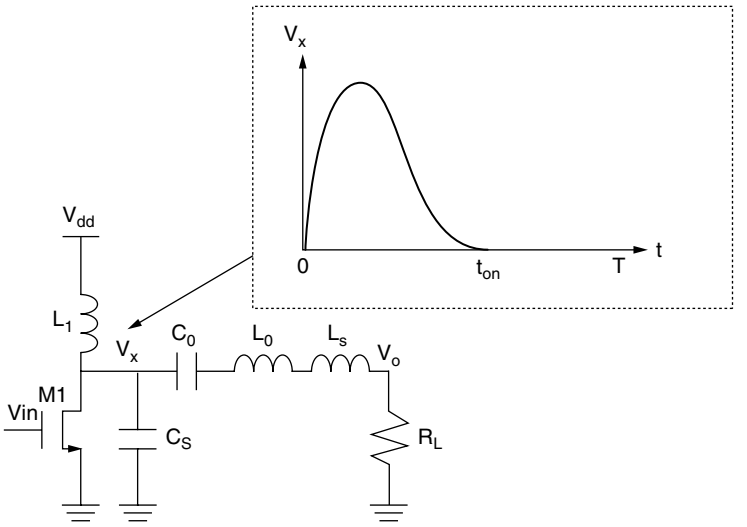


FIGURE 14.16 An example of a Class E power amplifier. Inset depicts the drain voltage waveform.

Using a switching PA, in which the transistor operates as a switch, can alleviate the strong dependence of efficiency on output power. Ideally, the voltage across the switch is zero when the switch is on and the current is zero when the switch is off. This results in zero power dissipation in the switch and thus higher efficiency. An example is the Class E PA depicted in Figure 14.16. The values of the components L_1, L_s, and C_s are chosen such that

$$v_x(t_{on}) = 0 \quad \text{and} \quad \left.\frac{dv_x}{dt}\right|_{t=t_{on}} = 0$$

where t_{on} represents the time at which the switch closes, and V_x is the voltage across the switch. C_0 and L_0 operate as a high Q filter to ensure that the output is sinusoidal.

In addition to the choice of the amplifier topology, some other factors can help to improve the power efficiency of the power amplifier and the transmitter as a whole:

1. Oscillator/PA Driver/PA codesign. The transistor in a nonlinear PA is usually much larger than the one in its linear counterparts. This means that the driver power is significant, especially if the

overall transmitted power is low. For example, if the oscillator output voltage is 160 mV zero to peak and a drive voltage of 800 mV is desired, the driver stage has to provide a gain of 5. For a g_m/I_d of 20 V^{-1} and an output resistance of 800 Ω, the driver stage will need 313 μW of power with a 1-V supply. To reduce this overhead, it is necessary to design the oscillator, the PA and the PA driver together for overall optimal efficiency. By increasing the drive power of the oscillator, for instance, it is possible to eliminate the driver stage altogether, resulting in lower overall transmitter power consumption.

2. Use of RF MEMS devices. The lack of on-chip high Q passive inductors is detrimental to the efficiency of the transmitter. To reduce losses, high-Q MEMS devices can be employed, for instance, to replace the LC tank in the Class C PA.
3. Power control. The transmitted power of the power amplifier can be reduced when transmitting to nearer sensor nodes. This not only preserves the scavenged energy but also reduces interference with other sensor nodes. The transmitter has to be designed to operate at optimal efficiency at various levels of radiated power.
4. Minimize the overhead turn-on time. The turn-on time of the entire transmitter is limited by the turn-on time of the high-Q local oscillator. During the turn-on time, no data can be transmitted and all energy expended is considered as overhead. To reduce this overhead, the PA can be switched on after a delay to minimize its idle time.

With the incorporation of these low-power design techniques, a nonlinear power amplifier with an efficiency of 50% is achievable. With a 20% overhead power for the oscillator and driver, the entire transmitter can operate with an on-state efficiency of 42%, meeting all the requirements discussed earlier.

14.5.5 On-Chip References and Bias Circuits

This section describes bias generation considerations for low-power transceivers for sensor node applications. This application presents interesting issues, partly due to the sub-threshold operation of most of the transistors present in the transceiver, which is primarily dictated by g_m/I_d efficiency considerations. First, it should be noted that, due to indoor operation and extremely low-power consumption (which eliminates the possibility of circuitry self-heating), the dynamic temperature behavior of the system does not impose aggressive specifications on the design. This allows the circuit design to be carried out at a nominal temperature of approximately 27°C, and successive translation of performance metrics at this temperature into lower bounds for performance metrics over the whole temperature range. The high sensitivity of the aforementioned transceiver architectures to both gain and oscillator startup time result in the need to stabilize device transconductance (g_m) over temperature.

In this scenario, a PTAT current source can be proven to be the optimal solution both for stability and from a power consumption perspective, as it draws from the supply only the amount of current needed to ensure $g_m(t) = g_{m,initial}$ over the entire temperature range. With a nonadaptive biasing technique, the required constant bias current would be determined by the g_m necessary at the worst-case temperature, thus increasing the average power consumption.

From an implementation perspective, a PTAT current source leaves little freedom to the designer: a current mirror, a low thermal coefficient resistor, and a translinear loop, usually implemented with subthreshold devices, are needed. Cascode implementation of the PMOS current mirror (see Figure 14.17) enhances the power supply rejection ratio and suppresses channel length modulation effects, but may be prohibitive if very low supply voltages are used.

Current levels in the mirror should be made as low as possible in order to minimize bias chain current consumption overhead; however, matching considerations limit the practical efficiency for a cascade mirror. For example, assuming an output current of 1 mA and an allowed 3-σ variation of 5% for g_m, matching analysis demonstrates that a mirror current (I_o) of 100 μA is reasonable (~85% efficiency), and requires an area of 1000 μm^2 for the bias circuitry. Mirror offset is temperature-dependent, and thus does not simply result in a constant offset for performance metrics, but instead in a distortion of

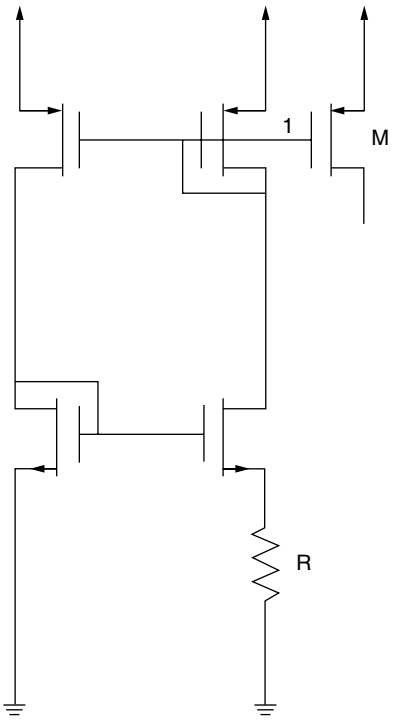


FIGURE 14.17 PTAT current reference schematic.

performance metrics over temperature. If a mirror current (I_o) of 10 μA was used to increase the efficiency (~98% efficiency), and the same matching criteria were used, the required die area would increase to 10,000 μm^2, which is not acceptable. An implementation of the mirror based on a simple PMOS pair (see Figure 14.17) would achieve poorer power supply rejection ratio (PSRR) and output impedance, but better performance with respect to matching for given area budget thanks to strong inversion mirror operation. Thus, multiple trade-offs exist in the design of ultra-low power biasing circuits, including area budget, parameter stabilization performance, supply insensitivity, and power efficiency.

14.6 System Integration

The packaging and physical integration strategy of the transceiver is crucial as it dramatically affects the performance, cost, and form-factor of the completed system. The number and size of the components and the assembly technology determine the size of the node. As mentioned previously, careful architectural decisions can help to minimize the part-count. Some components that deserve special attention are the antenna and the energy-supply chain.

As previously stated, using a relatively high carrier frequency can minimize the antenna size. Three main parameters that determine the type of antenna suitable for wireless sensor network are the radiation pattern, size, and bandwidth. As the deployment pattern of the sensor nodes is random, an omni-directional radiation pattern is desired. The form factor of the antenna should be small and allow for easy integration onto a PCB. The bandwidth of the antenna should be large enough to accommodate all channels. Given those considerations, commercially available chip-antennas (using ceramic dielectrics) are good candidates. These antennas have radiation patterns similar to a dipole and measure only about 30 mm by 5 mm. This allows for easy integration with the radio on a printed circuit board (PCB).

The energy-scavenging and the energy-storage (battery or capacitance) devices are the most volume-consuming components of the completed node. As listed in Table 14.1, their respective volumes are

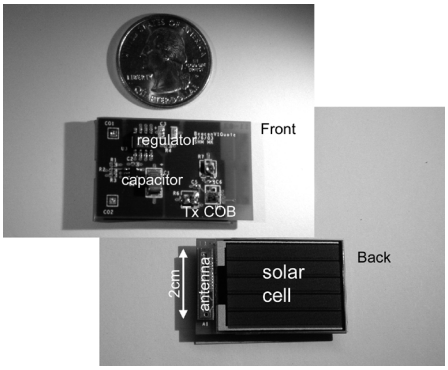


FIGURE 14.18 Photograph of the completed transmit beacon.

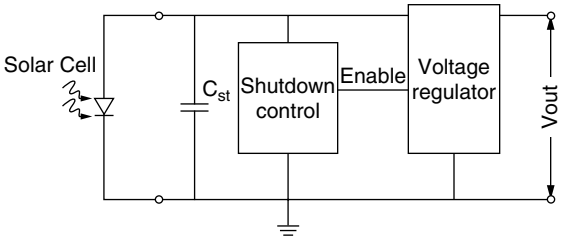


FIGURE 14.19 Transmit beacon powertrain.

directly proportional to the average and peak power dissipation levels of the node. Thus, decreasing the power dissipation of the transceiver — the most power-hungry component of the sensor node ultimately results in a linear decrease in the node size.

These guidelines are best illustrated with the aid of an implementation example, which also serves to demonstrate how the techniques introduced in this chapter effectively lead to ultra-low power and small size. Figure 14.18 is a photograph of a completed 1.9-GHz transmit beacon [26].

This system contains an ultra-low power 1.9-GHz transmitter, a 1.9-GHz chip antenna, a solar panel, an energy storage capacitor, and voltage regulation circuitry. Due to the high carrier frequency used, the chip antenna is very small and takes up little board space. It is mounted on the back of the board. Also on the back of the board is a small solar cell, which is able to provide all the energy necessary for transmitter operation. Figure 14.19 is a conceptual diagram of the transmit beacon power train.

The power train consists of a solar cell, an energy storage capacitor, shutdown control logic, and a linear voltage regulator, which supplies the RF circuitry with a stable 1.2-V supply. The transmitter is fully integrated, requiring only one MEMS resonator. It delivers 0 dBm (1 mW) to the 50-Ω chip antenna. The transmitter is mounted using a chip-on-board technology, and consumes minimal board space. See Figure 14.20 for a photograph of the transmitter circuitry.

As pictured in Figure 14.20, the transmitter requires no crystals or external inductors, and takes up approximately 2 mm × 3 mm of board space. The transmitter can operate from an approximately 1% duty cycle in low light conditions to 100% duty cycle in sunlight. Figure 14.21 depicts pertinent waveforms during the operation of the transmit beacon.

The top waveform is the voltage on the storage capacitor. Once the voltage regulator is enabled (bottom waveform), the RF transmitter turns on (middle waveform). During transmission, the energy in the storage capacitor is dissipated, and the voltage on the storage capacitor decreases. Once the energy on the storage capacitor is depleted, the transmitter is disabled and the charge process starts again. The board will operate indefinitely with no batteries or power supplies.

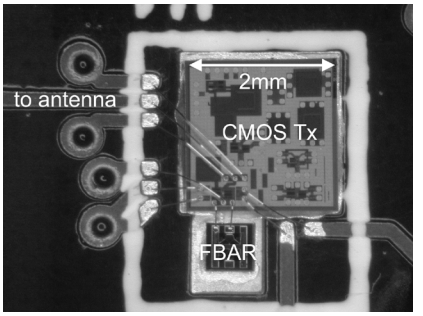


FIGURE 14.20 Transmitter CMOS/MEMS circuitry and antenna feed.

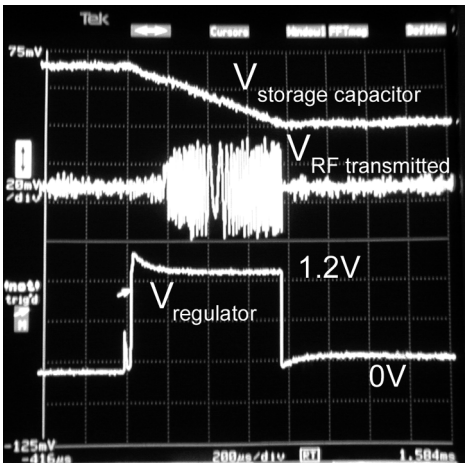


FIGURE 14.21 Transmit beacon ouput waveforms.

14.7 Conclusion

Functional and cost-effective ad hoc sensor networks will only reach ubiquity once the problem of ultra-low-power RF communication is solved. This chapter has discussed the design considerations for achieving ultra-low power and ultra-high integration in transceivers for wireless sensor networks. The concept of CMOS/RF MEMS codesign was introduced as a powerful tool in the successful implementation of these systems. Various transmitter and receiver architectures were analyzed for their applicability to this problem. Low-power design techniques were provided for the implementation of the essential RF modules. Finally, system-level considerations were discussed, including packaging and antennas.

It is the authors' conviction that the field of ultra-low power RF design is only in its infancy, and that exciting new approaches will continue to emerge in the coming years. With this chapter, we hope to have at least provided a glimpse into the myriad of potential solutions and to have projected some of the limiting bounds and constraints.

14.8 Acknowledgments

The authors thank Agilent Technologies for the resonator fabrication and ST Microelectronics for the CMOS fabrication. The generous support of DARPA (under the PACC and the IMT programs) is gratefully acknowledged.

References

[1] J. Rabaey et al., PicoRadio supports ad hoc ultra-low power wireless networking, *IEEE Comput.,* Vol. 33, No. 7, pp. 42–48, July 2000.

[2] F. Boekhorst, Ambient intelligence: the next paradigm for consumer electronics, *Proc. IEEE ISSCC 2002,* San Francisco, CA, February 2002.

[3] S. Roundy, Energy scavenging for wireless sensor nodes with a focus on vibration to electricity conversion, Ph.D. thesis, University of California–Berkeley, May 2003.

[4] L. Zhou, J.M. Kahn, and K.S.J. Pister, Corner-cube retroreflectors based on structure-assisted assembly for free-space optical communication, *IEEE J. Microelectromechanical Syst.,* Vol. 12, pp. 233–242, June 2003.

[5] E.A. Lin, A. Wolitz, and J. Rabaey, Power efficiency analysis of two rendezvous schemes for dense wireless sensor networks, *IEEE Int. Conf. on Commn.,* June 2004.

[6] R. Ruby, P. Bradley, J. Larson III, Y. Oshmyansky, and D. Figueredo, Ultra-miniature high-Q filters and duplexers using FBAR technology, *IEEE ISSCC Dig. Tech. Papers,* pp. 120–121, February 2001.

[7] B. Bircumshaw, G. Liu, H. Takeuchi, T.-J. King, R. Howe, O. O'Reilly, and A. Pisano, The radial bulk annular resonator: towards a 50-Ohm RF MEMS filter, *Tech. Dig., 12th Int. Conf. on Solid-State Sensors, Actuators, and Microsystems,* Boston, MA, pp. 875–878, June 8–12, 2003.

[8] K.M. Lakin, Thin-film resonators and high-frequency filters, http://www.tfrtech.com, retrieved June 2001.

[9] J. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan, PicoRadios for wireless sensor networks: the next challenge in ultra-low power design, *IEEE ISSCC Dig. Tech. Papers,* pp. 200–201, February 2002.

[10] B. Otis and J. Rabaey. A 300-μW 1.9-GHz CMOS oscillator utilizing micromachined resonators, *IEEE J. Solid-State Circuits,* Vol. 38, pp. 1271–1274, July 2003.

[11] A.-S. Porret, T. Melly, D. Python, C.C. Enz, and E.A. Vittoz, An ultralow-power UHF transceiver integrated in a standard digital CMOS process: architecture and receiver, *IEEE J. Solid-State Circuits,* Vol. 36, pp. 452–466, March 2001.

[12] S. Sheng and R. Brodersen, Low-power CMOS wireless communications, Kluwer, Dordrecht, 1998.

[13] J.S. Smith, High-density, low-parasitic direct integration by fluidic self-assembly (FSA), *Dig. IEEE Int. Electron. Devices Meeting,* pp. 201–204, Piscataway, NJ, 2000.

[14] C. van den Bos and C. Verhoeven, Architecture of a reconfigurable radio receiver front-end using overall feedback, *Proc. ProRISC,* The Netherlands, November 2001.

[15] J.R. Whitehead, *Super-Regenerative Receivers,* Cambridge University Press, U.K., 1950.

[16] B. Razavi, RF transmitter architectures and circuits, custom integrated circuits, 1999. *Proc. IEEE 1999,* May 16–19, 1999, pp. 197–204, San Diego, CA.

[17] A.R. Shahani, D.K. Shaeffer, and T.H. Lee, A 12-mW wide dynamic range CMOS front end for portable GPS receivers, *ISSCC Dig. Tech. Papers,* pp. 368–369, San Francisco, CA, February 1997.

[18] Triquint Semiconductor, TQ9203, low-current RF IC downconverter, Wireless Communication Products, 1995.

[19] B. Razavi, *RF Microelectronics,* Prentice Hall, Upper Saddle River, NJ, p. 180, 1998.

[20] Y. Tsividis, *The Operation and Modeling of the MOS Transistor,* McGraw-Hill, New York, 1999.

[21] R.G. Meyer, Low-power monolithic RF peak detector analysis, *IEEE J. Solid-State Circuits,* Vol. 30, No. 1, January 1995.

[22] F. Assaderaghi, D. Sinitsky, S. Parke, J. Bokor, P. Ko, and C. Hu, Dynamic threshold-voltage MOSFET (DTMOS) for ultra-low voltage VLSI, *IEEE Trans. on Elec. Devices,* Vol. 44, No. 3, March 1997.

[23] R. Ruby, Micromachined cellular filters, *IEEE MTT-S Int. Microwave Symp. Dig.,* pp. 1149–1152, June 1996.

[24] F.H. Raab, P. Asbeck, S. Cripps, P.B. Kenington, Z.B. Popovic, N. Pothecary, J.F. Sevic, and N.O. Sokal, Power amplifiers and transmitters for RF and microwave, *IEEE Trans. Microwave Theory and Techniques,* Vol. 50, No. 3, pp. 814–826, March 2002.

[25] R. Gupta and D.J. Allstot, Fully monolithic CMOS RF power amplifiers: recent advances, *IEEE Commn. Mag.,* Vol. 37, No. 4, pp. 94–98, April 1999.

[26] S. Roundy, B. Otis, Y.H. Chee, J. Rabaey, and P. Wright, A 1.9-GHz RF transmit beacon using environmentally scavenged energy, *Dig. IEEE Int. Symp. on Low-Power Elec. and Devices*, Seoul, Korea, 2003.

15

Power-Aware On-Demand Routing Protocols for Mobile Ad Hoc Networks

Morteza Maleki
Massoud Pedram
University of Southern California—Los Angeles

15.1 Introduction

Wireless mobile networks may be classified into two general categories:

1. Infrastructure-Based Networks. Wireless networks often extend, instead of replace, wired networks, and are referred to as infrastructure networks. A hierarchy of wide area and local area wired networks (WANs and LANs, respectively) is used as the backbone network. The wired backbone connects to special switching nodes called base stations. They are responsible for coordinating access to one or more transmission channel(s) for mobiles located within their coverage area. The end user nodes communicate via the base station using their respective wireless interfaces. Wireless LANs and WANs are a good example of this type.
2. Mobile Ad Hoc Networks (MANETs). A MANET is composed of a group of mobile wireless nodes that form a network independently of any centralized administration, while forwarding packets to each other in a multi-hop manner. Because the mobile devices are battery-powered, extending the network lifetime has become an important objective. Researchers and practitioners have focused on power-aware design of network protocols for the ad hoc networking environment.

Because each mobile node in a MANET performs the routing functions for establishing communication among different nodes, the "death" of even a few nodes, due to energy exhaustion, might cause the disruption of service in the entire network. The focus of this chapter is survey and design of power-aware unicast and multicast routing protocols and algorithms for wireless ad hoc networks with special attention to MANETs.

Metrics used by conventional routing protocols for the wired Internet, which is oblivious to an energy budget, typically do not need to consider any energy-related parameters. Thus, routing information protocol (RIP) [1] uses hop count as the sole route quality metric, thereby, selecting minimum-hop paths between the source and destinations. Open shortest path first (OSPF) [2], on the other hand, supports additional link metrics such as available bandwidth and link propagation delay. These algorithms, however, may result in a rapid depletion of the battery energy in the nodes along the most heavily used paths in the network. Routing protocols for wireless ad hoc environments contain special features to reduce the signaling overheads and convergence problems caused by node mobility and potential link failures. While these protocols do not necessarily compute the absolute minimum-cost path, they aim at selecting paths that have lower cost (in terms of metrics, such as hop count or delay). Such protocols must be modified to yield energy-efficient routing solutions.

A large number of researchers have addressed the problem of energy-efficient data transfer in the context of multi-hop wireless networks. Existing protocols may be classified into two distinct categories. One category of protocols is based on minimum-power routing algorithms, which focus on minimizing the power requirements over end-to-end paths. A typical protocol in this category selects a routing path from a source to some destination to minimize the total energy consumption for transmitting a fixed number of packets over that path. Each link cost is set to the energy required for transmitting one packet of data across that link and Dijkstra's shortest path algorithm is used to find the path with the minimum total energy consumption. These protocols traditionally ignore the power dissipated on the receiver side in a node, and therefore, tend to result in routing paths with a large number of short hops. A key disadvantage of these protocols is that they repeatedly select the least-power cost routes between source-destination pairs. As a result, nodes along these least-power cost routes tend to "die" soon by rapidly exhausting their battery energy. This is doubly harmful because the nodes that die early are precisely the ones that are most needed to maintain the network connectivity (and thus increase the useful service life of the network.)

A second category of protocols is based on routing algorithms that attempt to increase the network lifetime by attempting to distribute the forwarding load over multiple different paths. This distribution is performed by either intelligently reducing the set of nodes needed to perform the forwarding duties, thereby, allowing a subset of nodes to sleep over different periods of time, or by using heuristics that consider the residual battery power at different nodes and route around nodes that have a low level of remaining battery energy. In this way, they balance the traffic load inside the MANET to increase the battery lifetime of the nodes and the overall useful life of an ad hoc network. These protocols indeed constitute state-of-the-art power-aware network routing protocols and are the focus of this chapter.

This chapter is organized as follows. Section 15.2 gives a brief classification of the broad domain of ad hoc routing protocols. Section 15.3 gives a brief literature review of research in power-aware ad hoc routing protocols. Section 15.4 describes the rationale and details of the power-aware source routing (PSR) algorithm, and likewise, Section 15.5 describes the rationale and details of the proposed lifetime prediction routing (LPR) algorithm. Section 15.6 contains the experimental results comparing PSR and LPR with other popular ad hoc routing techniques.

15.2 MANET Routing Protocols

Routing protocols in ad hoc networks may be classified into three groups:

1. Proactive (table-driven)
2. Reactive (on-demand)
3. Hybrid

15.2.1 Proactive (Table-Driven) Routing Protocols

These routing protocols are similar to and come as a natural extension of those for the wired networks. In proactive routing, each node has one or more tables that contain the latest information of the routes to any node in the network. Each row has the next hop for reaching to a node/subnet and the cost of this route. Various table-driven protocols differ in the way the information about change in topology is propagated through all nodes in the network.

The two kinds of table updating in proactive protocols are the periodic update and the triggered update [3]. In periodic update, each node periodically broadcasts its table in the network. Each node just arriving in the network receives that table. In triggered update, as soon as a node detects a change in its neighborhood, it broadcasts entries in its routing table that have changed as a result. Examples of this class of ad hoc routing protocols are the destination-sequenced distance-vector (DSDV) [4] and the wireless routing protocol (WRP) [5]. Proactive routing tends to waste bandwidth and power in the network because of the need to broadcast the routing tables/updates. Furthermore, as the number of nodes in the MANET increases, the size of the table will increase; this can become a problem in and of itself.

DSDV, which is known not to be suitable for large dense networks, was described in Perkins [3]. A route table at each node enumerates all available destinations and the corresponding hop-count from the node. Each route table entry is tagged with a sequence number, which is created by a destination node. To maintain consistency of the route tables in a dynamically changing network topology, each node transmits table updates either periodically (periodic update) or when new, significant information is available (triggered update). Routing information is advertised by broadcasting or multicasting. The packets are transmitted periodically and incrementally as topological changes are detected. Topological changes include movement of a node from place to place or the disappearance of the node from the network. Information about the time interval between arrival of the very first routing solution and the arrival of the best routing solution for each particular destination is also maintained. Based on this information, a decision may be made to delay advertising routes that are about to change, thus, reducing fluctuations in the route tables. The advertisement of possible unstable routes is delayed to reduce the number of rebroadcasts of possible route entries that normally arrive with the same sequence number.

15.2.2 Reactive (On-Demand) Protocols

Reactive routing protocols take a lazy approach to routing. They do not maintain or constantly update their route tables with the latest route topology. Instead, when a source node wants to transmit a message, it floods a query into the network to discover the route to the destination. This discovery packet is called the route request (*RREQ*) packet and the mechanism is called route discovery. The destination replies with a route reply (*RREP*) packet. As a result, the source dynamically finds the route to the destination. The discovered route is maintained until the destination node becomes inaccessible or until the route is no longer desired.

The protocols in this class differ in handling cache routes and in the way route discoveries and route replies are handled. Reactive protocols are generally considered efficient when the route discovery is employed rather infrequently in comparison to the data transfer. Although the network topology changes dynamically, the network traffic caused by the route discovery step is low compared to the total communication bandwidth. Examples of reactive routing protocols are the dynamic source routing (DSR) [3,6], the ad hoc on-demand distance vector routing (AODV) [7] and the temporally ordered routing algorithm (TORA) [35]. The proposed power-aware routing algorithms belong to this category of routing algorithms. Because our approach is an enhancement over DSR, a brief description of DSR is warranted.

DSR, which is one of the widely accepted reactive routing protocols, is entirely on demand with no periodic activity of any kind at any level within the network. This pure on-demand behavior allows the number of routing discovery packets for a set of communication patterns to scale to zero when all nodes are approximately stationary. This is because if nodes are not moving about, all the routes employed by the current set of communication patterns will be discovered and will remain unchanged until the communications are completed. As nodes begin to travel or as communication patterns change, the

routing packet overhead of the DSR automatically scales only to that which is needed to track the routes currently in use.

In DSR when a node wishes to establish a route, it issues a RREQ to all of its neighbors. Each neighbor broadcasts this RREQ, adding its own address in the header of the packet. When the RREQ is received by the destination or by a node with a route to the destination, a RREP is generated and sent back to the sender along with the addresses accumulated in the RREQ header. The responsibility to assess the status of a route falls to each node in the route. Each node must ensure that packets successfully cross the link to the next node. If the start node does not receive an acknowledgement from the end node of a link on the path, it reports the error back to the source node and leaves it to the source to find and establish a new route. Because this process may consume a lot of bandwidth, DSR provides each node with a route cache to be used aggressively to reduce the number of control messages that must be sent. If a node has a cache entry for the destination when a route request for that destination is received at the node, it will use the cached copy instead of forwarding the request in the network. In addition, it promiscuously listens to other control messages (RREQs and RREPs) for additional routing data to add to its cache. DSR has the advantage in that no routing tables need to be maintained to route a given packet because the entire route is contained in the packet header; however, tables are used to cache routes and enhance performance. The caching of any initiated or overheard routing data can significantly reduce the number of control messages being sent, thus drastically reducing the overhead.

The disadvantages of DSR are twofold. DSR is not scalable to large networks. The *Internet Draft* acknowledges that the protocol assumes the diameter of the network is no greater than 10 hops. Additionally, DSR requires significantly more process resources than most other protocols. To obtain routing information, each node must spend much more time processing any control data it receives, even if that node is not the intended recipient. This is the ability of many network interfaces, to operate the network interface in "promiscuous" receive mode, including most current LAN hardware for broadcast media such as wireless. This mode causes the hardware to deliver every received packet to the network driver software without filtering, based on link-layer destination address. The promiscuous mode increases bandwidth utilization of DSR by reducing the number of control messages being sent out, though the use of promiscuous modes may increase the power consumption of the network interface hardware. Depending on the design of the receiver hardware, and in such cases, DSR can easily be used without the optimizations that depend on the promiscuous receive mode, or can be programmed to only, periodically switch the interface into promiscuous mode. Use of promiscuous receive modes is optional in DSR.

15.2.3 Hybrid Routing Protocols

Both the proactive and reactive protocols work well for networks with a small number of nodes. As the number of nodes increases, hybrid reactive/proactive protocols are used to achieve higher performance. Hybrid protocols attempt to assimilate the advantages of purely proactive and reactive protocols. The key idea is to use a reactive routing procedure at the global network level while employing a proactive routing procedure in a node's local neighborhood.

Zone routing protocol (ZRP) [3] is an example of the hybrid routing protocols. In ZRP, every node has a zone around itself, which includes nodes that are R hops away from that node. R is called the zone radius. ZRP limits the scope of proactive procedure to each node's zone. In this way, ZRP reduces the cost of frequent updates in response to continuously changing network topology by limiting the scope of the updates to the neighborhood of the change. The ZRP route discovery operates as follows. When a source node wants to find a route, it first checks whether the destination is within its zone. If so, the path to the destination is fetched from its table and no further route discovery is required. If the destination is not within the source routing zone, the source broadcasts a route request to its peripheral nodes, which are nodes in the border of the node's zone. The peripheral nodes execute the same algorithm — checking whether the destination is within their zone. If so, a route reply is sent back to the source indicating the route to the destination. If not, peripheral nodes forward the route request to their peripheral nodes, which execute the same procedure.

15.3 Low-Power Routing Protocols

The focus of research on routing protocols in MANETs has been the network performance. A handful of studies on power-aware routing protocols for MANETs have been conducted. Presented next is a review of some of them.

15.3.1 Minimum Power Routing

Singh et al. [8] proposed a routing algorithm based on minimizing the amount of power (or energy per bit) required to get a packet from source to destination. More precisely, the problem is stated as:

$$\min_{\pi}\left\{\sum_{(i,j)\in\pi} T_{ij}\right\} \tag{15.1}$$

where T_{ij} denotes the power expended for transmitting and receiving between two consecutive nodes i and j (aka cost of link (i,j)) in route π.

This link cost can be defined for two cases:

1. When the transmit power is fixed.
2. When the transmit power is varied dynamically as a function of the distance between the transmitter and intended receiver. Each node chooses the transmission power level for a link so that the signal reaches the receiver node with the same constant received power. To achieve this, clearly, links with larger distances require a higher transmission power than links with smaller distances.

For the first case, all the nodes in the network use a fixed power for all transmissions, which is independent of the link distance. Because the power cost of transmitting and receiving is fixed, then the link cost is fixed and consequently Equation (15.1) results in selecting a path with a minimum number of hops. In fact, assuming lossless links, a path with the minimum number of hops has a minimum number of transmissions and when the transmit power is fixed, then that path will also result in the least total power dissipation [9].

Generally, for a network with 802.11b as media access control (MAC) layer, energy consumption of each operation (i.e., receive, unicast transmit, broadcast, and discard) on a packet is given by Freeney and Nilsson [10]:

$$E(\,packet\,) = b \times packet\_size + c \tag{15.2}$$

where b and c are the appropriate coefficients for each operation. Coefficient b denotes the packet size-dependent energy consumption that depends on distance, wireless channel conditions and so on, whereas c is a fixed cost that accounts for acquiring the channel and for MAC layer control negotiation.

The link cost is the sum of all the costs incurred by the source and destination nodes. Traffic is classified as broadcast and unicast (i.e., point-to-point).

For unicast traffic, when receivers are in nonpromiscuous mode operation, the energy cost of the link between sender and receiver may be calculated as follows:

$$T_{SD} = E_{S\_send}(unicast\_packet) + E_{D\_recv}(unicast\_packet) \tag{15.3}$$

where S and D denote the sender and destination of the unicast packet.

In 802.11b, before sending a unicast packet, the source broadcasts a request-to-send (RTS) control message, specifying a destination and data packet size (duration of transmission). The destination responds with a clear-to-send (CTS) message. If the source does not receive the CTS, it may retransmit the RTS message. Upon receiving the CTS, the source sends the DATA and awaits an acknowledge (ACK)

from the receiver. For unicast traffic with nonpromiscuous mode operations, the energy cost for all nondestination nodes that can hear the packets is nearly zero because nondestination nodes only consume energy to receive the RTS packet. After this step, they will be discarding packets or even turning off their receivers during the ongoing transaction.

For unicast traffic when receivers are in promiscuous mode operation, the link cost between the sender and destination pair may be calculated as follows:

$$T_{SD} = E_{S\_send}(unicast\_packet) + \sum_{r \in Rs} E_{r\_recv}(unicast\_packet) \tag{15.4}$$

where R_s denotes the set of all nodes that can hear source S, which obviously includes destination D. Notice that $T_{S,D}$ represents an extended link cost. It accounts for the receiver energy cost of the neighboring nodes of the source that can hear the packets sent from the source to the intended destination. According to this link cost function, assuming that all candidate paths have same hop-count, the "best" paths are those that traverse sparse areas of the network where the node density is low.

For broadcast traffic, the sender listens briefly to the channel and sends data if the channel is free. If the channel is busy, the sender waits and retries later. The broadcast cost may be calculated as follows:

$$T_S = E_{S\_send}(broadcast\_packet) + \sum_{r \in R_S} E_{r\_recv}(broadcast\_packet) \tag{15.5}$$

This is not a link cost. Instead, it is a node cost which is assigned to sender(s) of broadcast packets. Broadcast and multicast routing algorithms may make use of this node cost to construct power-aware broadcast or multicast routing trees. These categories of routing algorithms will be explained later in this chapter.

The question of how to make use of the variable transmission power level is more involved. Stojmenovic and Lin [11] propose a local routing algorithm for this case. The authors assume that the power needed for transmission and reception is a linear function of d^{α} where d is the distance between the two neighboring nodes and α is a parameter that depends on the physical environment. The authors make use of the global positioning system (GPS) information to transmit packets with the minimum required transmit energy. The key requirement of this technique is that nodes in the MANET know the relative positions of themselves as well as all other nodes; however, this information may not be readily available. In addition, the GPS-based routing algorithm has two drawbacks. One is that the GPS cannot provide useful information about the physical environment (blockages and dynamics of wireless channels) to the nodes. The second weakness is that the power dissipation of the GPS is an additional power draw on the battery source of the mobile node.

Heinzelman et al. [12] proposed a minimum transmission energy (MTE) multi-hop routing algorithm for wireless sensor networks. Assuming a first-order radio model for a wireless sensor node and assuming d^n energy loss due to channel transmission where n is between 2 and 4, the article uses the following equations for calculating energy, sending and receiving k bit data over a distance d:

$$\begin{aligned} E_{Tx}(k,d) &= E_{tx\_elec} * k + E_{amp} * k * d^n \\ E_{Rx}(k) &= E_{rx\_elec} * k \end{aligned} \tag{15.6}$$

where E_{tx_elezc} and E_{rx_elezc} are energy dissipated in the transmitter and receiver electronics, and E_{amp} is energy dissipated in the transmit amplifier. If nodes A and B are separated by distance D (as depicted in Figure 15.1), then MTE calculates the optimum number of relaying nodes, K_{opt}, that is required to send data from A to B with minimum transmission energy as follows:

FIGURE 15.1 Relaying nodes are inserted between nodes A and B to reduce the energy of sending a packet from A to B.

$$K_{opt} = \left\lfloor \frac{D}{d_{char}} \right\rfloor \text{ or } \left\lceil \frac{D}{d_{char}} \right\rceil \tag{15.7}$$

where distance d_{char}, called the characteristic distance, is independent of D and is calculated as:

$$d_{char} = \sqrt[n]{\frac{(E_{tx\_elec} + E_{rx\_elec})}{(n-1)E_{amp}}} \tag{15.8}$$

15.3.2 Battery-Cost Lifetime-Aware Routing

The main disadvantage of the problem formulation of Equation (15.1) is that it always selects the least-power cost routes. As a result, nodes along these least-power cost routes tend to "die" soon by rapidly exhausting their battery energy. This is doubly harmful because the nodes that die early are precisely the ones that are most needed to maintain the network connectivity (and thus increase the useful service life of the network.) Therefore, it may be more advantageous to use a higher power cost route if this routing solution avoids using nodes that have low remaining battery energy. This observation has given rise to a number of "battery-cost lifetime-aware routing" algorithms as described next.

The min-sum battery cost routing algorithm [13] minimizes the total cost of the route. More precisely, this algorithm minimizes a summation of the inverse of remaining battery capacities for all nodes on the routing path. One drawback of this algorithm is that it may select a rather short path containing mostly nodes with high remaining battery capacity but also a few nodes with low remaining battery capacity. The cost of such a routing solution may be lower than that of a path with a large number of nodes all having medium level of remaining battery capacity. The former routing solution, however, is generally less desirable from the network longevity point of view because such a path will become disconnected as soon as the very first node on that path dies.

The min-max battery cost routing algorithm is a modification of the minimum battery cost routing to address the previously mentioned weakness. This algorithm attempts to select a route such that has the cost of the most "expensive" link (i.e., one with the minimum remaining battery capacity) on that path is minimum. Thereby, this algorithm results in a more balanced use of the battery capacity of the nodes in the network. One drawback of this algorithm is that because there is no guarantee that paths with the minimum hop-count or with the minimum total power are selected, it can select paths that result in much higher power dissipation to send traffic from a source to destination nodes. This feature does actually lead in shorter network lifetime because in essence the average energy consumption per delivered packet of user data has been increased.

A conditional min-max battery cost routing algorithm was also proposed in Toh [13]. This algorithm, which is a hybrid of the min-sum and the min-max battery cost routing algorithms, chooses the route with minimal total transmission power if there exists at least one feasible routing solution where all nodes in that route have remaining battery capacities higher than some prespecified threshold value. If there is no such routing solution, however, then the min-max routing algorithm is employed to select a route.

Several experiments were reported in Toh [13] to evaluate the effect of different battery cost-aware routing algorithms on the network lifetime. According to the reported results, the min-sum battery cost routing exhibits superior results compared to the min-max battery cost routing in terms of the expiration

times of the nodes in the network. Conditional min-max routing demonstrated better or worse results compared with the first two algorithms, depending on how the threshold value was chosen.

Maximum residual packet capacity (MRPC) was proposed in Misra and Banerjee [14]. MRPC is conceptually similar to the conditional min-max battery cost routing; however, MRPC identifies the capacity of a node not only by the residual battery capacity, but also by the expected energy spent in reliably forwarding a packet over a specific link. In fact, the objective function of Equation (15.1) is for a path with lossless links, however, for lossy links, the number of retransmissions in each link increases in proportion to the packet error rate of that link. Misra and Banerjee [14] proposed to rewrite the objective function of Equation (15.1) for reliable minimum total transmission power routing on lossy links and ignoring power expended for receiving packets as follows:

$$\underset{\pi}{Min}\left\{\sum_{(i,j)\in\pi}\frac{\rho_{ij}}{1-e_{ij}}\right\} \tag{15.9}$$

where e_{ij} is the packet error rate of *link*(*i,j*) (assuming constant packet size) when the transmit power level of the link is ρ_{ij}. Notice that Equation (15.9) is for the case of hop-by-hop retransmission where sender of each individual link provides reliable forwarding to the next hop by using localized packet retransmissions. Hop-by-hop retransmission may be contrasted to end-to-end retransmission where individual links do not provide link-layer retransmissions, and error recovery is achieved only via retransmissions initiated by the source node. For end-to-end retransmission, Equation (15.9) is modified as follows [15]:

$$\underset{\pi}{Min}\left\{\left(\sum_{(i,j)\in\pi}\rho_{ij}\right)\cdot\prod_{(i,j)\in\pi}\frac{1}{1-e_{ij}}\right\} \tag{15.10}$$

Several experiments are reported in Misra and Banerjee [14] to compare the routing method with different battery cost routings and minimum total transmission power routings. According to these results, although the first node dies sooner in the minimum total transmission power routings compared to the battery-cost routing algorithms, the last node dies later in the first case compared to the second case. MRPC, similar to other battery-cost routing algorithms, increases the expiration time of the first node while the death rate of the nodes is as smooth as the minimum total transmission power routing. Performance of MRPC, however, like that of the conditional min-max battery cost routing, depends on a threshold value. This threshold value determines exactly when either the min-max battery cost routing or the reliable minimum total transmission power is applied for route selection.

Chang and Tassiulas [16] describe a multi-path battery-cost routing algorithm to balance the energy consumption of nodes in a static wireless ad hoc sensor network. The routing has been designed for a network of stationary nodes whose task is to detect events inside a monitoring region. Nodes that detect an event (so-called source nodes) send their measurement data to specific destination(s) (so-called gateway) by using multi-hop routing. The article proposes a maximal residual energy path (MREP) routing algorithm, which has a min-max or min-sum objective function for selecting paths where the cost function for each link is as follows:

$$C_{i,j} = (F_i - \lambda.\rho_{ij})^{-1} \tag{15.11}$$

where C_{ij} is the cost of link (*i,j*), F_i is the full-charge battery capacity of node *i*, ρ_{ij} is transmit energy for sending a bit from node *i* to node *j*, and λ is an augmentation step size. Chang and Tassiulas [16] also propose a flow reduction (FR) algorithm. First, FR finds all possible paths from each source to a single gateway node (single commodity flow) or to several gateway nodes (multi-commodity flow). We define

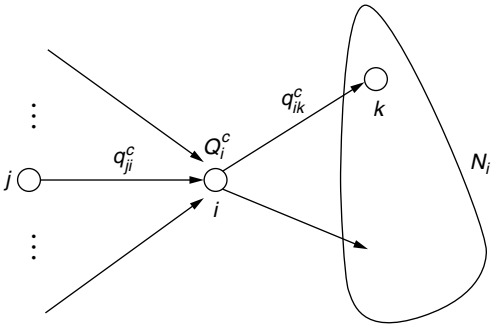


FIGURE 15.2 Node i sends out incoming traffic plus locally generated traffic toward nodes that can be reached from i (nodes in set N_i).

a commodity as data exchanged between a specific source-destination pair. FR defines longevity of a path π, L, as follows:

$$L_\pi(q) = \underset{i\in\pi}{Min}\ \tau_i(q)$$
$$\tau_i(q) = \frac{F_i}{\sum_{j\in N_i}\rho_{ij}\cdot\sum_{c\in C}q^c{}_{ij}} \tag{15.12}$$

where C is set of all commodities (i.e., data communications between various source-destination pairs), $q^c{}_{ij}$ is the rate at which data is sent from node i to node j in commodity c with transmit power $\rho_{i,j}$, N_i is the set of all nodes that can be reached by node i with power level $\rho_{i,j}$ (cf. Figure 15.2), $\tau_i(q)$ is the lifetime of node i, and $L(q)$ is the longevity of path under a given flow $q = \{q_{ij}\}$. The goal of FR is to divide the traffic flow of a source node to a sink (gateway) node on all paths between that pair of source and sink nodes such that all paths have the same lifetimes. FR tries to achieve this goal by redirecting some flow of each commodity from the shortest path (which has minimum longevity) toward longest path (which has maximum longevity) and this is repeated separately for each commodity in several steps until all paths between a source and sink have the same longevity. By defining network lifetime, as the time when the first node dies, Chang and Tassiulas [16] have demonstrated that the problem of finding maximum lifetime of a sensor network may be formulated as a linear programming problem as follows:

$$\begin{aligned}
&\textit{Maximize}\ \tau\\
&q^{(c)}_{ij} \geq 0, \forall i \in V, \forall j \in N_i, \forall c \in C\\
&\sum_{j\in N_i}\rho_{ij}\sum_{c\in C}q^{(c)}_{ij} \leq \frac{F_i}{\tau}, \forall i \in V\\
&\sum_{j:i\in N_j} q^{(c)}_{ji} + Q_i{}^{(c)} = \sum_{k\in N_i} q^{(c)}_{ik}, \forall i \in V - S^{(c)}, \forall c \in C
\end{aligned} \tag{15.13}$$

where $Q_i^{(c)}$ denotes the information generation rate at source nodes to be sent to destination nodes (or sink nodes) $S^{(c)}$ for each commodity c. V is set of all nodes, and N_i was defined previously. This linear programming can be solved in polynomial time. The solution to this linear programming problem provides the optimal network lifetime.

15.3.3 Energy-Conserving Techniques for Multi-Hop Ad Hoc Networks

It is known that an idle receiver listening for packets can consume almost as much as power as one doing active reception. More precisely, idle, receive, and transmit energy cost ratios for the transceiver part of a mobile node are 1, 2, 2.5 as per Kasten [17] and 1, 1.2, 1.7 as per Chen et al. [18]. Clearly, energy consumed in idle state of the transceiver cannot be ignored. In addition, Freeney and Nilsson [10] have demonstrated that a major source of extraneous energy consumption is from overhearing (or eavesdropping). Radios have a relatively large broadcast range. All nodes in that range must receive each packet to determine if it is to be received locally or forwarded to some other node in the network. Although most of these packets are immediately discarded, they cause superfluous energy consumption in the mobile node. Because the network interface may often be idle or simply overhearing data, the energy dissipated at these states can be saved by turning the radio off when it is not in use. In practice, however, this approach is not straightforward: a node must arrange to turn its radio on not only to receive packets addressed to it, but also to participate in any higher level routing and control protocols. The need for power-aware routing protocols is particularly acute for multi-hop ad hoc networks.

15.3.3.1 Power-Aware Multiple Access Protocol with Signaling (PAMAS)

PAMAS [19] is a MAC-level protocol that avoids overhearing problem by powering off radios in any of the following cases:

- A node powers off if it is overhearing a transmission and does not have a packet to send.
- If at least one neighbor is transmitting and at least one neighbor is receiving a transmission, a node may power off. This is because, even if the node has a packet to transmit, it cannot do so because of fear of interfering with its neighbor reception.
- If all neighbors of a node are transmitting and the node is not a receiver, it powers itself off.

In PAMAS, nodes attempt to capture the communication channel by exchanging RTS/CTS packets. These packets contain duration of data packet transmission. A node can learn about the times that it can be sleeping (or turn off its radio transceiver) by listening to the RTS/CTS exchange. In PAMAS, this exchange takes place over a separate signaling channel. Thus, this exchange does not interfere with ongoing data transmission. It is possible that a new transmission starts when a node is asleep. In such a case, the node does not know about the duration of data transmission. To solve this problem, nodes probe the signaling channel to find out the length of remaining transmission. Although PAMAS avoids the overhearing problem, it does not address the problem of energy consumption when nodes are idle. Solutions to this latter problem are proposed in GAF and span described next.

15.3.3.2 Geography-Informed Energy Conservation for Ad Hoc Routing

Geographical adaptive fidelity (GAF) [20] employs intelligent node scheduling techniques to conserve the energy. In MANETs, GAF is driven by this observation that when there is significant node redundancy in a MANET, multiple paths will exist between nodes, thus some intermediate nodes can be powered off to conserve energy while still maintaining the network connectivity.

GAF divides the whole area where the nodes are distributed to small virtual grid cells such that every node in each virtual grid cell can communicate with other nodes in that same cell. At any instant of time, exactly one node in each grid is active while all other nodes are in the power saving mode (sleep or discovery). As illustrated in Figure 15.3, nodes make transitions between discovery, sleep, and active states. In the discovery state, which is the initial state, a node identifies all other nodes that are located in the same grid cell by exchanging discovery messages. A node goes to the active state, T_d seconds after it enters the discovery state. A node stays in the active state for T_a seconds after which it goes to the discovery state. A node that is in the discovery or active states enters the sleep state when it finds out that some other node in the same grid is active and will thus handle routing. When transitioning to the sleep state, a node cancels all pending timers and powers down its radio. A node in the sleep state wakes up after an application-dependent sleep time T_s.

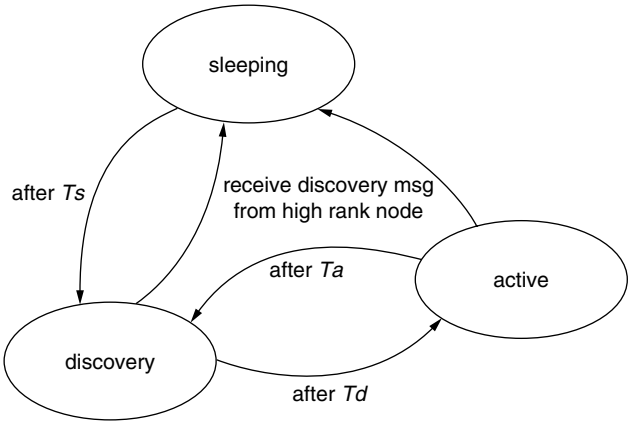


FIGURE 15.3 The state transition graph in GAF.

In GAF, nodes are ranked by several rules. An active node has a higher ranking than a node in the discovery state. For nodes that are in the same state, GAF gives higher ranking to nodes with a higher remaining battery capacity. Thus, a node with higher energy resource has a greater chance to become active. In this way, GAF achieves its load balancing strategy. When nodes have high mobility, it is possible that an active node moves out of its grid cell and leaves the grid cell with no active node in it. This problem significantly increases the packet drop rate. To avoid this problem, each node uses the GPS information to determine its bearing and velocity, thereby estimating the time that it expects to stay in the current grid, and adds this expected time to the discovery message. Any node that enters the sleep state wakes up after a time equal to the minimum of T_s and the expected time for the active node to stay in the grid cell.

Assuming a static network and without accounting for the protocol overhead, the maximum increase in the network lifetime that is achieved by GAF is equal to $(n \cdot R^2 / 5 \cdot A)$ where R is radio range of each node, and n is the total number of nodes distributed in an area A. In order for nodes in each grid cell to be able to communicate with nodes in the neighboring grid cells, the grid side length cannot be greater than $R/\sqrt{5}$. Thus, area A is divided to $A/(R/\sqrt{5})$ virtual grid cells.

15.3.3.3 Topology Maintenance for Energy Efficiency in Ad Hoc Networks (Span)

Span [18] builds on the observation that when there is a region of dense nodes, only a small number of these nodes need to be on at any given time to forward traffic. Span thereby adaptively elects some nodes as coordinators in the network. Coordinators stay awake to maintain connectivity of the network and to route packets in the network. All other nodes go to sleep to save power. These nodes periodically check if they should wake up and become a coordinator. One possible way for some node x to become a coordinator is that two neighbors of x cannot communicate with one another directly or through one or at most two coordinators. In addition, if node x has data to send out, it becomes a coordinator during its data transfer. When a node decides to be a coordinator, it uses a slotting and damping technique to delay its announcement of the fact. The node picks a random slot and delays its announcement until that slot. The random delay helps keep away from contention when several nodes decide to become coordinator at the same time. The delay function is as follows:

$$delay_i = \left((1 - \frac{R_i}{F_i}) + (1 - \frac{P_i}{\binom{N_i}{2}}) + \zeta \right) . N_i . \mu \tag{15.14}$$

where R_i is the remaining energy of the node, F_i is the full-charge battery capacity, P_i denotes the number of pairs of neighbors of i that cannot talk to one another unless through i itself, N_i denotes the number of neighbors of i (i.e., those nodes that can directly be reached from i). Recall that $\binom{N_i}{2}$ gives the number of pairs of neighbors of i. According to this equation, a node is more likely to pick an earlier time slot to become a coordinator if its ratio of remaining energy to full-charge battery capacity is high (close to 1). The node is also more likely to pick an earlier time slot if it can help connect a large number of pairs of its neighbors that would be disconnected without its assistance. μ is a random number that is picked uniformly in the (0,1) range, whereas ζ is the link propagation delay. A node switches from the coordinator to a noncoordinator role if every pair of its neighbors can reach each other directly or through one or two other coordinators. To balance the rate of energy consumption over all nodes, however, a node switches from a coordinator to a noncoordinator after some fixed period. In this way, it allows other nodes to become coordinators.

In span, nodes make all their decisions based on their local information and, unlike GAF, no knowledge of geographical information is required. Nodes find out about their neighbors proactively by broadcasting HELLO messages. These HELLO messages contain the status of the sender (i.e., coordinator or noncoordinator role), a list of its current coordinators, and its current neighbors. The list of the coordinators and neighbors are used by each of the node's neighbors in coordinator election and withdrawal rules that were described previously.

15.3.4 Energy-Aware Multicast Routing Algorithms

The primary goal of the conventional multicast routing protocols and algorithms has been to reduce the route latency because most multicast applications tend to be delay-sensitive audio/video broadcasting. Therefore, most of the multicast routing protocols are designed to construct a multicast tree that minimizes the communication latency. Because the number of hops is a good heuristic metric for capturing this latency, a multicast tree with the minimum number of hops has been favored by most routing protocols [21–23]. We call this tree the minimum hop-count tree (MHT). As has been described, in MANETs, two other criteria that make routing design an even more complex task (i.e., mobility and power efficiency) are used. The issue of mobility has been addressed extensively in the literature. In fact, the performance of multicast routing protocols has been evaluated in regard to their robustness to link failure due to the mobility [21,22,24,25]; however, little work has been accomplished on the development of a wireless multicast routing protocol in which power is key objective or constraint. More precisely, although some studies on the construction of energy-efficient broadcast and multicast tree in ad hoc networks [26,27] have been conducted, most of these works require a global view of the network and cannot be applied in a distributed way where the nodes have only local knowledge.

15.3.4.1 Minimum Energy Broadcasting

The objective of the minimum energy broadcasting is to reach from a specific source to all other nodes in the network by using multi-hop transmission while consuming the minimum total transmission energy and assuming that nodes have variable transmission power. In MANETs, broadcasting takes place by flooding the network from a specific source. Because the main use of flooding is in route discovery, it is important that flooding is done with the minimum total energy. Minimum energy broadcasting has been demonstrated as an NP-hard problem. Several heuristic algorithms for solving this problem have been proposed [26].

15.3.4.2 Energy-Aware Multicast Routing

The goal of energy-efficient multicast routing is to reach a subset of nodes (one-to-many cast) that we will refer to as multicast receivers, from a multicast source, such that we have maximum longevity of the paths between the source and the receivers. The problem of the energy-aware multicast tree is mathematically defined as follows. Consider a network graph $G(V,E)$, when V is set of nodes (or vertices) and

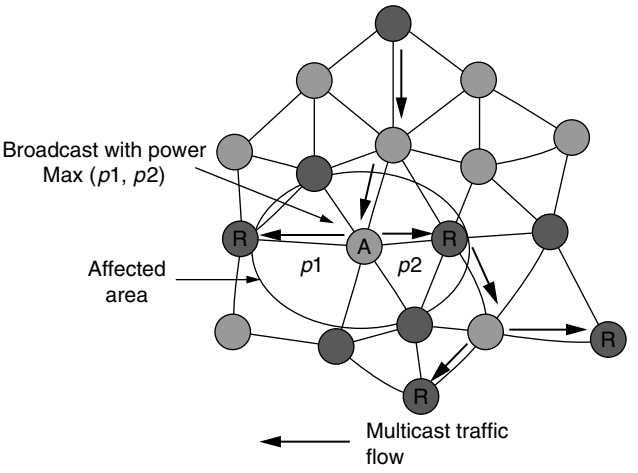


FIGURE 15.4 Neighbor cost effect in wireless networks.

E is set of edges in graph G. Let R_S denote the set of muticast receivers, s the multicast source, and $c(u,v)$ the cost of edge (u,v). The objective function may be stated as follows:

$$Min\ C(M) = \sum_{(u,v)\in M} c(u,v) \tag{15.15}$$

where $C(M)$ is the cost of multicast tree, M, connecting s to R_S.

The edge cost function $c(u,v)$ may represent the transmit power level needed for sending data from u to v. In this case, the previously mentioned objective function results in a minimum total transmit power multicast tree. In addition, $c(u,v)$ may be a battery-related cost of node u if the objective is to extend the lifetime of the network graph G. Figure 15.4 is an example of multicast tree. In general, finding a minimum energy multicast tree is equal to finding a minimum Steiner tree that is known to be an NP-hard problem [28]. Two related works on developing heuristic energy-aware multicast (or broadcast) trees are as follows:

- Least-Cost Shortest Path Tree (LPT). This is a tree obtained by superimposing all the least cost paths (or shortest paths) between the source and each multicast receiver.
- Broadcast Link-Based MST (BLMST). This is a minimum spanning tree where the link cost is set to the transmission energy needed to sustain communication over that link.
- Multicast Incremental Power Tree (MIPT). This tree is obtained from the Broadcast Incremental Power (BIP) tree proposed in Wieselthier et al. [27]. The BIP algorithm consists of the following steps:
  - For all nodes i in the tree and all nodes j not in the tree, evaluate $\rho'_{ij} = \rho_{ij} - \rho_i$, where ρ_{ij} was defined earlier, ρ_i denotes the power level of node i. (Note that ρ'_{ij} provides the incremental cost associated with adding node j to the tree.) Initially, the tree includes only the source node (i.e., the broadcast initiator node).
  - A pair (i,j) that results in the minimum value of ρ'_{ij} is chosen, and node j is added to the tree.
- This procedure is continued until all intended destination nodes are included. The MIPT is generated by pruning the broadcast tree (i.e., by eliminating all subpaths that are not required to reach the multicast receivers).

15.3.4.3 The Neighbor Cost Effect in Multicast Routing

Assume that a multicast tree from the source to several receivers has been constructed. The packet flow is coming out from the source, and is terminated at the leaves of the tree where the receivers are located. We will refer to those intermediate nodes of the tree that have more than one child in the tree as multi-

fanout nodes (e.g., node A in Figure 15.4). In MANETs because the MAC layer does not have the ability of multicasting [10], two distinct methods are used to send out the packets from a multi-fanout node:

- Multiple unicast. The parent node sends unicast packets to every child node in the multicast tree separately.
- Single broadcast. The parent broadcasts the packets to all nodes in its immediate neighborhood (which may include nodes that are not in the multicast tree).

Freeney and Nilsson [10] experimentally studied the power-optimal choice between these two methods. According to its results, the multiple unicast method results in much higher power consumption for the sender (parent node in the multicast tree). The following is empirical energy-cost measurement by Freeney and Nilsson [10] for broadcast and unicast send/receive packets:

	Unicast	Broadcast
Send (μW.sec/byte + μW.sec)	1.9 · packet_size + 454	1.9 · packet_size + 256
Receive (μW.sec/byte + μW.sec)	0.5 · packet_size + 356	0.5 · packet_size +56

These measurements have been completed on Lucent IEEE 802.11 2 MBPS WAVELAN PC card with 2.4-GHZ direct sequence spread spectrum (DSSS).

Based on these results, a single broadcast method in multi-fanout nodes is more energy efficient. When using the single broadcast method, however, all the nodes that are in the radio range of the sender listen to the channel and receive the packet, thereby, unnecessarily consuming power in receiving the packet. As a result, these nodes will find the multiple unicast method to be more beneficial to them from a power dissipation viewpoint. Consequently, one must consider the power consumption cost of all neighbors of nodes that broadcast packets when calculating the cost of a multicast tree, in which multi-fanout nodes use a single broadcast method. This phenomenon, which we will refer to as the neighbor cost effect, makes the problem of finding a multicast tree with optimal cost quite complex. Regarding neighbor cost effect, the general objective function of the multicast tree problem is changed as follows:

$$C(M,t) = \sum_{(u,v)\in M} c(u,v) + (\text{if } \deg(u) \geq 2 \quad \textit{then} \sum_{v\in N_u \wedge j\notin M} c(u,v) \text{ else } 0) \tag{15.16}$$

where $deg(u)$ denotes degree of node u in multicast tree M (including incoming and outgoing edges), and N_u refers to the set of nodes that are in the radio range of node u.

Another issue concerning the single broadcast method of multi-fanout nodes is that the farthest child from the parent determines the broadcast transmission power of that transmitting node. For example, in Figure 15.4, the transmission power of node A is $Max(\rho 1,\rho 2)$. Considering the neighbor cost effect in multi-fanout nodes makes the multicast routing problem even more challenging. Recall that finding a minimum energy-cost multicast tree without considering the neighbor cost effect is equivalent to that of finding a minimum Steiner tree, which is NP-hard. As a result, the problem of finding an energy-aware multicast tree with consideration of the neighbor cost effect is also an NP-hard problem.

Many algorithms for finding a tree with near optimal cost are available [29,30]. Although it is possible to modify some of these algorithms to account for the neighbor cost effect at multi-fanout nodes, this approach is ill advised in our context because these algorithms are too complex and require global information about the network connectivity graph to be applied. However, we are interested in finding solutions that can be deployed in an ad hoc network where nodes only have local knowledge about themselves and perhaps their neighboring nodes and must do the route discovery in a distributed, ad hoc manner (no global depository of information exists.) Furthermore, in ad hoc networks, the underlying network topology (connectivity graph) changes dynamically due to the mobility and link failure. Thus, ad hoc routing algorithms should be able to update their routes periodically. The routing update cost should be rather low.

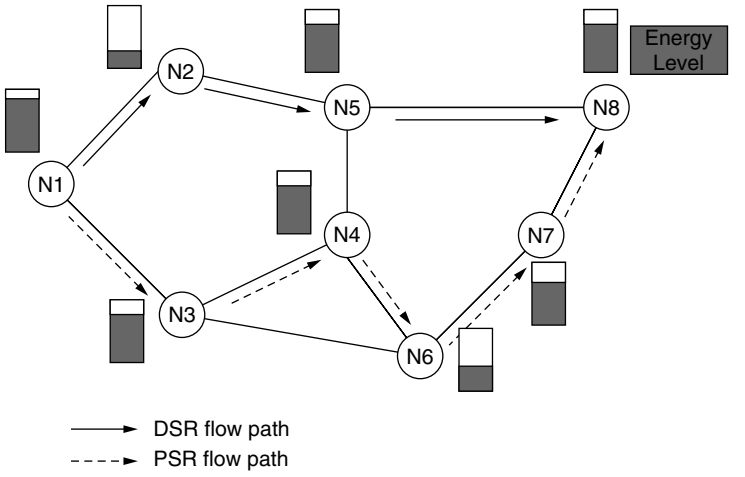


FIGURE 15.5 PSR avoids routes consisting of nodes with low remaining battery capacity.

15.4 Power-Aware Source Routing

15.4.1 Cost Function

The objective of power-aware source routing (PSR) [31] is to extend the useful service life of a MANET. This is highly desirable in the network because node death leads to a possibility of network disconnectedness, rendering other live nodes unreachable. PSR solves the problem of finding a route π at route discovery time t such that the following cost function is minimized:

$$C(\pi,t) = \sum_{i\in\pi} C_i(t)$$
$$C_i(t) = \rho_i \cdot \left(\frac{F_i}{R_i(t)}\right)^{\alpha} \tag{15.17}$$

where ρ_i is transmit power level of node i, F_i, and R_i are full-charge and remaining battery capacities of node i at time t, and α is a positive weighting factor.

PSR uses a graded cost function as explained next. The exponent α is a discrete function of the ratio of the remaining battery capacity over the full-charge battery capacity. As this ratio decreases and successively becomes less than a specified set of threshold values, α increases according to a fixed schedule. In this way, nodes with very low battery capacity contribute a much higher value to the total path cost. In other words, if a path from source to destination has some nodes with a very low residual battery, the cost of the path will be very high, and therefore, PSR will behave similar to the min-max battery cost routing. Figure 15.5 illustrates how PSR avoids routes that include node(s) with low remaining energy. Routing path N1-N2-N5-N8 has the minimum hop-count from N1 to N8, and, therefore, it is selected by DSR; however, this route includes node N2, which has a very low remaining energy capacity. Thus, PSR selects another route: N1-N3-N4-N7-N8.

In DSR, because the route selection is done based on a shortest path finding algorithm (i.e., it selects paths with the minimum number of hops), a selected path may become invalid only due to node movements. In contrast, in PSR, both the node mobility and the node energy depletion may cause a path to become invalid. Because the route discovery and route maintenance processes in PSR are slightly more complicated compared to their counterparts in DSR, these two steps will need to be described in

detail. In addition, because PSR is derived from DSR, the PSR description will often be contrasted with that of DSR.

15.4.2 Route Discovery

In DSR, activity begins with the source node flooding the network with RREQ packets when it has data to send. An intermediate node broadcasts the RREQ unless it gets a path to the destination from its cache or it has already broadcast the same RREQ packet. This fact is known from the sequence number of the RREQ and the sender ID. Consequently, intermediate nodes forward only the first received RREQ packet. The destination node only replies to the first arrived RREQ because that packet usually takes the shortest path.

In PSR, all nodes except the destination calculate their link cost (cf. Equation 15.17) and add it to the path cost in the header of the RREQ packet (cf. Equation 15.17). When an intermediate node receives a RREQ packet, it starts a timer (T_r) and keeps the cost in the header of that packet as mincost. If additional RREQs arrive with same destination and sequence number, the cost of the newly arrived RREQ packet is compared with the mincost. If the new packet has a lower cost, mincost is changed to this new value and the new RREQ packet is forwarded. Otherwise, the new RREQ packet is dropped. The destination waits for a threshold (T_r) number of seconds after the first RREQ packet arrives. In that time, the destination examines the cost of the route of every arrived RREQ packet. When the timer T_r expires, the destination node selects the route with minimum cost and replies. Subsequently, it will drop any received RREQ. The reply also contains the cost of the selected path appended to it. Every node that hears this route reply adds this route along with its cost to its route cache table. Although this scheme may somewhat increase the latency of the data transfer, it results in a significant improvement of network lifetime, as discussed later.

15.4.3 Route Maintenance

Route maintenance is needed for two reasons:

1. Mobility. Connections between some nodes on the path are lost due to their movement.
2. Energy Depletion. The energy resources of some nodes on the path may be depleting too quickly.

In the first case, a new RREQ is sent out and the entry in the route cache corresponding to the node that has moved out of range is purged. In the second case, two possible approaches are used:

1. Semi-Global Approach. The source node periodically polls the remaining energy levels of all nodes in the path and purges the corresponding entry in its route cache when the path cost increases by a fixed percentage. Notice that this results in very high overhead because it generates extra traffic.
2. Local Approach. Each intermediate node in the path monitors the decrease in its remaining energy level (thus the increase in its link cost) from the time of route discovery because of forwarding packets along this route. When this link cost increase goes beyond a threshold level, the node sends a route error back to the source as if the route was rendered invalid. This route error message forces the source to initiate route discovery again. This decision is only dependent on the remaining battery capacity of the current node, and thus, is a local decision.

PSR adopts the local approach that minimizes the control traffic. Furthermore, it assumes that all transmit power levels ($\rho_{i,j}$) are constant. This enables PSR to separate the effect of mobility from that of energy depletion during route maintenance. More precisely, for each node i along a path π, we define a "delta cost" function as follows:

$$\Delta C_i(t_a) = C_i(t_a) - C_i(t_d) = \rho_i \cdot \left(\frac{F_i}{R_i(t_a)}\right)^{\alpha} - \rho_i \cdot \left(\frac{F_i}{R_i(t_d)}\right)^{\alpha} \tag{15.18}$$

where t_a denotes the time instance when this route entry is fetched from the cache table of node I; t_d denotes the time instance at which π was added to the cache table of node i; and $C_i(t_a)$ is the fractional cost contributed by node i to total cost of the path π at time t_a, whereas $C_i(t_d)$ is the fractional cost contributed by node i to total cost of the path π at time t_d.

Assuming that α remains unchanged from time t_a to time t_d, then the condition for invalidating route π from the cache table of node i is:

$$\frac{\Delta C_i(t_a)}{C(\pi, t_d)} > \delta \tag{15.19}$$

where δ is a user-specified threshold value.

This condition invalidates a path π in the cache table of node i if the change in the normalized cost of node i exceeds a threshold δ. This metric appears to be a good way of capturing the dynamics of the node usage in MANETs. As the remaining energy of a node decreases, the cost of the node increases. The node will force new routing decisions in the network by invalidating its own cache entries to various destinations. If a path was recently added to the cache table, however, the node will not force a new decision (route finding step) unless the node's remaining energy is depleted by a certain normalized amount, due to messages passing through that path. The effect of δ on the performance of PSR is studied in detail in Section 15.6.

It should be noted that we provision for the reuse of invalidated paths if node i was the source of the message and wanted to continue to send data via this path as follows. When node i has data to send to the destination, it looks up its route cache and chooses a route, if such a route can be found in the cache, irrespective of whether the route was invalidated or not. In this way, we avoid redundant route discoveries in the presence of an existing route. The invalidated cache is purged after a fixed time. The invalid entries are analogous to the victim buffer in the cache structure of general-purpose processors; however, the same does not hold good for relaying data. If a cache entry is invalidated in a node and that node is asked to relay data/reply to the destination of that cache entry, then the node will send a route error back to the source. This reply will invalidate routing entries for all nodes on the trace path back to the source. The PSR function of intermediate nodes is given in Figure 15.6 in pseudo code. The function of the destination node is similar to the intermediate node with the exception that it does not need to check for validation of the path when it refers to its cache because it is the end point for each possible path between that itself and the source.

15.5 Lifetime Prediction Routing

15.5.1 Basic Mechanism

Lifetime prediction routing (LPR) [32] is an on-demand source routing protocol that uses battery lifetime prediction. The objective of this routing protocol is to extend the service life of MANET with dynamic topology. This protocol favors the path with the maximum remaining lifetime. We represent our objective function as follows:

$$\max_{\pi} \left\{ L_\pi(t) = \min_{i \in \pi} (\tau_i(t)) \right\} \tag{15.20}$$

where $L_\pi(t)$ is lifetime of path π and $\tau_i(t)$ is the predicted lifetime of node i at time t.

15.5.1.1 Lifetime Prediction

Each node tries to estimate its battery lifetime based on its past activity. This is achieved using a simple moving average (SMA) predictor by keeping track of the last N values of residual energy and the

```
Event: T_x expired
      If reply-route-from-cache
       Add the cost of the path in the cache to the mincost and append
       it to RREP packet and Send it back;
       else
            Drop any coming RREQ packet;
            mincost=0;
Event: RREQ  packet Arrival
If T_x expired
      Drop the packet;
      Exit;
If packet→ cost ≥ mincost
      Drop the packet;
Else
      mincost = packet→cost;
      Add the cost of this node to packet→cost;
If earliest RREQ arrival
        Start T_x timer;
Look up in the Cache;
If any path to the destination exists in the cache
         Check for validation of the fetched path from the cache;
          If path is valid
            Reply back after T_x timer expiration;
           Else Invalid that path and Propagate the RREQ packet;
   Else Propagate the RREQ packet;

Event: RREP or Data packet arrival
Check for the validity of the route used if exists is in the cache
If valid
      Forward it;
 Else Make that path (the cache line) invalid and Send error back
```

FIGURE 15.6 Pseudo code for the key operations performed in the intermediate nodes of a path in the PSR.

corresponding time instances for the last W packets received/relayed by each mobile node. This information is recorded and stored in each node. We have carefully compared the predicted lifetimes based on the SMA approach to the actual lifetimes for different values of W and found $W = 10$ to be a good value.

Our motivation in using lifetime prediction is that mobility introduces different dynamics into the network. In Chang and Tassiulas [16] the lifetime of a node is a function of residual energy in the node and energy to transmit a bit from the node to its neighbors (cf. Equation (15.12)). This metric works well for static networks for which it was proposed; however, it is very difficult to efficiently and reliably compute this metric when we have mobility because the location of the nodes and their neighbors constantly change.

PSR does not use prediction and only uses the remaining battery capacity. LPR is superior to PSR because LPR not only captures the remaining (residual) battery capacity but also accounts for the rate of energy discharge. This makes the cost function of LPR more accurate. This is true in MANETs because mobility can change the traffic patterns through the node, which thereby affects the rate of depletion of its battery. In addition, recent history is a good indicator of the traffic through the node, and thus we chose to employ lifetime prediction.

Our approach is a dynamic distributed load balancing approach that avoids power-congested nodes and chooses paths that are lightly loaded. This helps LPR achieve minimum variance in energy levels of different nodes in the network. As an example, consider the scenario in Figure 15.7. Here, node F has three flows going through it (D → F →, B → F →, and C → F →). Now, if A wants to transmit data to E, the shortest path routing will use A → F → E. LPR will use A → B → C → D → E, however, because E is very power-congested (as a result of relaying multiple flows) and the path passing through F will not be selected by LPR.

Figure 15.8 is an example that depicts how different policies of DSR, PSR, and LPR give different answers with the same scenario. Although PSR avoids choosing a path that goes through node N6, because of low remaining energy, the path selected by LPR (N1-N3-N6-N7-N8) includes N6. The reason is that N6 has a low depletion rate, and its estimated lifetime is high.

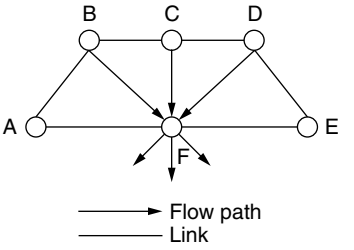


FIGURE 15.7 LPR avoids power-congested paths.

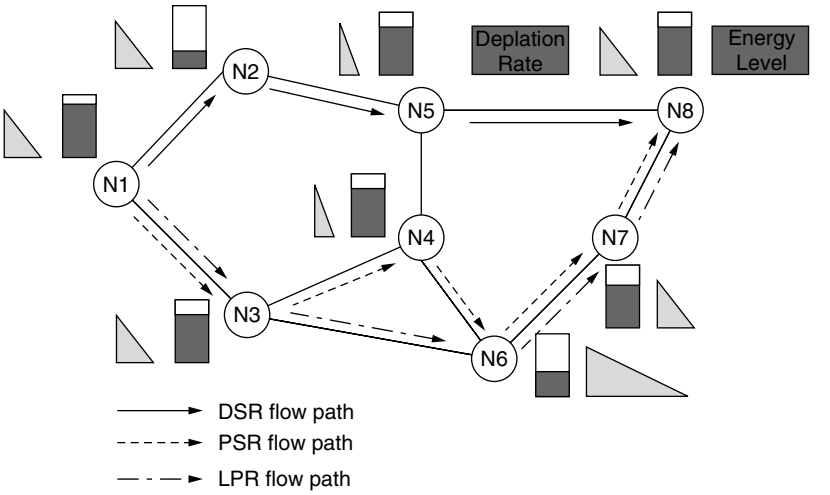


FIGURE 15.8 LPR avoids paths consisting of nodes with high-energy depletion rates.

15.5.2 Route Discovery

Route discovery in LPR is similar to PSR. In LPR, all nodes except the destination calculate their predicted lifetime, τ_i (see Equation (15.21)) and replace the minlifetime in the header with τ_i if τ_i is lower than the existing minlifetime value in the header.

$$\tau_i(t) = \frac{R_{r,i}(t)}{\frac{1}{W-1}\sum_{k=i-W+1}^{i} r_k(t)} \tag{15.21}$$

where $R_{r,i}(t)$ denotes remaining energy at the i_{th} packet is being sent or relayed through the current node, $r_k(t)$ is rate of energy depletion of the current node when the k_{th} packet was sent and is calculated by the ratio of the difference between residual energies of the nodes for packets $k-1$ and k and the difference between arrival times of these two packets, and W is length of the history used for calculating the SMA.

When an intermediate node receives a RREQ packet, it starts a timer (Tr) and keeps the min. lifetime in the header of that packet as minlifetime. If additional RREQs arrive with the same destination and sequence number, the cost of the newly arrived RREQ packet is compared with the mincost. If the new packet has a lower cost, mincost is changed to this new value, and the new RREQ packet is forwarded. Otherwise, the new RREQ packet is dropped (see Figure 15.9).

In LPR, the destination waits for a threshold number (Tr) of seconds after the first RREQ packet arrives. During that time, the destination examines the cost of the route of every RREQ packet that

```
Predict its lifetime;
If its lifetime < minlifetime
      Replace minlifetime with its lifetime;
If Sequence Number exists
      Compare minlifetime of current RREQ with minlifetime
      of existing one;
      If new minlifetime <= old minlifetime
            Discard new RREQ;
      If new minlifetime > old minlifetime
            Replace old minlifetime with new minlifetime;
            Forward new RREQ;
If Sequence Number does not exist
      Save this minlifetime;
      Forward RREQ;
```

FIGURE 15.9 Pseudo code of functions performed in an intermediate node as it is executing the LPR algorithm.

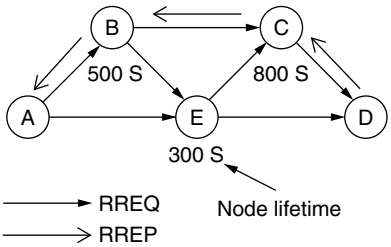


FIGURE 15.10 The route setup process in LPR.

arrived. When the timer (*Tr*) expires, the destination node selects the route with the minimum cost and replies. Subsequently, it will drop any received RREQs. The reply also contains the cost of the selected path appended to it. Every node that hears this route reply adds this route along with its cost to its route cache table. Although this scheme can somewhat increase the latency of the data transfer, it results in a significant power savings, as discussed later. A simple example of this process is illustrated in Figure 15.10. Here, the route A-B-C-D is chosen by LPR over the route A-E-D because the path lifetime of the former is in the 500s, which is greater than the latter.

LPR has a route invalidation timer that invalidates old routes. This helps in removing old routes. This also avoids over usage of particular routes in cases of low mobility.

15.5.3 Route Expiration

Route maintenance is needed for two reasons:

1. Connections between some nodes on the path are lost due to their movement
2. Change in the predicted lifetime

In the first case, a new RREQ is sent out and the entry in the route cache corresponding to the node that moved out of range is purged. The following policy is adopted to tackle the second situation.

Once the route is established, the weakest node in the path (the node with minimum predicted lifetime at path discover time) monitors the decrease in its battery lifetime. When this remaining lifetime decrease goes beyond a threshold level, the node sends a route error back to the destination as if the route was rendered invalid. The destination sends this route error message to the source. This route error message forces the source to initiate route discovery again. This decision is only dependent on the remaining battery capacity of the current node and its discharge rate in the short history, and thus is a local decision. LPR adopts this local approach because this approach minimizes control traffic. Figure 15.11 is an example of the route expiration process.

More precisely, node *i* generates a route error at time t when the following condition is met:

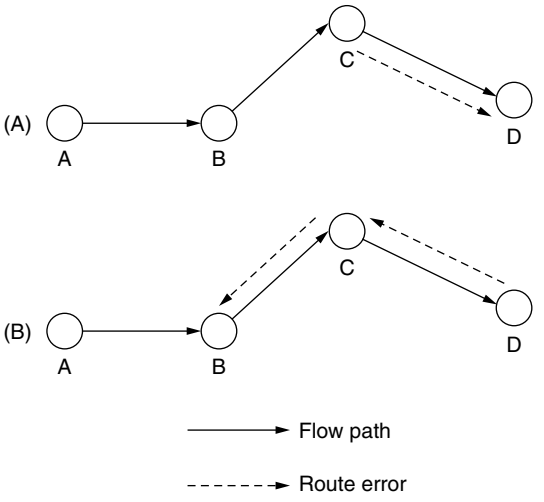


FIGURE 15.11 (a) Node C sends route error to destination node D; (b) Node D sends route error to source A to invalidate the whole path.

$$\tau_i(t_0) - \tau_i(t) \geq \delta \tag{15.22}$$

where t denotes the current time, t_0 is time of the route discovery, and δ is threshold value.

15.6 Quantitative Evaluation of Source Routing Algorithms

15.6.1 Simulation Setup

We used the event driven simulator ns-2 [33] along with the wireless extensions provided by CMU [34]. The simulation consists of a network of 20 nodes confined in a 1000×1000 m^2 area. Random connections were established using CBR traffic (at 4 packets/second) such that each node has chance to connect to every other node. Packet size was 512 bytes and each simulation was executed for 20,000 sec. The initial battery capacity of each node is 100 units. Nodes followed a random waypoint mobility model with a specific max velocity and no pause time. Each packet relayed or transmitted consumes a fixed amount of energy from the battery as given by Equation (15.2); a and b are constants.

The key parameters of study are the network lifetime, node lifetime, and root mean square (RMS) of energy consumption (E_{RMS}) in the network. We vary the speed and radio transmission range and study their effects on these metrics.

15.6.2 Simulation Results

The network lifetime is defined as the time taken for a fixed percentage of the nodes to die due to energy resource exhaustion. Network lifetime of DSR, PSR, and LPR are compared for a given scenario. Here, the speed of each node is 10 m/s and radio transmission range is 125 m. Figure 15.12 plots the time instances at which a certain number of nodes have died when simulating LPR, PSR, and DSR. Note that in Figure 15.12, node death of all 20 nodes is not plotted because some nodes are still alive at the end of the simulation. Some of these nodes, however, are rendered unreachable because many of the nodes have exhausted their energy and thus cannot reach other nodes consistently.

As can be seen, the first node in DSR and PSR dies about 20% earlier than in the case of LPR. Similarly, in DSR 5 nodes die approximately 32% earlier than LPR and 27% earlier than LPR in the case of PSR.

Due to the dynamic nature of the path cost function of PSR (and LPR), a discovered path cannot remain valid for a long time. This is because these connections, if maintained for a long period, may

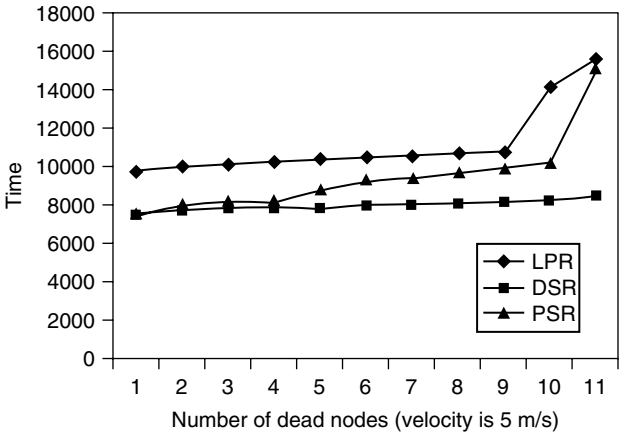


FIGURE 15.12 Number of dead nodes in DSR, PSR, and LPR as a function of the elapsed time.

exhaust the energy of some nodes on that path. Discovered paths are in the cache, however, and can be accessed whenever they are required in DSR, and (as implemented in ns-2) only mobility can invalidate these cache entries. In addition, cache invalidation is very expensive for the network because the route is reconstructed by flooding the network. This is handled in PSR as described in the next paragraph.

When the path is discovered, every node puts its remaining energy and path cost in the cache entry. Intermediate nodes check for validity of this path by computing the cost difference as in Equation (15.19). Here, δ (the threshold) is a metric that decides how often we invalidate the cache. This threshold affects the performance of PSR. If the threshold is very high, we do cache invalidation very rarely, and might end up overexercising some nodes in the path. If it is very low, the cache invalidation rate is very high and may lead to unnecessary flooding in the network. The effect of varying this threshold is plotted in Figure 15.13.

Because LPR outperforms PSR in terms of results in a longer network lifetime, we have selected LPR to compare it with DSR for the rest of simulation.

To increase the lifetime of the network, the variance of the residual energy of the nodes should be minimized. Figure 15.12 is not informative in this regard. A histogram of the snapshots of the energy

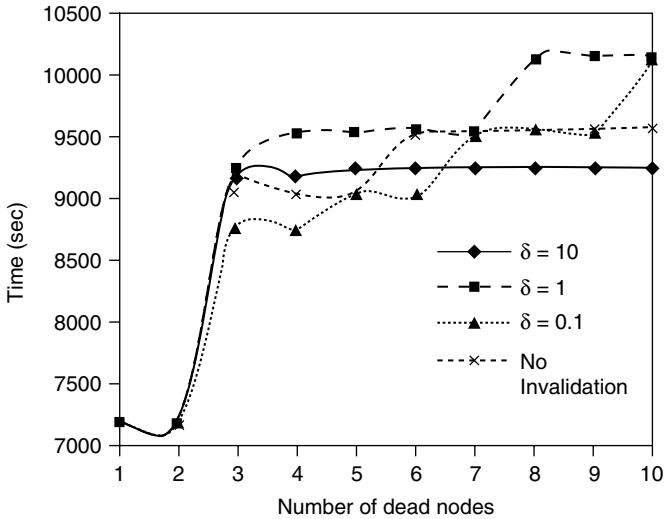


FIGURE 15.13 Effect of the threshold value, δ (for the PSR path invalidation step) on the network lifetime.

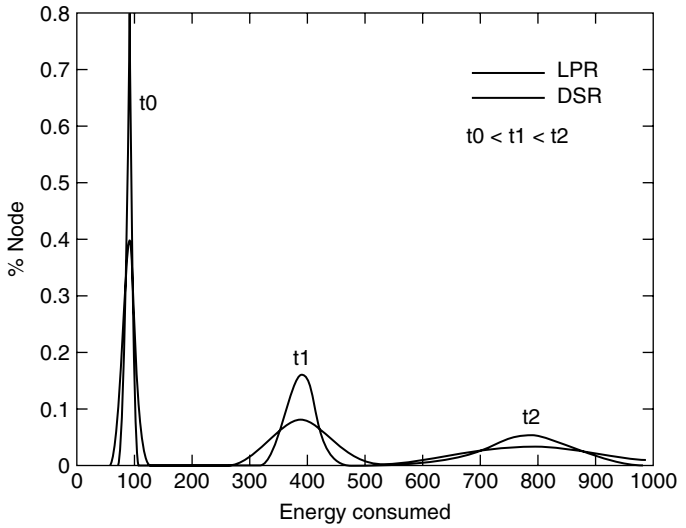


FIGURE 15.14 Distribution of energy consumed at three different time instances for LPR and DSR.

consumption in each of the nodes at different time instances would be more informative. Figure 15.14 depicts this histogram at three time instances. Initially, all nodes have zero energy consumption. As time increases, variance of energy consumption or remaining energy of nodes increases, but the rate of increasing for LPR is more than DSR. One of the ways to compare such histograms would be to look at the RMS of the remaining energy (E_{RMS}) at different time instances. It provides information about the total energy consumed and spread of consumed (residual) energy. Figure 15.15 plots the evaluation of E_{RMS} as a function of time for DSR and LPR before any node dies out. The effect of mobility on E_{RMS} can also be observed in this figure. A linear estimation of E_{RMS} is depicted for ease of comparison. As can be observed, LPR is always better than DSR in terms of E_{RMS} value. This graph is in agreement with our expectations. As the velocity of node movement increases, however, the rate of energy consumption in the network goes up. This is expected because higher velocity of movement implies more route discoveries being performed and consequently higher energy consumption in the network. In addition,

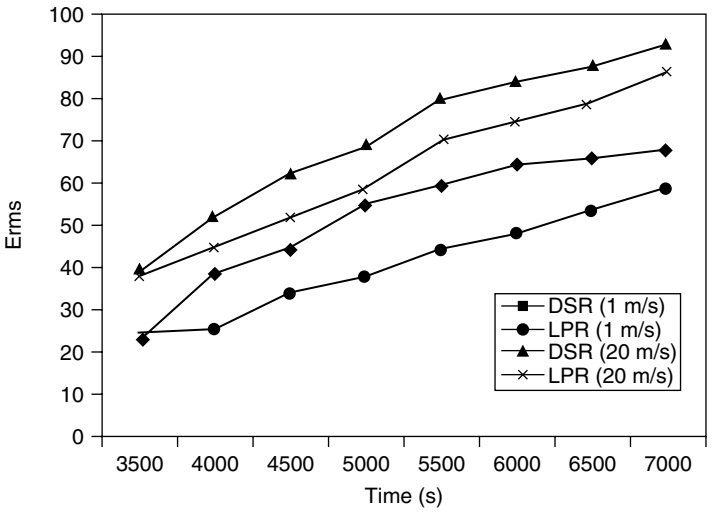


FIGURE 15.15 Evaluation of E_{RMS} for different velocities of node movement.

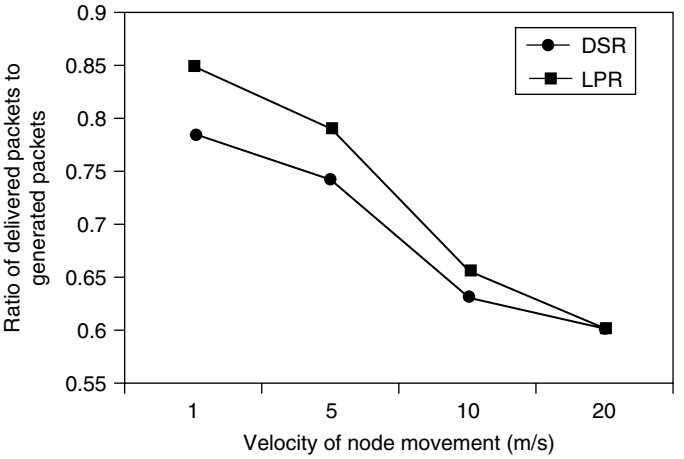


FIGURE 15.16 Packet delivery ratio vs. velocity of node movement.

as the node mobility increases, the difference between DSR and LPR decreases. This could be attributed to two reasons:

1. LPR makes use of the fact that DSR overloads certain nodes and has a big variance between remaining energies of the nodes. As mobility increases, the amount of overhead (control packets for route discovery) increases for both DSR and LPR. Consequently, less room is available for LPR to balance the energy consumption among the nodes in the network and extend its network lifetime.
2. Because more route discoveries occur, no paths are overused even by DSR. Consequently, DSR also achieves load balancing to an extent, decreasing the gain seen by LPR.

Packet delivery ratio is defined as the number of delivered data packets to the number of generated data packets in all nodes. Note that the number of generated packets is the "expected" number of generated packets. We generate as many as 200,000 data packets during the simulation. They are generated between random sources and destination pairs at random times. Many of these might not have reached their intended destination because of the lack of existence of a route between the source and destination for various reasons. In addition, the network lifetime clearly affects this ratio. If the network was alive for longer time, it implies that more data traffic goes through because we establish random connections throughout the time of simulation.

As plotted in Figure 15.16, for lower velocities of node movement, LPR has a greater ratio of delivered packets. As the mobility increases, however, this ratio goes down. The intuition for why LPR does not perform as well in higher velocities was presented earlier.

The transmission range is another parameter that can affect the performance of routing protocols because it changes the connectivity of the network. We changed the transmission range to see the effect of the degree of connectivity on our metric (see Figure 15.17). We assume the same transmission power for all nodes in a simulation. The node transmit range was assigned two different values (125 and 200 m) for the simulations. We make the following observations based on this figure:

- When the transmission range increases, each node covers more nodes. In other words, when a node sends a unicast or broadcast packet, more nodes will receive packets and they consume power in their receiver. Thus, each transmission has a lot of power overhead for the network. As a result, when the range increases, nodes discharge faster.
- The number of hops per route decreases by increasing the transmission range. Thus, nodes have less participation in relaying packets resulting in lower activity for each node and slower discharge of its battery capacity.

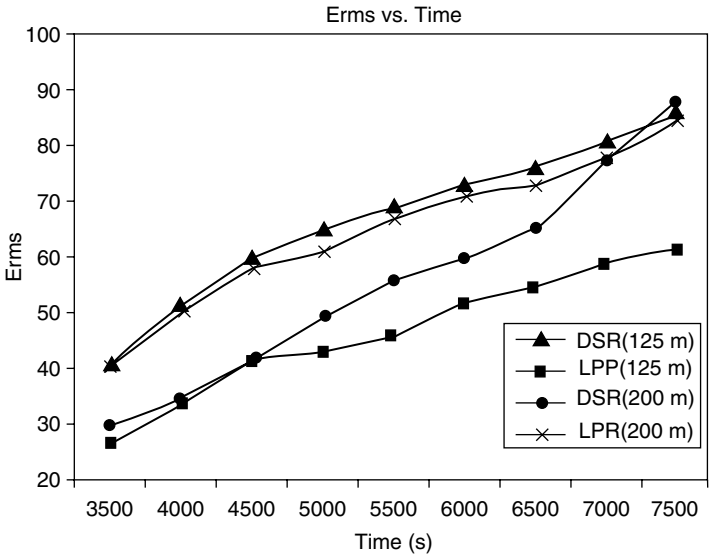


FIGURE 15.17 Effect of transmission range on the E_{RMS} (node velocity is 5 m/s).

When range increases from 125 to 200 m, the dominant effect is the first and the charge rate of the nodes increases drastically. Both of those effects reduce the effect of the LPR scheme and as can be seen, the difference between LPR and DSR decreases, such that when the range is 200 m, the difference is not clear. To reduce the cost of the power due to the second effect, one way is to shut down the nondestined nodes in the range of a transmitting node.

In LPR, route discovery process needs more control packets to be propagated in the network because it needs to compare all possible paths between a source and a sink and selects a path with maximum lifetime. To illustrate the overhead of LPR on the network, we have measured the ratio of the number of control packets to the number of delivered packets in the network. This normalizes the overhead of the routing protocol to the goodput (i.e., number of received packets) in the network. Figure 15.18 plots this ratio for LPR and DSR for different velocities of node movement and for 380 user datagram protocol (UDP) connections. As the velocity of movement increases, routes are valid for a shorter time and more route discoveries are done in the network resulting in more control packets and more difference between

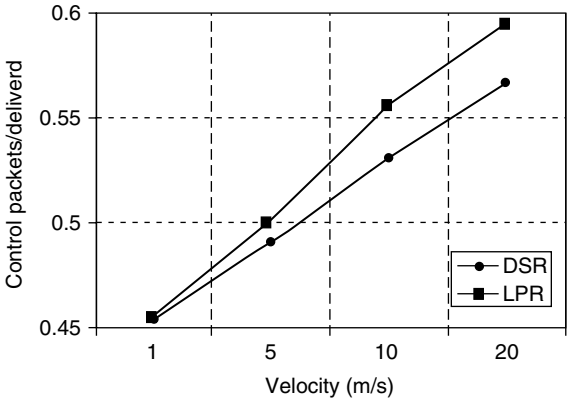


FIGURE 15.18 The ratio of control packets to delivered packets as a function of velocity of node movement for LPR and DSR for 380 UDP connections.

LPR and DSR. LPR increases the ratio of the control packet to transmit a packet less than 4%. The increase in the size of a control packet in DSR to that of the LPR is approximately 1/10, and the overhead in energy for sending such a packet increases by approximately 0.4%. Thus, the additional energy overhead of LPR for route discovery is small.

15.7 Conclusion

One of the main design constraints in MANETs is that they are energy constrained. Thus, network routing algorithms must be developed to consider energy consumption of the nodes in the network as a primary objective. In MANETs, every node has to perform the functions of a router. Therefore, if some nodes die early due to lack of energy so that the network becomes fragmented, then it may not be possible for other nodes in the network to communicate with each other. This chapter presented PSR and LPR protocols for MANETs where the aim is to maximize the network lifetime (which is typically defined as the duration time after which a fixed percentage of the nodes in the network "die" as a result of energy exhaustion). This goal of extending the network lifetime was accomplished by finding routing solutions that tend to minimize the variance of the remaining energies of the nodes in the network. Although these power-aware network routing protocols and algorithms tend to create additional control traffic, simulations reported in this chapter demonstrate that they improve the network lifetime by more than 20% on average.

References

[1] C. Hedrick, Routing information protocol, *RFC 1058,* http://www.faqs.org/rfcs/rfc1058.html, Aug. 2001.
[2] J. Moy, OSPF version 2.0, *RFC 2328,* http://www.faqs.org/rfcs/rfc2328.html, Aug. 2001.
[3] C.E. Perkins, *Ad Hoc Networking,* Addison-Wesley, Reading, MA, 2001.
[4] C. Perkins and P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, *Proc. of ACM SIGCOMM Conf. on Communications Architectures, Protocols, and Applications,* pp. 234–244, Oct. 1994.
[5] S. Murthy and J.J. Garcia-Luna-Aceves, An efficient routing protocol for wireless networks, *ACM Mobile Networks and Applications J., Special Issue on Routing in Mobile Communication Networks,* vol. 1, no. 2, pp. 183–197, 1996.
[6] D.B. Johnson, D.A. Maltz, Y.-C. Hu, and J.G. Jetcheva, The dynamic source routing for mobile ad hoc wireless networks, *IETF Internet Draft,* http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt, Nov. 2001.
[7] C.E. Perkins, E.M. Belding-Royer, and S. Das, Ad hoc on-demand distance vector (AODV) routing, *IETF Internet Draft,* draft-ietf-manet-aodv-12.txt, Nov. 2002.
[8] S. Singh, M. Woo, and C.S. Raghavendra, Power-aware routing in mobile ad hoc networks, *Proc. of Mobile Computing and Networking (Mobicom),* pp. 181–190, 1998.
[9] IEEE Standards Board 802 Part 11: *Wireless LAN Medium Access Control (MAC) and Physical Layer (Phy) Specifications,* Mar. 1999.
[10] L.M. Freeney and M. Nilsson, Investigating the energy consumption of a wireless network interface in an ad hoc networking environment, *Proc. IEEE Infocom,* pp. 1548–1557, Apr. 2001.
[11] Stojmenovic and X. Lin, Power-aware localized routing in wireless networks, *Proc. IEEE Trans. on Parallel and Disrtibuted Systems,* vol. 12, no. 11, pp. 1122–1133, May 2001.
[12] W. Rabiner, W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, *Proc. 33rd Annu. Hawaii Int. Conf. on System Sciences,* pp. 3005–3014, Jan. 2000.
[13] C.K. Toh, Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks, *IEEE Communication Mag.,* pp. 138–147, June 2001.

[14] A. Misra and S. Banerjee, MRPC: maximizing network lifetime for reliable routing in wireless environments, *Proc. IEEE Wireless Commn. and Networking Conf.,* pp. 800–806, Aug. 2002.
[15] S. Banerjee and A. Misra, Minimum energy paths for reliable communication in multi-hop wireless networks, *Proc. MobiHoc,* pp. 146–156, June 2002.
[16] J.-H. Chang and L. Tassiulas, Energy-conserving routing in wireless ad hoc networks, *Proc. Infocom,* pp. 22–31, Mar. 2001.
[17] O. Kasten, Energy consumption. ETH-Zurich, Swiss, Federal Institute of Technology. Available at http://www.inf.ethz.ch/~kasten/research/bathtub/energy_consumption.html, Apr. 2001.
[18] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks, *Proc. Mobile Computing and Networking (Mobicom),* pp. 85–96, July 2001.
[19] S. Singh and C. Raghavendra, PAMAS: power-aware multiple access protocol with signaling for ad hoc networks, *ACM Computer Communication Review,* 28(3):5–26, July 1998.
[20] Y. Xu, J. Heidemann, and D. Estrin, Geography-informed energy conservation for ad hoc routing, *Proc. Mobile Computing and Networking (Mobicom),* pp. 70–84, July 2001.
[21] S.-J. Lee, W. Su, and M. Gerla, On-demand multicast routing protocol (ODMRP) for ad hoc networks, *IETF Internet Draft,* http://www.cs.ucla.edu/NRL/wireless/PAPER/draft-ietf-manet-odmrp-02.txt, Oct. 2002.
[22] J.J. Aceves and E. Madruga, The core-assisted mesh protocol, *IEEE JSAC,* vol. 17, no. 8, pp. 1380–1394, Aug. 1999.
[23] E. Royer and C. Perkins, Multicast ad hoc on-demand distance vector (MAODV) routing, *IETF Internet Draft,* draft-ietf-manet-maodv-00.txt.
[24] S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, A performance comparison study of ad hoc wireless multicast protocols, *Proc. IEEE Infocom*, pp. 565–574, Mar. 2000.
[25] M. Gerla, C.-C. Chiang, and L. Zhang, Tree multicast strategies in mobile, multi-hop wireless networks, *ACM/Kluwer Mobile Networks and Applications,* vol. 4, no. 3, http://www.ietf.org/proceedings/00dec/I-D/draft-ietf-manet-maodv-00.txt, Oct. 2002.
[26] M. Cagalj, J. Phubaux, and C. Enz, Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues, *Proc. Mobile Computing and Networking (Mobicom),* Sep. 2002.
[27] J.E. Wieselthier, G.D. Nguyen, and A. Ephremides, On the construction of energy-efficient broadcast and multicast trees in wireless networks, *Proc. Infocom,* pp. 585–594, Mar. 2000.
[28] A. Goel and K. Munagala, Extending greedy multicast routing to delay sensitive applications, *J. Algorithmica,* vol. 33, no. 3, pp. 335–352, 2002.
[29] M. Parsa, Q. Zhu, and J.J. Garsia-Luna-Aceves, An iterative algorithm for delay-constrained minimum-cost multicasting, *IEEE/ACM Trans. on Networking,* vol. 6, no. 4, pp. 461–474, Aug. 1998.
[30] J. Cong, A.B. Kahng, G. Robins, M. Sarrafzadeh, and C.K. Wong, Provably good performance-driven global routing, *IEEE Trans. on Computer-Aided Design,* vol. 11, no. 6, pp. 739–752, 1992.
[31] M. Maleki, K. Dantu, and M. Pedram, Power-aware source routing in mobile ad hoc networks, *Proc. Int. Symp. on Low-Power Electronics and Design (ISLPED),* pp. 72–75, Aug. 2002.
[32] M. Maleki, K. Dantu, and M. Pedram, Lifetime prediction routing in mobile ad hoc networks, *Proc. IEEE Wireless Commn. and Networking Conf.,* Mar. 2003.
[33] NS -2 Manual, http://www.isi.edu/nsnam/ns/doc/index.html, Feb. 2002.
[34] CMU Monarch Extensions to ns, http://www.monarch.cs.rice.edu/, Feb. 2002.
[35] V.D. Park and S. Corson, Temporally ordered routing algorithm (TORA) version 1 functional specification, *IETF Internet Draft,* draft-ietf-manet-tora-spec-01.txt, Aug. 1998. Network simulator, Feb. 2002.

16

Modeling Computational, Sensing, and Actuation Surfaces

Phillip Stanley-Marbell
Diana Marculescu
Radu Marculescu
Pradeep K. Khosla
Carnegie Mellon University

16.1 Introduction

Recent years have seen the emergence of many efforts to embed computing resources in everyday environments. These efforts have ranged from the use of wireless sensor networks, to wired ubiquitous computing environments in homes and commercial installations. A promising culmination of these directions is that of general purpose flexible surfaces, with large numbers of computational, sensing, and actuation elements embedded in them. Thin and flexible sheets of general purpose active materials could find use in a variety of commercial and household applications. These materials with embedded computation, sensing, and actuation capabilities, may be deployed cheaply over large surfaces, for both the interiors and exteriors of buildings, automobiles, marine vessels (e.g., as a hull lining), and aerospace applications. Such active or computational surfaces will take advantage of their large contiguous spatial extents, and the ability to actuate these surfaces. Such an active material with embedded actuators might be used in building structures that self-repair, or adapt to weather conditions.

16.1.1 Computational Surfaces

Technologies being developed for wireless sensor networks are targeted at enabling the use of cheap, miniscule, discrete sensing devices for monitoring. Such devices, when either dispersed over large areas

or embedded into commodity items, enable the gathering of data and monitoring of phenomena. Wireless networked sensors will enable more efficient tracking (e.g., of items in a warehouse) continuous monitoring of inhospitable or remote environments, and will generally enable a significant increase in our abilities to extract useful information from our environments.

Beyond simple sensing tasks which can be encapsulated in discrete networked sensors, many opportunities exist for "intelligent" materials with embedded sensing, computation and more importantly, actuation capabilities. Such a platform is the macro-scale analog of micro-electro-mechanical systems (MEMS): it integrates computational, sensing, and mechanical actuation devices into a general purpose material surface. Surfaces form an integral part of the design of structures. The ability to control and sense phenomena over large surfaces is, however, not easily achieved by straightforward extension of the capabilities of discrete sensors. Although it is possible to employ large numbers of wireless networked sensors over surfaces, the use of individual, discrete sensors does not enable the use of actuation, or take advantage of the possibility of using less power- and cost-intensive wired networking between devices.

The technologies to enable the inclusion of computation, sensing, and actuation arrays in such surfaces, are not a simple extension of any existing technologies. The manner in which the devices will be interconnected will most likely be through the embedding of conductors in the surface, however, it is premature to tell whether this will indeed be the norm. Despite these uncertainties, a common set of issues need to be addressed to make such computational surfaces a reality.

The physical construction of such substrates will vary based on their applications. For example, for the purposes of commercial applications such as "smart" lining materials for applications such as automobiles, aircraft, or marine vessels, a flexible polymer- or carbon-fiber–based substrate might be desirable. On the other hand, for other applications, a woven substrate might be desirable. Regardless of the actual method of fabrication, these substrates will share the following common properties:

- General purpose. It will be desirable to obtain such platforms as a general purpose programmable substrate. Instead of constructing custom systems, they will be obtained in units of area and programmed for application specific purposes. Differentiation between products will be achieved by the area density and types of computational, sensing, and actuation devices, as well as the type of the substrate material (e.g., polymer substrate, carbon fiber, or woven materials).
- Large surface area. The spatial extents of the substrate can be harnessed by applications which benefit from the combination of computational resources, sensing, and actuation over a large area.
- Computation. Large numbers of computational devices embedded into surfaces at low cost. The use of large numbers of devices will be driven not by a requirement for increased performance, but instead for fault-tolerance, as well as the desire to spatially distribute computational resources over the surface. If the platform will be obtained in units of area, a swath of material cut from a larger piece should still be programmable and usable.
- Sensing. A major benefit of the large area systems will be their spatial extent, which makes them particularly useful for sensing applications such as active antenna arrays. Such arrays for many types of signals (e.g., acoustic or RF) can take advantage of such large intelligent surfaces, to augment applications such as speaker location or ultra-wideband radios. For example, a surface being employed as an antenna array might adapt its shape to achieve better signal reception.
- Actuation. It is possible to embed many types of actuation devices in these large area surfaces. Actuating strands such as shape-memory-alloys, can be embedded to enable the surfaces to change shape in response to data obtained from the sensors and driven by the computational elements. Other possible actuators include materials such as those which change color or reflectivity in response to signals, as well as heating and Peltier cooling elements.

Many challenges must be addressed to make such a platform a reality. The materials aspects of these substrates may indeed be the easiest to address. Although significant benefits in integration could be derived from customizing device-packaging technologies for these flexible large area computation and sensing arrays, it will nonetheless be possible to employ off-the-shelf components, which have been

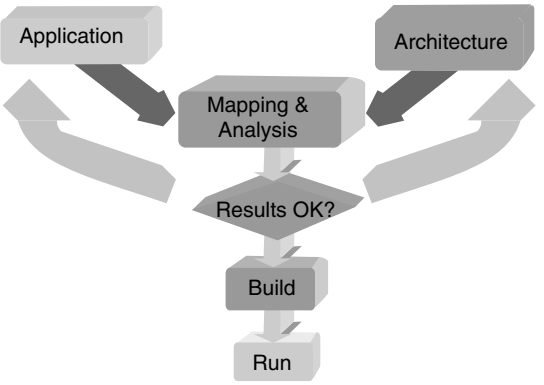


FIGURE 16.1 Classical static design cycle: no remapping occurs after the initial design is built.

packaged and optimized for traditional computing platforms in the first generations of systems. The greater challenges, we believe, lie in ensuring reliability in applications, particularly in the computer-aided design (CAD) methodologies, systems software, and programming language technologies needed to enable the harnessing of the unique capabilities of these systems. To address these issues, modeling frameworks and prototype systems are required to evaluate the use of the platform. To assess the efficacy of proposed designs, appropriate metrics that take into consideration the unique properties (e.g., restricted energy resources, performance constraints, reliability, and battery subsystem nonlinearities) must be employed.

Techniques to program such networks are required, permitting useful applications to be constructed over the defect and fault-prone substrate. In the classical design cycle (Figure 16.1), the application is mapped onto a given platform architecture, under specified constraints (e.g., performance, area, and power consumption). When these constraints are met, the prototype is tested, manufactured, and used for running the application. In the platforms of interest (Figure 16.2), the substrate is comprised of large numbers of interconnected computing elements, with no prescribed functionality. To achieve high yields, as well as high fault-tolerance later in the lifetime cycle, regularity is important. An application must be partitioned to expose concurrency. At system startup, the partitions of the application are mapped to hardware, so as to optimize different metrics of interest (e.g., quality of results, power consumption, operational longevity, and fault-tolerance) and later remapped whenever operating conditions change.

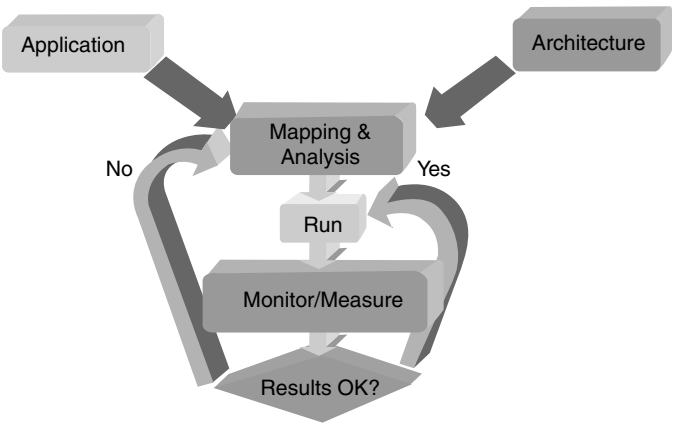


FIGURE 16.2 Dynamic continuously adapting living designs: through continuous online monitoring, the mapping of applications to the hardware substrate may evolve over time.

This chapter presents a conceptual framework and a simulation infrastructure for modeling several aspects of large surface arrays of computational, sensing, and actuation devices. Together, these provide a foundation for possible research directions. The next three sections present the conceptual framework, and Section 16.5 follows with a description of computation, communication, failure, and battery modeling in the simulation framework. The chapter ends with a summary of the presented ideas and possible directions for research in this new field.

16.2 Colloidal Computing

The model of colloidal computing (MC^2) [1] proposes local computation and inexpensive communication among computational elements: simple computation particles are "dispersed" in a communication medium which is inexpensive, possibly unreliable, yet sufficiently fast (Figure 16.3). The concept of colloids is borrowed from physical chemistry [2].* In the case of unstable colloidal suspensions, colloidal particles tend to coalesce or aggregate together due to the Van der Waals and electrostatic forces among them. Coalescing reduces surface area, whereas aggregation keeps all particles together, without merging. Similarly, the resources of a classic system are coalesced together in a compact form, as opposed to the case of colloidal computation where useful work can be spread among many, small, possibly unreliable computational elements that are dynamically aggregated depending on prevailing needs (Figure 16.4). Dynamic or adaptive aggregation is explicitly performed whenever operating conditions change (e.g., failure rate of a device is too high or battery level is too low) — a "stable" configuration is one that achieves the required functionality, within prescribed performance, power consumption, and probability of failure limits. The mapping and reconfiguration process of the application onto the underlying architecture is achieved via explicit mechanisms, as opposed to classic computing systems where mapping and resource management is done via implicit mechanisms.

The MC^2 model [1] was previously proposed to model both the application software and architecture platform. Most of the applications under consideration consist of a number of computational kernels with high spatial locality, but a low degree of communication among them. Such kernels (typically associated with media or signal processing applications) can thus be mapped on separate computational "particles" that communicate infrequently for exchanging results.

Reorganization and remapping requires thin middleware or firmware clients, sufficiently simple to achieve the required goals without prohibitive overhead. In addition, fault and battery modeling and

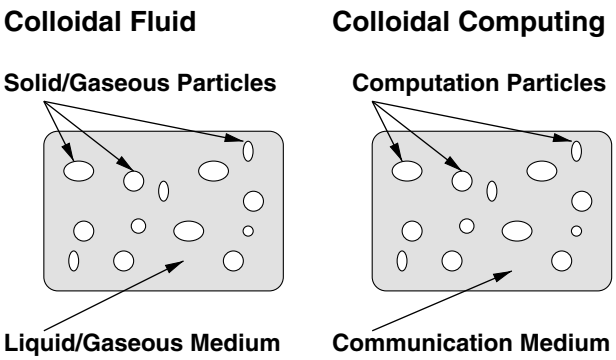


FIGURE 16.3 Colloidal computing model: analogy to colloidal chemistry [1].

**Colloid* [käl'oid] = a substance consisting of very tiny particles (between 1 nm and 1000 nm), suspended in a continuous medium, such as liquid, a solid, or a gaseous substance.

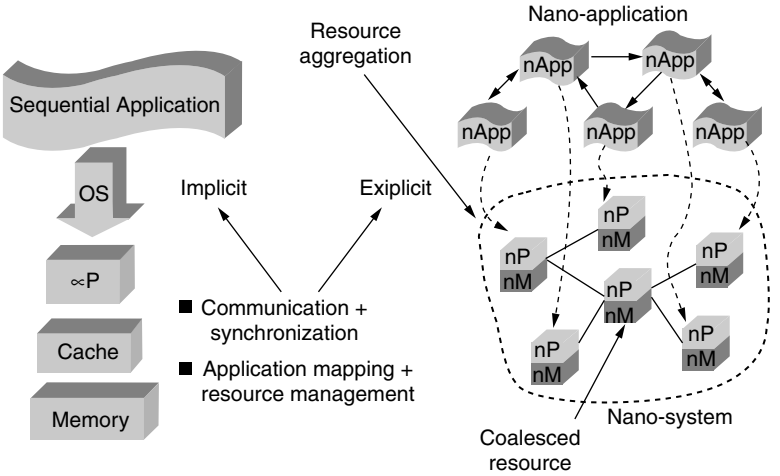


FIGURE 16.4 Coalesced vs. aggregated resources: partitioning applications to expose concurrency [1].

management become intrinsic components for achieving requisite levels of quality of results or operational longevity. We describe in the following sections, some of the issues that are critical to application lifetime, namely, application partitioning, followed by communication and fault management.

16.3 Application Partitioning

An issue of concern in mapping applications to hardware is that of concurrency. Given that the hardware substrate will contain many computational particles distributed on large surfaces, it is of crucial importance to expose the concurrency available in applications. The methods by which such concurrency may be extracted are a very interesting research avenue in their own right.

16.3.1 Driver Application: Beamforming

Beamforming consists of two primary components — source location and signal extraction. It is desired to detect the location of a signal source, and "focus" on this source. In a classic implementation, signals from spatially distributed sensors are sent to a central processor, which processes them to determine the location of the signal source and reconstruct a desired signal. Each received sample is filtered, and this filtering could indeed be performed at the sensor. Figure 16.5 illustrates the organization for a wired network of sensors used to perform beamforming.

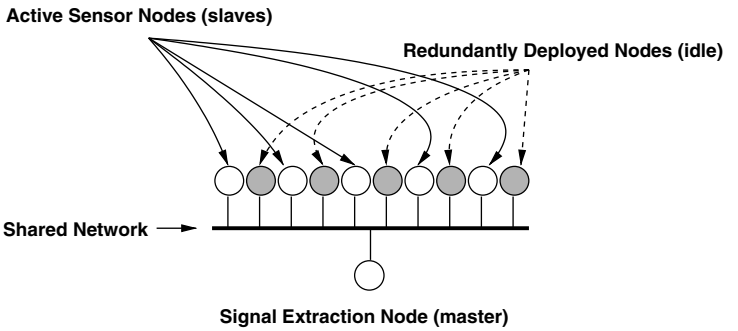


FIGURE 16.5 Beamforming in a wired sensor network.

In contrast to a classic implementation, the beamforming application as depicted in the figure is partitioned for execution over a network of processing devices. The filtering operation on each collected sample can be considered to be independent of other samples, thus it could be performed individually at each sensor node (slave node). The final signal extraction need only be performed at one node (master node). This division of tasks scales well with increasing number of sensors because the complexity of processing at each sensor node remains the same, and the only increase in complexity is in the number of filtered samples collected at the master.

Our example system operates in periods. During each period, all the slaves collect samples, filter them, and send the filtered samples to the master. The duration of the sampling period will vary for different applications of beamforming. In the case of beamforming for speech applications, an overall sampling rate of 8 KHz is sufficient. For geophysical phenomenon, a sampling rate of 1 KHz is enough, whereas for tracking the motions of animals, a sampling rate of 10 Hz is sufficient. In the analysis used throughout the rest of the chapter, a sampling rate of 10 Hz corresponding to a 100 msec sampling period is used.

The communication messages between the master and slave nodes consist of 4-byte data packets containing the digitized sample reading. When the battery level on any of the slave nodes falls below a specified threshold, the slave application attempts to use its remaining energy resources to migrate to one of the redundant nodes. If migration is successful, the slave application resumes execution on the redundant node, and adjusts its behavior for the fact that it is now executing on a different sensor, which is detected when it restarts. The migrated application code and data for the slave application is small (only 14 KB). The application on the processing elements with attached sensors implements a 32-tap FIR filter, and consists of 14 KB of application code and 648 bytes of application state. The application mapped on the sample aggregation (master) node performs a summation over the samples. The sequence of messages that are exchanged between the master and slaves during normal operation, and between the slaves and redundant nodes during migration is illustrated in Figure 16.7.

16.4 Communication Architecture and Fault Management

Achieving reliable computation in the presence of failures has been an active area of research dating back to the early years of computing [3,4]. Unlike large networked systems, in which failure usually occurs only in communication links or in computational nodes and communication links with low correlation, in the platform of interest, nodes and links coexist in close physical proximity and thus witness high correlation of failures.

It is assumed that the application is initially mapped at system startup for given quality of results (QoR), power and fault-tolerance constraints. As operating conditions change (e.g., permanent failures due to wear and tear, or intermittent failures due to battery depletion), the entire application (or portions of it) will have to be remapped, or communication links rerouted (Figure 16.6). Such reconfiguration mechanisms assume that redundancy exists for both nodes and links. In a fixed infrastructure, the logical implementation of redundancy is to replicate resources, with one resource taking over on the failure of the other. Upon such failures, applications must be remapped, for example, by code migration or remote execution. Code migration is generally a difficult problem, as it could, in the worst case, require the movement of the entire state of an executing application; however, migration of running applications can be greatly simplified by restricting the amount of application state that must be preserved.

16.5 Simulation Infrastructure

To investigate system architectures and programming models for the platform, it is necessary to be able to model it at the level of detail of its computation and inter-device communication. Using high-level behavioral models will be insufficient, as such an approach will fail to capture the interplay between computation and communication performance, as well as computation and communication failures, and their effects on performance, power consumption, and system reliability.

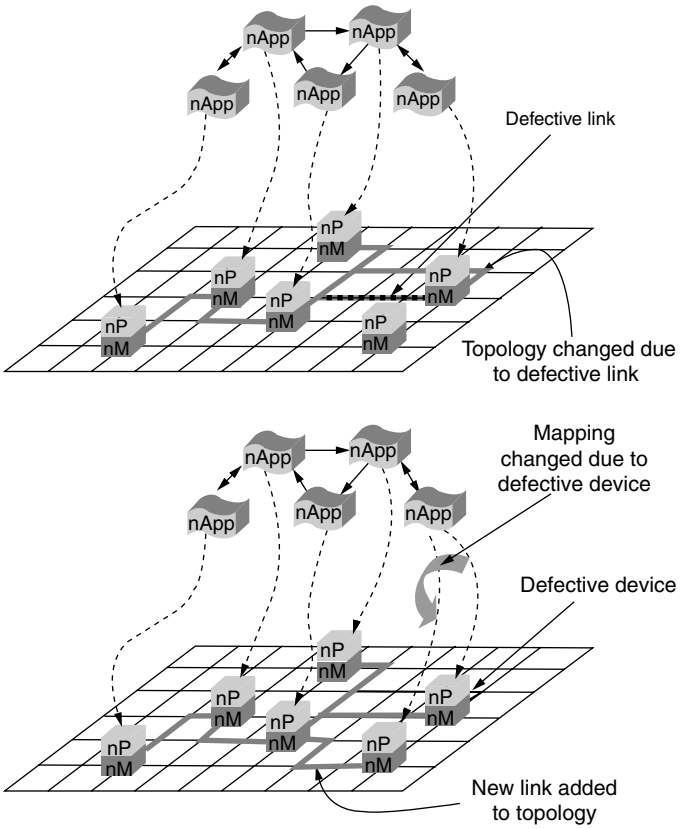


FIGURE 16.6 Dynamic reconfiguration in the presence of failures: in the event of a defective link (top), the network topology mapping is adapted to circumvent the faulty link. Likewise, in the case of a defective node, mapping and topology change to use a redundant node and a different link (bottom) [1].

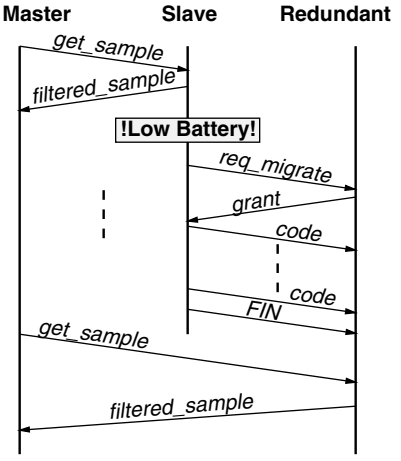


FIGURE 16.7 Messages exchanged in a beamforming application.

Any given platform will consist of multiple processing elements, distributed over a spatial extent in the surface. Such processing elements may either be general purpose programmable devices such as microcontrollers, or might be programmable logic devices. The processing devices will communicate over some communication infrastructure, whose topology might take advantage of the two-dimensional nature of surfaces. Failures in both the devices and in their interconnection networks will be common. These failures might be predictable, as in the case of depletion of energy resources, or they might be unpredictable, such as a defect in a communication or power conductor leading to intermittent communication or device failure.

It is therefore desirable for a simulation infrastructure to support the cycle-accurate simulation of multiple processing elements, for both general purpose processing and programmable logic devices. It must likewise enable the modeling of bit-level communication, in user-defined communication topologies. Failures in both processing devices as well as the communication links that interconnect them should be modeled, along with the relevant power dissipation for both computation and communication, and the effects of the current discharge profiles on power sources such as battery subsystems.

To support the investigation of CAD methodologies, system architectures and programming models for the hardware platforms of interest, we have developed a simulation framework [5] that aims to address the aforementioned modeling issues. The simulator models:

- Processing devices. The simulator permits the instantiation of multiple processing elements. Each instantiated element is modeled at the level of instruction execution. Two different processor architectures are modeled: the Hitachi SH3 architecture and the TI MSP430. Each processing node may have instantiated with it, one or more network interfaces, and each of these can be connected to an interconnection link.
- Interconnection links connecting the processing nodes. Interconnection links may be instantiated as necessary to create networks, and the network interfaces of processing nodes are attached to links. The links may be configured for variable transmission delay (link speed), link frame size, link failure probability, and link failure modes.
- Batteries and DC-DC converters. Each processing node is associated with a source of energy. The first-order effects of discharge rate on the battery cell and DC-DC converter efficiency are modeled by the simulation framework.
- Failures. The failure rate, average failure duration, and failure probability distribution of both processing nodes and interconnection links may be specified. Correlated failures between nodes and links may also be enabled, by specifying appropriate correlation coefficients.

The following describe each of the components of the simulation framework in more detail, motivating the need to perform modeling at the level of abstraction employed.

16.5.1 Processing Devices

At the core of the simulation framework is the modeling of instruction execution. Modeling applications at the level of detail of the simulation of the execution of their compiled code, make it possible to employ the simulation framework as a debugging platform for actual prototypes. It also makes it possible to determine important interactions between the requirements of computation, communication, and reliability, and the effects of these constraints on power consumption.

For example, Figure 16.8 plots the variation in two indicators of performance for the beamforming application implemented over the simulation framework. The beamforming application, as previously described, consists of two phases, which require differing amounts of network bandwidth. For both of these phases, however, increasing network speed does not indefinitely lead to increased performance, and may actually lead to reduced system performance. The "migration" phase of the said application, where the performance is depicted by the "average migration cost" curve in Figure 16.8, can actually not keep up with increasing network performance beyond a link speed of 1.6 Mb/s. This is because the

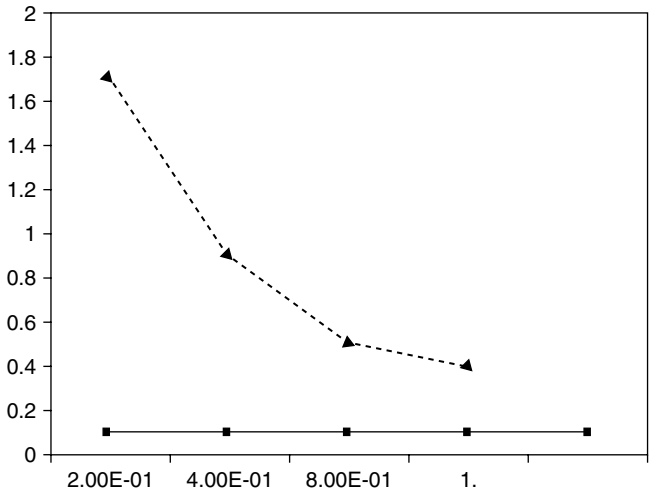


FIGURE 16.8 Variation in performance (average sample inter-arrival time and average migration cost) with communication link speed for an example application.

application attempts to transmit data at the fastest rate possible on the network, however, at this communication rate and for the modeled processing speeds, the receiving node does not have sufficient time (i.e., its computation is not fast enough) between arriving data frames to copy data it has received, before it is overwritten. Such delicate interplay between computational resources and networking resources is easily missed in high-level simulation (e.g., when employing a simple behavioral model of the entire system implemented in a high-level language).

The simulation framework includes two different architectural models, one for the Hitachi SH architecture, based on the Hitachi SH3 SH7708 (Figure 16.9), and the other of the Texas Instruments (TI) MSP430 architecture (Figure 16.10). Support for new architectures is easily added, and requires primarily the addition of code for implementing instruction decode and execution. The modeling of on-chip structures, such as interrupt generation, caches, and memory interfaces, as well as some standard peripherals, such as a network interface, is shared across the different architectures.

The Hitachi SH3 model includes detailed modeling of the CPU core, on-chip cache, and on-chip peripherals such as an RS-232 Universal Asynchronous Receiver/Transmitter (UART). It incorporates two complementary means of estimating the energy cost of application software — an empirical instruction level power model and circuit activity estimation. The instruction level power model functions by assigning to each instruction executed, an energy dissipation based on empirically measured values, scaled if necessary for a given operating voltage and frequency, as the model supports dynamic scaling of both operating voltage and frequency. Employing this simple energy estimation scheme enables fast simulation, which is critical because the framework is often used to simulate such platforms consisting of tens of processing devices. Although simple, the employed instruction level power estimation has been demonstrated to be within 6.5% of measured values for the hardware it models [6]. The instruction level power model can be augmented with a circuit transition activity estimation, which reports, for each simulation cycle, the signal transition activity on the address and data buses, in the register file, the program counter, and pipeline registers. The SH3 core model provides six levels of detailed simulation, enabling a tradeoff between power estimation accuracy and simulation speed [6].

The Texas Instruments MSP430 architecture model provides functional simulation of the processor and its peripherals for the MSP430F11 series of microcontrollers. Unlike the SH3 model, it currently provides only functional modeling of the modeled microcontroller to enable applications compiled for a prototype system to be modeled and debugged in the simulation framework.

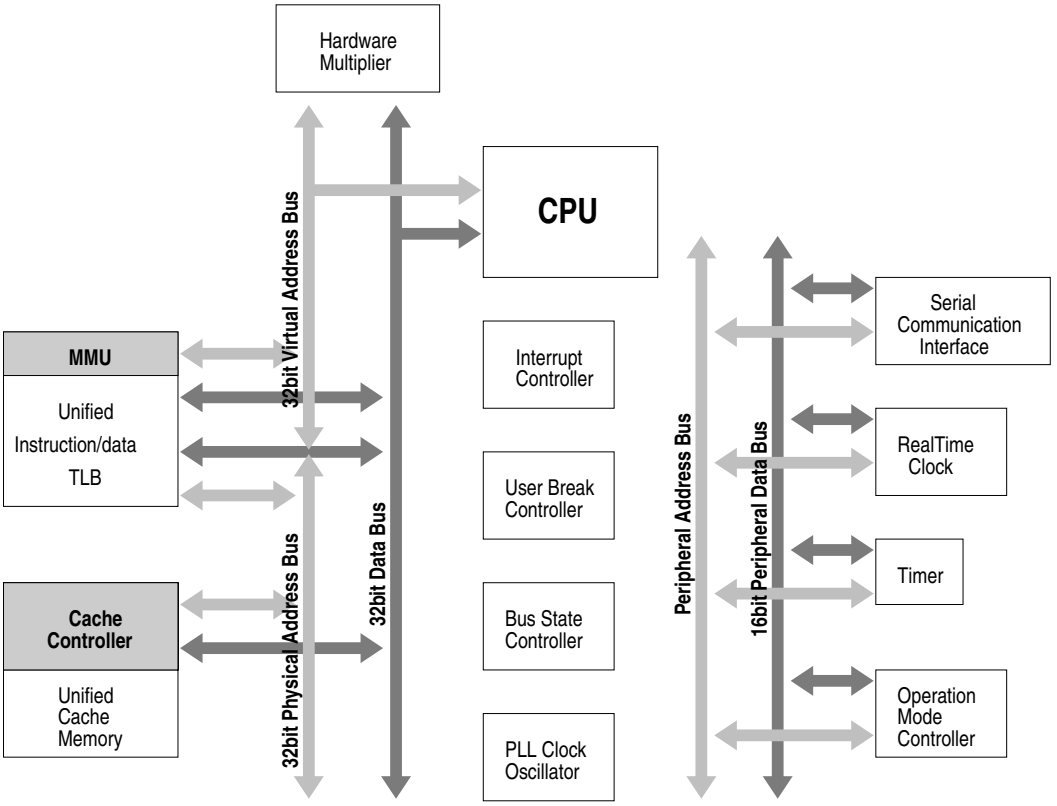


FIGURE 16.9 Hitachi SH3 architecture.

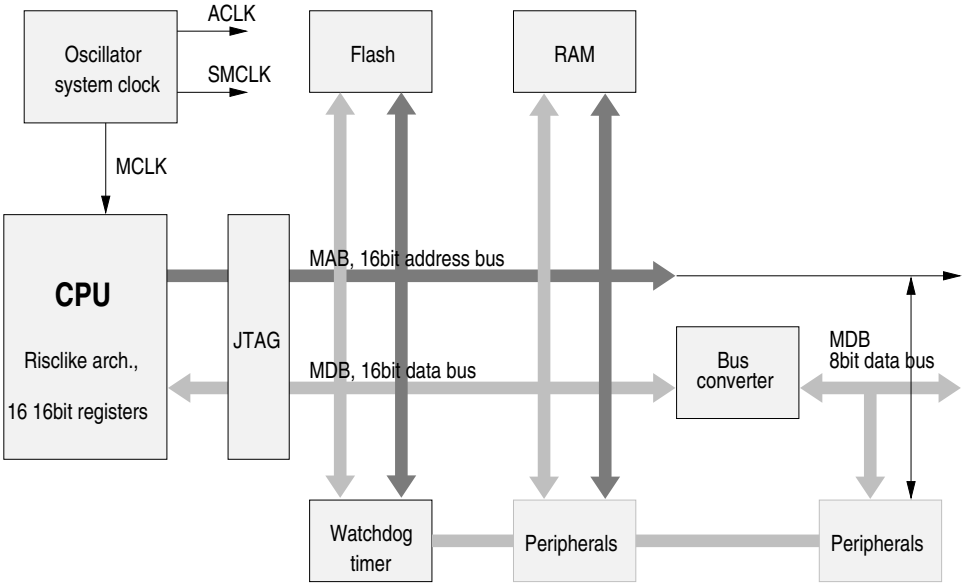


FIGURE 16.10 Texas Instruments MSP430 architecture.

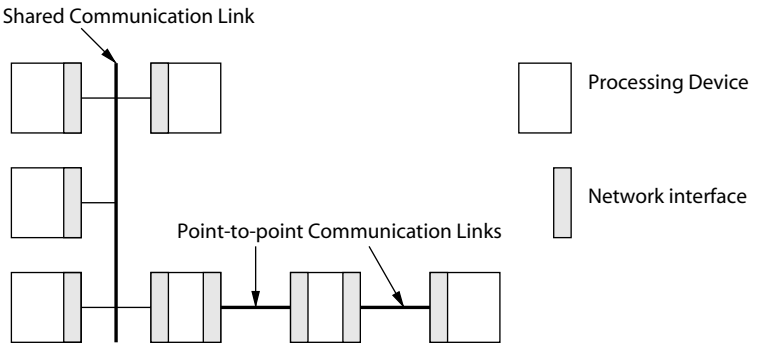


FIGURE 16.11 An example of communication topology.

16.5.2 Communication

The communication modeling architecture in the simulation framework enables the construction of a variety of communication topologies, such as that illustrated in Figure 16.11. Processing elements communicate with each other through their network interfaces, which are connected to communication links. The behavior of the network interface modeling is independent of the processing core model chosen for a given device. The properties of a modeled communication links and interfaces are flexible and parametrizable, enabling them to be configured to model the properties of media ranging from one with properties like RS-232, to one that behaves like Ethernet.

Each communication interface on a device must be associated with a communication link. Each communication link or network segment may be configured for the following specific properties:

- Frame size. Data is transmitted on a communication link in groups of bytes referred to as a "frame."
- Propagation speed. The propagation delay specifies the speed at which a signal travels in the communication medium, over the communication link. When modeling wired communication, this is taken to be the speed of light. Nodes in the simulation can have associated with them a location in three-dimensional space, which will then be used in conjunction with the propagation speed to determine the propagation delay. For most simulation scenarios, however, this parameter can be ignored.
- Transmission speed. The transmission speed specifies the number of bits that are modulated per second, or the bit-rate of the communication medium.
- Maximum simultaneous accesses. Specifying a maximum number of simultaneous accesses permits a medium to be configured to behave, for example, either as a carrier sense multiple access with collision detection (CSMA/CD) medium, or as one that employs frequency division multiplexing (FDM).
- Failure probability and maximum failure duration. These are discussed further in the description of the failure in Section 16.5.4.

To ensure network interfaces are always compatible with the networks to which they are attached, network interfaces inherit the aforementioned properties from a network segment to which they attached. The transmission and receive power consumption of a network interface may, however, be configured independently of the properties of the link with which it is associated. The simulation of data transmission and receipt is kept cycle-accurate with respect to computation. The granularity at which data is transferred from one device to another is determined by the smallest cycle time of all the modeled processing devices.

16.5.3 Battery Subsystem

The simulator includes a detailed discrete-time battery modeling engine based on [7]. In brief, the model takes into account properties of battery cells, such as dependence of battery terminal voltage on the state

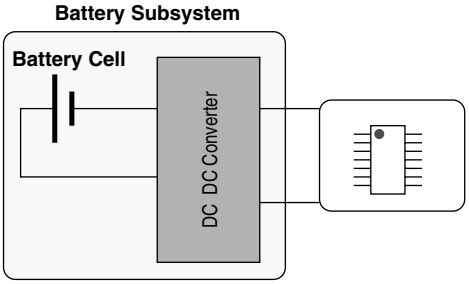


FIGURE 16.12 Organization of battery subsystem: the DC-DC converter is required to obtain a constant voltage to power electronics, due to the dependence of battery cell terminal voltage on battery state of charge.

of charge (SOC) of a battery, dependence of usable capacity on discharge rate, and dependence on the rate of change of current over time. To provide a constant voltage to the powered electronics in the face of variation in battery terminal voltage over time, a DC-DC converter provides voltage stabilization, at the cost of a loss due to inherent inefficiencies in the conversion. A simple organization of a battery-powered system (Figure 16.12) illustrates this further.

To model different types and sizes of batteries and DC-DC converters, the model (and its implementation in the simulator employed in this work) uses lookup tables (LUTs) and additional fixed parameters to store the characteristics of specific batteries. The default battery characteristics employed in our implementation are those for a lithium ion cell from the Panasonic CGR18 family. The DC-DC converter characteristics employed are those for a Dallas semiconductor/Maxim MAX1653 device. User LUTs may be loaded into the simulator to mimic other device's characteristics, for both the battery cell and DC-DC converter. Figure 16.13 plots the dependence of battery terminal voltage with time for a nominal discharge rate of 150 mA. The data in Figure 16.13, although plotting the voltage at the terminals of the battery cell, also includes the effect of DC-DC conversion, and depicts the lumped behavior of the battery cell if the battery subsystem were attached to electronics that had a constant current draw of 150 mA.

The components of the battery properties are illustrated in Figure 16.14. The parameters of interest in this work are V_t, a measure of the rate of discharge, V_{rate}, a time-sluggish (i.e., low-pass filtered) version of V_t, V_{lost}, which models the dependence of battery terminal voltage on the magnitude of V_{rate} for a

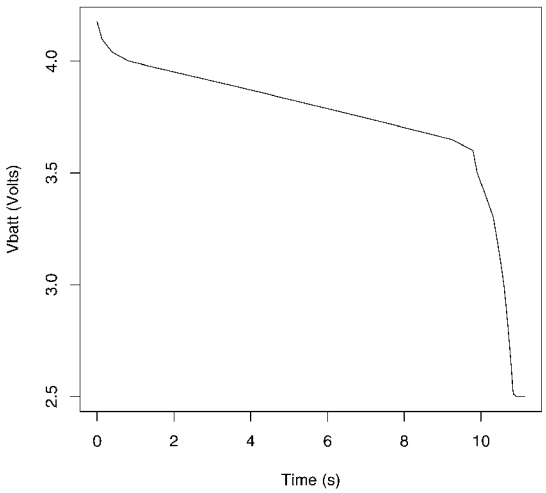


FIGURE 16.13 Variation of battery cell terminal voltage over time for a nominal current draw of 150 mA from outside the battery subsystem.

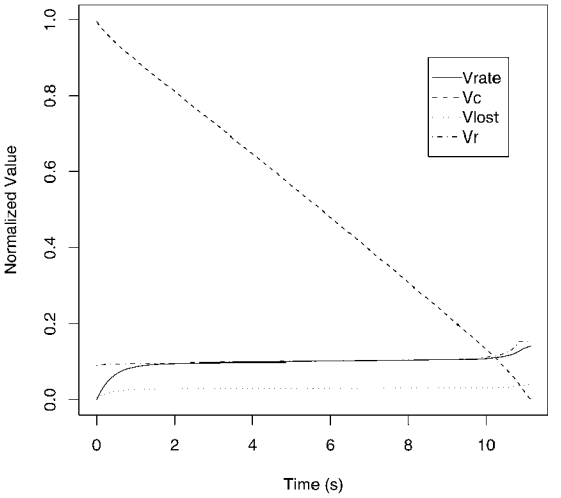


FIGURE 16.14 Variation of components of a battery model with time for a nominal current draw of 150 mA from outside the battery subsystem.

particular battery type (from an LUT). Last, V_C models the instantaneous state of charge, taking into consideration V_{lost}.

16.5.4 Modeling Failures

The simulation framework models failures in both processing devices and communication links. Failures in processing devices manifest as intermittent stalls of the entire processing device for the duration of the failure. Failures in communication links manifest as intermittent loss of carrier for the duration of the failure. Failures, such as bit errors introduced into the communication stream and into device computation, are not currently supported, but are planned. For both failures in devices and communication links, the failure rate and maximum failure duration are configurable. Correlated failures between processing devices and communication links can be modeled by specifying appropriate correlation coefficients for a given node-link pair.

16.6 Conclusion

New technologies often pose new challenges in terms of system architectures, device architectures, and sometimes, models of computation. The technology of interest in this chapter is that of computational, sensing, and actuation surfaces, which are flexible meshes of material containing large number of unreliable, networked computing elements, sensors, and actuators. The challenges addressed herein were those of design methodologies, system architectures, modeling, and fault-tolerance.

Computational, sensing, and actuation surfaces inherently have high defect rates, as well as high fault rates, and thus must by necessity provide mechanisms for extracting useful work out of the unreliable substrate. By employing a detailed simulation infrastructure designed to enable the simulation of the computation, communication, power consumption, and battery discharge characteristics, the dynamic adaptation of applications in the presence of faults can be investigated.

16.7 Acknowledgments

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) Information Processing Technology Office, under contract F33615-02-1-4004, and the Semiconductor Research Corporation, under grant 2002-RJ-1052G.

References

[1] R. Marculescu and D. Marculescu. Does Q = MC²? (On the relationship between quality in electronic design and model of colloidal computing). *Proc. IEEE/ACM Intl. Symp. on Quality in Electronic Design (ISQED),* March 2002.

[2] R. Rajagopalan and P.C. Hiemenz. *Principles of Colloid and Surface Chemistry.* Marcel Dekker, New York, 1992.

[3] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies,* pp. 43–98, 1956.

[4] M.D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Trans. on Comput.,* c-27(6):540–547, June 1978.

[5] P. Stanley-Marbell. *Myrmigki Simulator Reference Manual.* Technical report, Center for Silicon System Implementation (CSSI), Department of Electrical and Computer Engineering (ECE), Carnegie Mellon University, Pittsburgh, PA, 2003.

[6] P. Stanley-Marbell and M. Hsiao. Fast, flexible, cycle-accurate energy estimation. *Proc. Int. Symp. on Low-Power Electron. and Design (ISLPED),* pp. 141–146, August 2001.

[7] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. *Proc. Conf. on Design, Automation, and Test in Europe (DATE*), pp. 35–39, January 2000.

III

Embedded Software

17

Low-Power Software Techniques

Catherine H. Gebotys
University of Waterloo

17.1 Introduction

Software can have a large impact on the average power, peak power, energy dissipation, and instantaneous power of the embedded processor core. In turn, average power is directly related to battery lifetimes. Peak power constrains the thermal design of the embedded system. In addition, the peak power affects the power supply design and instantaneous power can affect reliability and security. Today, low-power dissipation is critical for wireless communication devices, which demand long battery lifetimes, high reliability, low thermal dissipation, and high security. This chapter discusses the relationship between software and power. First, instruction-level models for predicting the average power and predicting the average energy of applications executing on an embedded processor are reviewed. An example of an instruction-level model combined with statistics is presented for a digital signal processing (DSP) processor. Next, recent research in instruction-level models for predicting instantaneous power of a processor core is discussed. Finally, new emerging applications of instantaneous power design utilizing software, specifically in security, are addressed.

The need for low-power dissipation in many general-purpose processor cores and DSP processor cores has created a need for further understanding of power in system on chip (SoC) devices. Architectural design for low-power and high-level transformations for low-power applications are becoming a well understood area of research [1]. Previous methods of estimating power at a high level using gate-level or architecture-level simulations were either inaccurate or too time consuming. For embedded systems designers, power measurement and optimization for code is very important, however, gate-level processor representations are not always available for estimating power. Instruction-level accurate power prediction tools for embedded processor cores are important. The equations for power measurement and prediction are discussed next.

The energy dissipation of a processor running a program [2], *E,* can be approximated by the product of the time required to execute the program (T), the average current (I), and the supply voltage (V_{dd}) as in Equation (17.1).

$$E = PT = IV_{dd}\,T = INV_{dd} \tag{17.1}$$

This equation ignores the additional power dissipation arising from leakage current and short circuit current [3]. The term T is equal to $N\tau$, where N is the number of clock cycles and τ is the clock period.

From Equation (17.1), a reduction in N (equivalently a performance increase) will always provide equivalent or higher reduction in energy as long as the new value of current, I, is equivalent or lower in value. In cases where I increases as N is decreased (i.e., performance improvement could be derived from the use of more parallel instructions which help create higher average current per cycle), the reduction in energy will be less than the reduction in N. In some cases (possibly rare) it may even result in increased energy dissipation if $p_i > p_n/(1 - p_i)$, where p_i is the fraction of I that is increased, and p_n is the fraction of N that is reduced.

For many embedded processor core applications, N is fixed (by the throughput requirements), making energy reduction techniques rely solely on reducing I or reducing power. Thus, I is an important parameter for embedded systems design that needs to be studied and predicted for software energy prediction. The problem becomes how to modify or generate processor code that is energy efficient or meets power constraints. An excellent review of system-level power optimization techniques and tools can be found in Benini and DeMicheli [1]. This chapter provides an in-depth review of experimental setups for measuring current of processors. Additionally, instruction-level average power modeling of embedded processors is reviewed. An example of modeling instantaneous power of a very long instruction word (VLIW) DSP processor at the instruction-level is outlined. Throughout the chapter, current, power, and energy models are discussed. This chapter concludes with emerging areas for instantaneous power prediction in security. The next section discusses predictive models of average power.

17.2 Software Models for Predicting Average Power

Instruction-level models for predicting the power dissipation of processors was investigated in Lee et al. [2], Tiwari et al. [4], Qu et al. [5], and Russell and Jacome [6]. These models were verified by real measurements of average power for both instructions and programs on the target processor board. The first section briefly describes the different experimental setups used in power research, followed by a description of the verified power-prediction models and their accuracy in the next section. A general review of other instruction-level power models is also presented, followed by a detailed example of utilizing a statistical approach to model building. Section 17.3 and Section 17.4 present recent research in building instruction-level instantaneous power models of embedded processors and applications of instantaneous power modeling to security, respectively.

17.2.1 Experimental Setups for Average and Instantaneous Current

Measuring the current drawn by the processor while executing an application has been used by many researchers to verify power models. The equipment setups vary from ammeters to oscilloscopes. The objective of measuring the current drawn by the processor while it is executing an application varies as well. In some cases, current measurements are used to obtain average power readings per instruction or per program, and in other cases the measurement is used to obtain the real execution times of a processor (i.e., including cache misses and memory stalls), which are too difficult to simulate.

For current measurements of a general-purpose processor and a DSP processor in Lee et al. [2] and Tiwari et al. [4], a current meter was used. This setup allowed power measurements for small programs (whose execution times were much less than 100 ms). These programs were repeated several times in a loop until a stable current reading could be obtained. These current readings were taken visually so

stability was important. Other researchers [7–9] have used this same type of experimental setup except the current meter (e.g., a Fluke 867B GMM) was more sophisticated, allowing sampled readings of current to be transferred to a workstation. This allowed variations in current readings to be averaged. In addition, the current measurements of longer programs could be supported. The user can set the period over which samples can be averaged and these averaged samples are then transferred to the workstation for further analysis. Thus, the requirement for stability of the readings and limitations of short programs as in Lee et al. [2] and Tiwari et al. [4] was not necessary. This type of setup (using HP34401A digital multimeter) has also interestingly been used to validate a runtime power estimator in Joseph and Martonosi [10]; however, here a shunt resistor was placed between the power supply and the processor's board power terminal. The voltage across the shunt resistor was measured and then divided to obtain current measurements. In Joseph and Martonosi [10], the chipset power was subtracted from the total board power to obtain the CPU power. Sinha and Chandraksan [11] also used a source meter to measure the current drawn by subroutines. This setup was used to analyze leakage current as V_{dd} changed.

An interesting experimental setup described in Chang et al. [12] used capacitors in between the power supply and the processor. This setup allowed the researchers to analyze specific cycle-by-cycle energy [12]. The minimum and maximum voltages on the capacitors were acquired in real time. Each cycle the capacitors were charged up by the current drawn by the processor and in the next cycle the capacitors were fully discharged. This technique allowed measurement of current effects from single cycle activity alone (without the influence of previous cycles). The capacitors were completely discharged before the next clock cycle; thus, only activity in the clock cycle of interest was measured. This setup allowed researchers to isolate cycle-by-cycle influences of instructions in the pipeline on the current draw.

Other researchers have measured power [6,13–16] using an oscilloscope. Oscilloscopes offer higher sampling rates and more accuracy than the digital multimeters. The oscilloscope in Wolf et al. [16] was used to accurately measure the execution time of applications. In Russell and Jacome [6], an oscilloscope was used to obtain the instantaneous power measurement of a single instruction (for an instruction-level power model). The instruction was repeated several times in a loop [6]. In the later case, a resistor was placed in between the power supply and power pin of the processor. Decoupling capacitors were introduced to reduce the voltage noise during current surges. Although instantaneous power was measured in Russell and Jacome [6] using an oscilloscope, the average power was calculated from this waveform over the loop body (which consisted of 100 instances of an instruction). In Wolf et al. [16], a resistor was again used, however, a custom experimental setup using differential amplifiers, an integrator, and an ADC was used to transfer and process the power readings into a logic analyzer. Power measurements per clock cycle were recorded using this setup. Nickolaidis et al. [28] utilized current mirror circuitry to obtain current measurements. This technique avoided the use of a resistor between the supply and the power pin (which typically may cause supply noise problems).

Instantaneous power was also captured in Muresan and Gebotys [13,14] and Muresan [15] by using an inductive probe (instead of a resistor in between the supply and power pin). An oscilloscope and pattern generator were both used to synchronize the program and oscilloscope. Additionally the pattern generator produced the clock signal as well as the trigger signal. This allowed accurate measurements of the instantaneous current over various sections of a program executing on the processor. The purpose of this setup was not to measure average current per instruction, but to measure and model the instantaneous power. Section 17.3 outlines the research resulting from this experimental setup for instantaneous power modeling. Other researchers [17] used a National Instruments data acquisition card to simultaneously read 16 power sources (at rates up to 1 million samples per second) in a portable laptop PC environment. This setup was used to evaluate power management algorithms. Researchers in the emerging area of security have also used oscilloscopes to measure instantaneous power [21,22,26]. Here, instantaneous power is analyzed to ensure no confidential information is leaked from the security application, to be further discussed in Section 17.4. The next section gives a brief overview of instruction-level power models, which have been researched.

17.2.2 Previous Instruction-Level Average-Power Models

Previously researched instruction-level power models for processors are briefly reviewed in this section. Some of these models have been verified with real power measurements or with lower-level power estimation tools. These models are discussed, followed by an illustration of building a power model using statistics and instruction-level power measurements. The power model is then verified with real power measurements of applications executing on the processor.

One of the earlier instruction-level models of power [2,4] was derived from a base power cost per instruction along with an overhead cost related to the next or nearby instruction. They achieved an accuracy of 10% but required characterization not only on a per instruction basis, but also for pairs of instructions and beyond. Some tools were developed that provided performance improvement in code in addition to power improvements. In several cases and where data was available, their results demonstrated that fewer instructions lead to faster code and lower energy. This approach has been utilized by several researchers to build instruction-level power models of various processors.

Russell and Jacome [6] measured instantaneous power of individual instructions across one loop iteration and used this in an instruction-level average-power model. Using statistics, their model concluded that for the two reduced instruction set computer (RISC) 32-bit processors considered, a model utilizing only the average power of all assembly instructions multiplied by the execution time (also determined from the oscilloscope) provided an 8% accurate model for energy with a 99% confidence level. Other researchers have also captured instantaneous power for purposes of building an average energy instruction-level model. In Nikolaidis et al. [28], the oscilloscope captured the instantaneous power of a single instruction embedded by NOPs. The power was integrated over one clock cycle to obtain the average energy measurement for the single instruction. Simunic et al. [18] extended an instruction-level simulator of an ARM processor for power along with energy models for board interconnect and memory. The total energy of this embedded system was modeled overall with a verified accuracy of 5%. Chang et al. [12] studied specific cycle-by-cycle energy, however, a power prediction model was not created. Instead, the research studied what factors influence the power dissipation of each instruction. For example, the analysis indicated that hamming distance between address values, and other switching had a significant influence on the power. Step-power analysis [19] uses power simulators to study effects of clock gating on maximum power consumption. Step power is defined as dp/dt, and it causes reliability concerns. It is studied to identify the causes of high step-power.

17.2.3 Example of Statistically Generated Model for Average Power

This section briefly describes a statistically generated model for power dissipation of a TMS320C5x DSP processor. The DSP processor's datapath has an accumulator register, product register (of the multiplier), and the input register to the multiplier, respectively. The memory addressing can support direct or indirect memory addressing modes. Offset addressing is also supported, but only one offset address register is available. Eight address registers are available, and a three-bit register points to the current one. The processor has address characteristics similar to many popular DSP processors. Similar to the DSP processor used in Lee et al. [2], parallel instructions are used, and some instructions have design features similar to other DSP processors (specifically that of two nonsequential instructions changing the same state). The TMS320C5x DSP processor along with the Fluke 867B digital multimeter was used. All experiments were repeatable. After the board was powered up, a warm-up period was allowed before any experiments were run. Figure 17.1 illustrates the current measurements over time when the board is initially warmed up. A series of NOP (no-operation) instructions were executed before and after each series of programs were run to calibrate any variation in power measurements due to temperature variation. In all cases, standard deviations were lower than 0.03 mA.

The average current for each type of instruction is recorded and used to generate a variable, x_p, which is detailed later in this section. Several DSP benchmark programs are also run with four different types of input data (e.g., voice and pseudorandom), and current measurements are recorded. The power

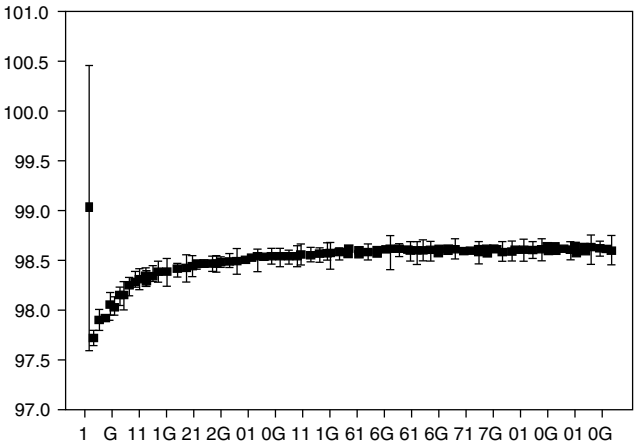


FIGURE 17.1 Warm-up current measurements approaching stability.

predictor model generation is based upon linear regression performed on a number of variables from the benchmark DSP code and x_p. The output is a model (or equation $y = f(x_p, \text{variables})$), which predicts current from only the DSP code itself (or variables extracted from it). The embedded systems designer then generates code for their application and uses it with the power-prediction model to predict current. Code is regenerated (using some technique such as rescheduling or rewriting the application) in an attempt to obtain code that meets the performance constraint and minimizes the predicted power dissipation.

DSP code for embedded types of applications, such as the fast Fourier transform, least means squares, high pass filter, and discrete cosine transform, were generated. The programs ranged from 60 to 150 instructions. Different schedules, addressing arrangements, and coding were used to study power effects. In many cases, different codes for the same filter were created with equivalent performance (e.g., using different schedules and address generation). One set of programs was used to generate the power prediction model, and a different (independent) set of programs were used to verify the model.

For each DSP benchmark program, a straight-line basic block code sequence was repeated several times and then placed within a loop. Each repeat of the program used a different part of the input data. For example if a DSP program used 40 words of speech data as input. The DSP program was repeated 100 times in a loop, performing computations on 4000 words of a continuous speech sample. This study would be repeated with pseudo-random generated data and other types of data. The different types of input data used were:

1. Random data generated from a pseudorandom number generator
2. A second set of pseudorandom numbers
3. Raw voice data from a voice sample
4. A second sample of voice data

Variables obtained directly from analysis of the DSP benchmark code are listed in Table 17.1. For example, IR in Table 17.1 refers to the average switching of data stored in the instruction register (available from the DSP code), whereas DABUS refers to the average switching of the data address bus.

A new variable, x_p, was added to the statistical methodology. The value of this variable, x_p, was created for each DSP program by summing the number of each instruction multiplied by the average current per cycle (measured with this instruction repeated several times in a loop) divided by the total number of instructions in the program. This approach is similar to that used in Lee et al. [2] and Tiwari et al. [4], however we use details of addressing and include this variable, which can be obtained directly from the code, among all other data independent variables to form a model. Furthermore, we do not have to use pairs of instructions and record their currents. We instead model the overhead or state with data-independent variables.

TABLE 17.1 Variables Used to Build Power Prediction Model

Variable	Average Switching in the
IR	Instruction register
PC	Program counter
ACC	Accumulator
PREG	P register
TREG	T register
ARi	Address register *i*
ARP	Pointer to address registers
DMEM	Data memory
DBUS	Data bus
DABUS	Data address bus
PMEM	Program memory
PBUS	Program bus
PABUS	Program address bus

A number of linear models were fit using the measured current as the dependent or y variable. Several independent or predictor variables (x) were considered, see Table 17.1, along with x_p. The models were fit using a least squares algorithm to minimize the distance between the observed data and the predicted data under the model. We assume that the y data is some linear function of the x, $y = f(x)$. The least squares equation predicting average power from x is given by the following linear equation $E(y|x) = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_k x_k$, where the b_i represents the least squares estimates of the population parameters, and $E(y|x)$ is the average or expected value of y given x. A stepwise selection method was used to automatically find the best model for predicting current. The model reported is an excellent model statistically; model adequacy tests have p-values < 0.001, where p-value is the observed level of significance.

The R^2 value is reported indicating the percent of variation in current accounted for by the model. The least squares equation is given predicting average current for the model (all coefficients are highly statistically significant, p-values < 0.001) and standard error of prediction are given for the maximum residual as another measure of accuracy of the model. The normality assumption for statistical tests and confidence intervals was verified using normal probability plots and histograms. The statistical package SPSS [20] was used for all statistical calculations.

Specifically 168 benchmark DSP programs (each repeated several times in a loop) run with different types of input data were executed on the DSP processor and average current was read from the meter. Using the average current measurements (obtained from the single instruction tests) and the variables extracted directly from the DSP programs, the variable x_p was obtained. The variable x_p together with the variables (also obtained directly from the DSP code) from Table 17.1 were then used by the linear regression algorithm (see details of statistical procedure outlined in the experimental section of the *SPSS User's Guide* [20]) to automatically form the equation for predicting current. The automatic power prediction model generation results are presented here.

For 168 cases (DSP benchmark programs), the stepwise selection procedure automatically produced the following model. The x variables automatically chosen due to their significance by the statistical procedure are listed in order of their importance in predicting energy: x_p, IR, DABUS. The value of R^2 for this model is 0.78 or 78% of the variation in current is accounted for by these three variables. The equation for predicting current from these x variables (where x_p, x_4 = IR, x_5 = DABUS) is $y = 27.41 + (0.38)\, x_p + (2.65)\, x_4 + (0.08)\, x_5$.

The standard error of prediction for the largest residual is 0.17. In other words we would be 95% confident that ±0.34 mA of the predicted value of current would contain the average current. The confidence interpretation and tests of significance depend on the assumption of a normal distribution of residuals. A histogram of the residuals was analyzed to verify this assumption. In Figure 17.3, this histogram clearly indicates a normal curve. The worst case residual was –1.7 mA, providing a maximum

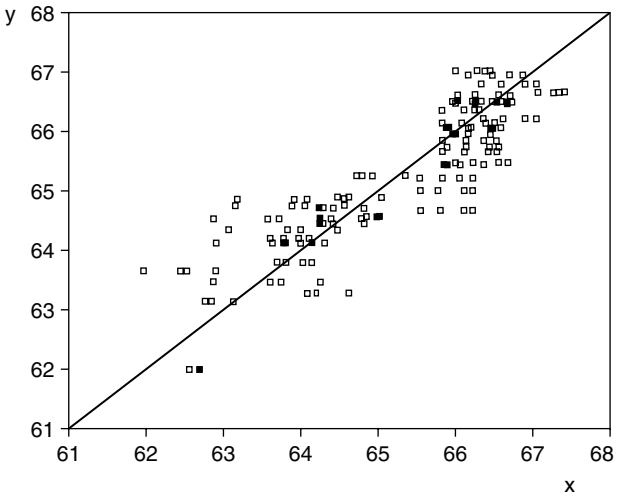


FIGURE 17.2 Measured current (x) vs. predicted current (y) in mA.

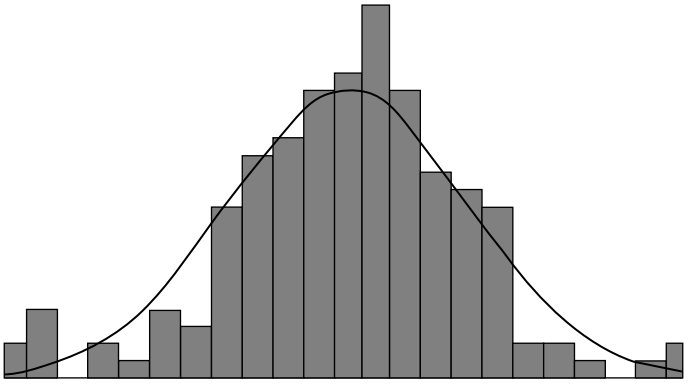


FIGURE 17.3 Histogram of the residuals indicating a normal curve.

worst-case error of 2.7%. The overall fit of the model can be seen in Figure 17.2 where the predicted current, y is plotted against the actual current, I, both measured in mAs.

Table 17.2 compares this statistically derived model to other statistically derived models that have access to higher level algorithmic variables or to more detailed switching details. The prediction ability of a higher-level model (algo for algorithmic) where only the number of each type of operation is used to predict power is very poor (see row 1 of Table 17.2 with R^2 value of only 0.63). For example, the number of additions (and/or subtractions), and the number of multiplications in the application are recorded as variables. A more detailed model (switching level in row 3 of Table 17.2) that records the actual average switching in registers and busses of the DSP processor provides a better R^2 value [7],

TABLE 17.2 Average Power Model Levels, Variables, Maximum Error, and R^2

Model	Example	Maximum Error, R^2
Algorithm level	No. of additions, multiplications	3.7%, 0.63
Instruction level	x_p, IR, DABUS	2.7%, 0.78
Switching level	No. of loads, PABUS, No. of subtracts, IR, DABUS	2.1%, 0.89

however, this requires an instruction-level simulator with switching activity to which embedded systems designers typically do not have access. The instruction-level power model (see row 2 of Table 17.2) compares very well in R^2 value to the more detailed switching level model (of row 3). More important, it is very suitable for embedded systems design because all inputs to the model can be obtained from the generated code itself along a one time only model generation phase using single instruction and DSP benchmark tests together with statistical optimization.

To further independently test out the validity or accuracy of the model, variables from other DSP programs (that were not used to derive the statistical model) were used in the previously presented equation for predicting current, y, and this predicted current was compared with actual current measurements. The predicted power or current value had an error less than 2.7% of the actual measured current for the different types of voice and pseudorandom data input. This approach has been used for various processors including a highly parallel DSP processor [7].

17.3 Instruction-Level Models for Predicting Instantaneous Power

Models of dynamic power at the software level have been researched in Muresan and Gebotys [13,14] and Muresan [15]. The current model is based upon summing instruction-level current models (gamma functions) together and using multiplicative factors to correct for block-to-block current variation at the higher application software level.

A simpler formulation than in Muresan [15], based upon processor clock cycles, is given next. The variable $i_{processor}(c)$ represents the current of the processor at clock cycle c.

$$i_{processor}(c) = i_{base} + \sum_{n=0}^{w} \sum_{instr} (\beta_{instr}) gamma_n x_{instr,c-n}$$

The variable $x_{instr,c-n}$ is a binary variable, which is one of the instructions, and *instr,* is executed at clock cycle *c-n,* otherwise it is zero. The i_{base} is the base current similar to base current in Lee et al. [2] and Tiwari et al. [4]. The parameter β_{instr} represents the amplitude of the current for instruction *instr.* The variable $gamma_n$ represents the current modeled (at 100 MHz and targeted for SC140 processor [15] with the parameter 0.0038) as a gamma function

$$gamma_n = (0.0038)^2 (n\tau + \frac{\tau}{2}) e^{-(0.0038)(n\tau + \frac{\tau}{2})}$$

where τ is the clock period, and $gamma_n$ uses the gamma value at the center of the clock period *n* (because this gamma function is actually a function of time but simplified here in clock cycles). For example, $gamma_0$ is the gamma value for a general instruction in clock cycle 0 when it starts executing. Whereas $gamma_4$ is the gamma value for an instruction which started executing four clock cycles ago. Note that more than one instruction can be executed in one clock cycle (thus supporting parallelism) and again the gamma values are summed. Figure 17.4 illustrates the measured power traces and superimposed gamma models for three separate types of instructions: *MOVE.2L (EA), Da:Db, MOVE.L #s32,C4,* and *EOR Da,Dn* in order of highest to lowest amplitudes. The first instruction loads two 32-bit words into two data registers. The second instruction loads an absolute 32-bit value into a control register. The lowest current draw was obtained from the exclusive or on two registers. For plotting purposes, the amplitudes for the exclusive or instruction (*EOR*) were multiplied by an additional factor of two. It is interesting to note that unlike previous research [28] which integrated the single instruction waveform over one clock period, the waveforms of this highly parallel processor extended over many clock cycles [15]. This is most likely due to the larger capacitance of the processor because it contains many more ALU units.

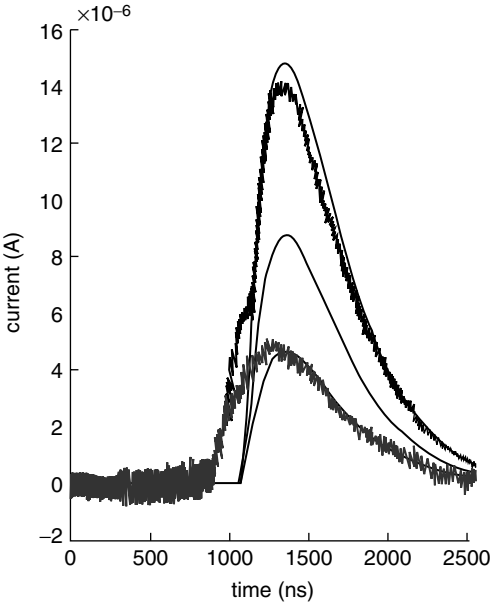


FIGURE 17.4 Gamma functions superimposed with current of *MOVE.2L (EA),Da:Db, MOVE.L #s32,C4,* and *EOR Da,Dn* (illustrated as double the amplitude) from highest to lowest amplitudes, respectively.

The application, where power is being modeled, is divided into blocks where the average instruction parallelism per block and variation of parallelism is used to create multiplicative factors. These factors correct for block-to-block current variation at the higher application software level. The multiplicative factors are derived statistically utilizing a benchmark set of applications. The final multiplicative factors are used for all subsequent current models representing instantaneous current of new application software. Results in Muresan and Gebotys [13,14] and Muresan [15] found that these current models captured over 94% of the real measured current variation. An example of the instantaneous current model is given in Figure 17.5 where the top waveform is the real current measured, and the lower waveform is the instantaneous current model based on gamma functions. The bottom arrow indicates the software execution time. In general, the gamma function could be retargeted to other processors by fitting it to their single instruction instantaneous power waveforms (through modifying 0.0038).

17.4 Emerging Applications of Instantaneous Power Prediction: Security

Security is crucial for today's portable devices including PDAs, cell phones, and other wireless devices. For example, some PDAs or cell phones are Internet-enabled and contain credit card information, others used in the healthcare industry contain confidential health information, and still other portable devices provide access to private corporate networks. In all these cases if the portable device is lost, it must be secure: specifically it must prevent unauthorized users from breaking into the portable device or obtaining any valuable information from the device. Even if the device is not lost, it may still be possible to obtain valuable information from the EM waves being radiated from the device while it is in use. One of the greatest feared attacks on SmartCards arose in the late 1990s [21,22] when it was demonstrated that the secret key could be determined by measuring the power (highly correlated with EM waves) drawn by the SmartCard processor. This is known as a power analysis attack. Since then, much research has concentrated on enhancing SmartCard security. However, portable devices also demand high security, yet are typically more complex than SmartCards. For example, they often have debug modes which can be used by attackers to access data without even knowing the users password or even download hostile code and thus are

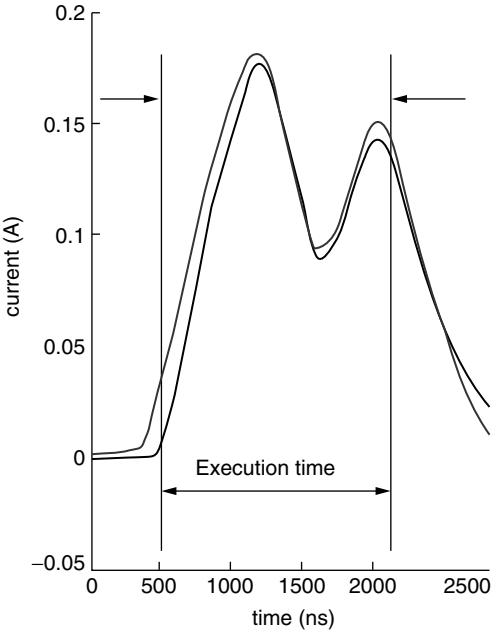


FIGURE 17.5 Real measured current (top) vs. simulated current of a program.

vulnerable to attack. This section introduces the power analysis attacks and gives examples of instantaneous power measurements in this security field.

The measurement of instantaneous power while a processor is executing an application (or a power trace) has been used in power-attacks of cryptographic devices, such as smart cards (typically 8- or 16-bit embedded processors). The equipment setup consists in general of an oscilloscope measuring the voltage over a small resistor placed in series between the processor supply pin (contact point on the smart card) and the supply (external to the smart card). In particular, the analysis of the variation of instantaneous power and statistical computations on a number of power traces can be used to detect data and algorithmic dependencies. This research studied the correlation of power variation with data values being manipulated and instruction sequencing. In the former case, known as differential power analysis attacks (DPA), encryption applications were analyzed [22]. In the latter case, known as a simple power analysis attack (SPA) [21], it was concluded that the correlation was significant and techniques such as random sequencing of instructions have since been researched. Typically, SmartCard applications are not time critical and energy dissipation is not a major concern because power is attained from the card reader (or ATM machine). Power attacks of more sophisticated processors with parallel instruction execution have more recently reported in Gebotys and Gebotys [23].

17.4.1 Simple Power Analysis

As an example of a simple power attack, consider a security algorithm running on a VLIW processor. The security algorithm implements elliptic curve point multiplication for NIST approved elliptic curve $y^2 + xy = x^3 + ax^2 + b$ over 163 bit binary fields (F_2^{163}) using prime polynomial $x^{163} + x^7 + x^6 + x^3 + 1$ [24,25] (using affine coordinates). Two power traces are plotted in Figure 17.6. The top power trace is a sum routine, and the bottom is a double routine from an elliptic curve point multiplication, widely used in public key cryptography. If an attacker can determine when a sum is being performed and when a double is being performed, the secret key can be easily determined. The detailed differences in the algorithm's power traces can be seen (i.e., sum routine does not have a loop in between the μl and square) from this figure. Thus, it can be seen that instantaneous power models are important for designing security applications. Figure 17.7 illustrates a more secure design where it is now more difficult to identify

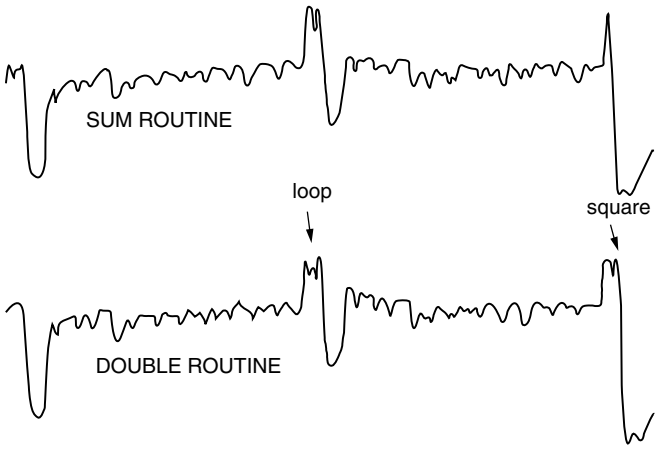


FIGURE 17.6 Current for eliptic sum (top) and double (bottom) routines.

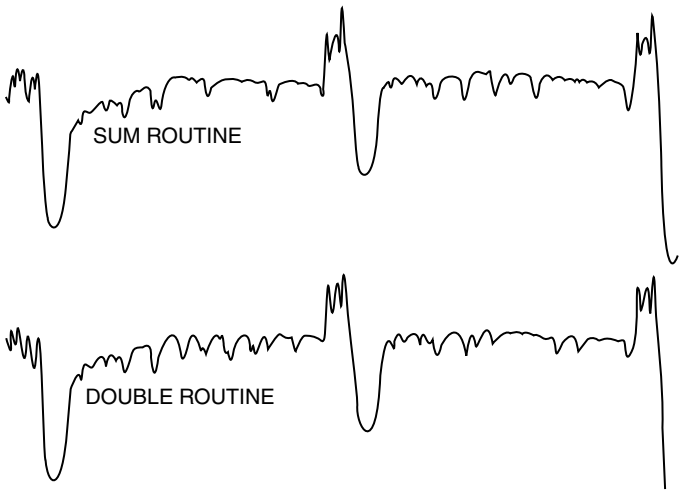


FIGURE 17.7 Secure current/power trace for sum and double eliptic curve routines.

the differences between the sum and double routine from the power traces. Further results can also be found in Gebotys and Gebotys [23] for prime fields.

17.4.2 Differential Power Analysis

Instantaneous power measurement is also crucial for verifying power–analysis security, particularly if an attack is able to acquire a large number of power traces from a SmartCard or some device, it may be possible to again obtain the key or reduce the search space size for enumerating through possible keys. Differential power analysis is an attack based upon the data-dependent switching activity component of power, particularly when data is placed on a processor bus the power reveals information about the data's hamming weight. With a sufficient number of power traces generated with different text inputs, it may again be possible to confirm key bits (and entire key values). Some hamming weights have been measured in Messerges et al. [26] for an 8-bit, 5-V, 4-MHz processor, however, hamming weights for a 32-bit, 2-V, 100-MHz VLIW processor, SC140, as plotted in Figure 17.8 for hamming weights 0, 4, and 6, are more difficult to determine. Nevertheless differential power analysis is still a threat as indicated by the differential signal plotted in Figure 17.9 (second plot from the top), whose differential peaks are greater than two standard deviations [27] (and therefore significant), acquired with 3000 power traces

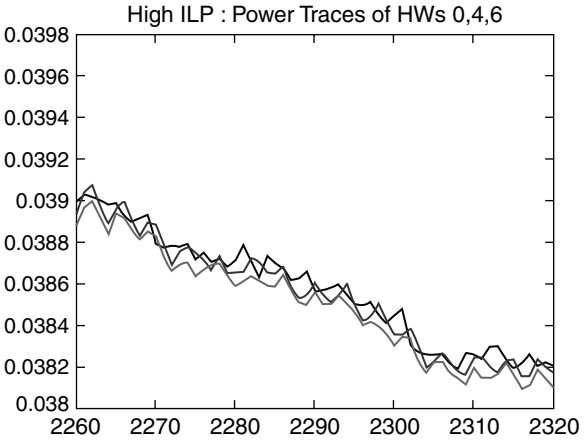


FIGURE 17.8 High, average, and low hamming weights (mA).

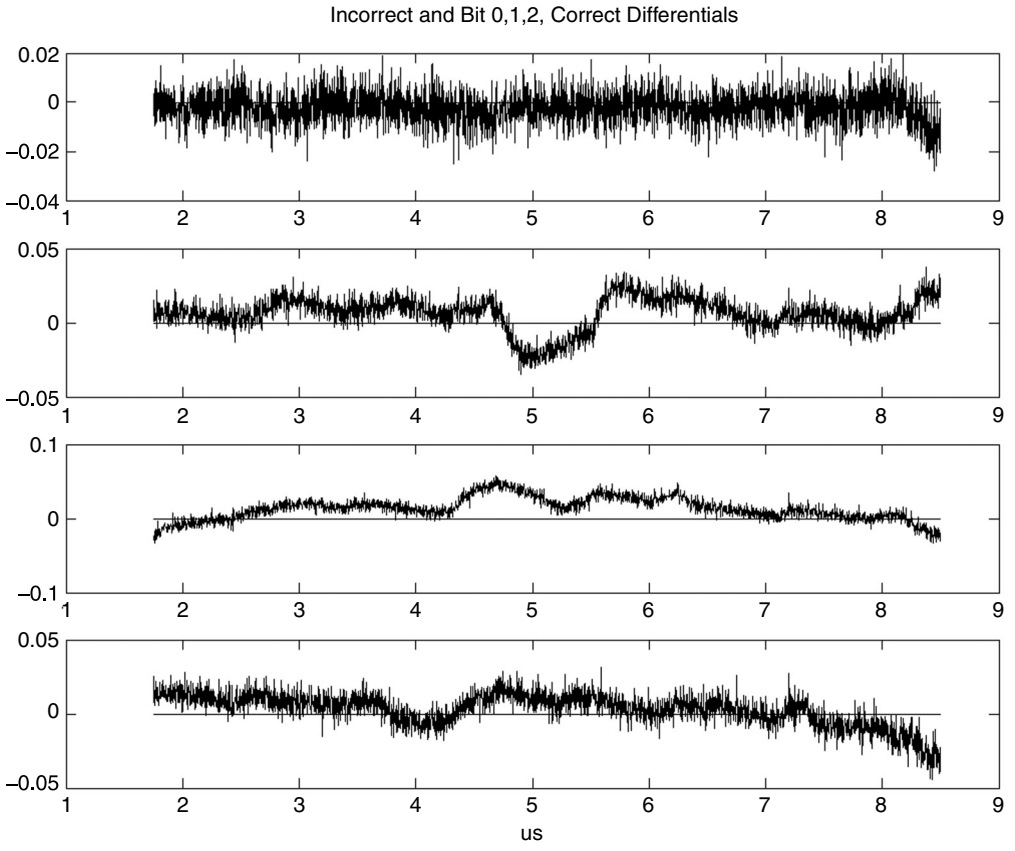


FIGURE 17.9 Incorrect (top) and correct differential traces for bits 0,1,2 for parallel program.

of a parallel program. In this DPA experiment, one group of power traces (group 0) is averaged and subtracted from another group of averaged power traces (group 1). Each power trace is the instantaneous power dissipated by the processor while it runs a specific application. The application involves several memory loads (of 6-bit data) as well as ALU and logical instructions executing in parallel. Group 0 represents the power traces obtained from this program when the memory loads different 6-bit data

words, where the LSB (least significant bit) of the data word is always zero. The group 1 traces are acquired with the same application code; however, the six-bit data word has a LSB of one. In theory, the difference of means removes the variation of the power due to the algorithm (or program). The averaging also removes the variation in power due to the higher 5 bits of the data word (assuming sufficient power traces have been obtained to average out the noise and all combinations of 2^5 data words are exercised to average out the power variation due to the upper 5 bits). The difference of means then theoretically represents the power variation solely due to the LSB, which differs from group 0 to group 1. The theory assumes that the hamming weight of the data being placed on the bus has an influence on the instantaneous power. The top differential power signal in Figure 17.9 illustrates an incorrect guess of the bits (where half of the bit i's of the data word are 0 and the other half of bit i's are 1 in each group). The lower 3 differential traces lie off of the zero axis indicating that the power was influenced by bit 0, bit 1, and bit 2 of the data word respectively. The bottom two traces are less clear and indicate it may be more difficult to obtain a DPA on all bits of the bus. This approach has been used in security applications (e.g., to guess the key bit when the data being loaded is the result of the exclusive or operation on the key and the plain text). Here, the user can input a larger number of plain texts and record the power traces for each. The approach has also been used to correctly guess a double or sum in SPA-resistant elliptic curve cryptography (thus also providing key bit information).

The emerging application of high level software models of power, that of security, briefly discussed in this chapter, is crucial for design of many portable devices, such as PDAs and cell phones, that are internet-enabled or support wireless communications. Unlike SmartCard security research, these portable embedded systems must be energy efficient [29] to maintain long battery lifetimes. Thus, the emerging area of low-energy security will be important, and will be built upon the equipment setups for measuring current, power, and energy as well as instruction-level power models developed for many embedded processors.

17.5 Acknowledgment

The author acknowledges the financial support provided by NSERC, CITO, RIM, and Motorola.

References

[1] L. Benini and G. DeMicheli, System-level power optimization: techniques and tools, *International Symposium on Low Power Electronic Design (ISLPED)*, pp. 288–293, 1999.

[2] M. Lee, V. Tiwari, S. Malik, and M.Fujita, Power analysis and minimization techniques for embedded DSP software, *IEEE Trans. on VLSI Design,* pp. 123–135, March 1997.

[3] A. Chandrakasan and R.Brodersen, *Low-Power Digital CMOS Design,* Kluwer Academic Publishers, Dordrecht, 1995.

[4] V. Tiwari, S. Malik, and A. Wolfe, Power analysis of embedded software, *IEEE Trans. on VLSI,* pp. 437–445, Dec. 1994.

[5] G. Qu, N. Kawabe, K. Usami, and M. Potkjonak, Function-level power estimation methodology for microprocessors, *Design Automation Conference (DAC)*, 2000.

[6] J. Russell and M. Jacome, Software power estimation and optimization for high-performance 32-bit embedded processors, *International Conference on Computer Design (ICCD),* 1998.

[7] C. Gebotys and R. Gebotys, Statistically based prediction of power dissipation for complex embedded DSP processors, *Microprocessors and Microsystems,* 23, pp. 135–144, 1999.

[8] C. Gebotys, R. Gebotys, and S. Wiratunga, Power minimization derived from architectural-usage of VLIW processors, *Proc. Design Automation Conf., ACM,* pp. 308–311, June 2000.

[9] C.H. Gebotys and R.J. Gebotys, An empirical comparison of algorithmic, instruction, and architectural power prediction models for high-performance embedded DSP processors, *Proc. IEEE Int. Symp. on Low-Power Electron. Design,* pp. 121–123, August 1998.

[10] R. Joseph and M. Martonosi, Run-time power estimation in high-performance microprocessors, *ISLPED*, pp. 135–140, 2001.
[11] A. Sinha and A. Chandraksan, Energy aware software, *13th Int. Conf. on VLSI Design,* pp. 50–55, 2000.
[12] N. Chang, K. Kim, and H. Lee, Cycle-accurate energy consumption measurement and analysis: case study of ARM7TDMI, *ISLPED*, pp. 185–190, 2000.
[13] R. Muresan and C. Gebotys, Dynamic power simulation model for VLIW DSP processor VLSI cores with secure applications, *Proc. 11th IFIP VLSI-SOC*, pp. 67–72, December 2001.
[14] R. Muresan and C. Gebotys, Current consumption dynamics at instruction and program level for a VLIW DSP processor, *Proc. ACM/IEEE 14th Int. Symp. on Syst. Synthesis (ISSS)*, pp. 130–135, October 2001.
[15] R. Muresan, Measurements, macro-modeling, and applications of current dynamics in complex core processors. Ph.D. thesis, Department of Electrical and Computer Engineering, University of Waterloo, 2003.
[16] F. Wolf, J. Kruse, and R. Ernst, Compact trace generation and power measurement in software emulation, *Proc. SPIE,* Vol. 42, 28, 2000.
[17] Y.H. Lu, L. Benini, and G. DeMicheli, Requester-aware power reduction, *Int. Symp. on System-Level Synthesis*, pp. 18–23, 2000.
[18] T. Simunic, L. Benini, and G. DeMicheli, Source code optimization and profiling of energy consumption in embedded systems, *Int. Symp. on System-Level Synthesis,* pp. 193–198, 2000.
[19] W. El-Essawy, D. Albonesi, and B. Sinharoy, A microarchitectural-level step-power analysis tool, *ISLPED*, pp. 263–266, 2002.
[20] *SPSS User's Guide, Base 8.0 for Windows,* SPSS Inc., 1998.
[21] P. Kocher, Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems, *Lecture Notes in Computer Science (LNCS)*, 1998.
[22] P. Kocher, J. Jaffe, and B. Jun, Differential power analysis, *CRYPTO '99,* pp. 388–397, 1999.
[23] C. Gebotys and R. Gebotys, Designing VLSI cores with secure applications, *Proc. Cryptographic Hardware and Embedded Syst.,* Redwood City, CA, LNCS 2523, pp. 114–128, August 2002.
[24] IEEE Std. 1363-2000, *IEEE Standard Specifications for Public-Key Cryptography,* IEEE Computer Society Press, Washington, DC, 2000.
[25] C. Gebotys and R. Gebotys, A framework for security on NoC technologies, *IEEE Int. Symp. on VLSI,* February 2003.
[26] T. Messerges, E. Dabbish, and R. Sloan, Investigations of power analysis attacks on SmartCards, *USENIX Workshop on SmartCard Technol.,* 1999.
[27] C. Gebotys, Design of secure cryptography against the threat of power-attacks in DSP embedded processors, *ACM Trans. on Embedded Comput. Syst.,* pp. 92–113, February 2004.
[28] S. Nikolaidis, N. Kavvadias, P. Neofotistos, K. Kosmatopoulos, T. Laopoulos, and L. Bisdounis, Instrumentation set-up for instruction level power modeling, *PATMOS 2002,* LNCS 2451, pp. 71–80, 2002.
[29] C. Gebotys and Y. Zhang, Security wrappers and power analysis for SoC technologies, *Int. Symp. on Syst.-Level Synth.-CODES,* 2003.

18

Low-Power/Energy Compiler Optimizations

Ulrich Kremer
Rutgers University

18.1 Introduction

Embedded processors and systems on chip (SoCs) are used in many devices, ranging from pace makers, sensors, phones, and personal digital assistants (PDAs), to general-purpose, handheld computers and laptops. Each of these devices has their own requirements for performance, power dissipation, and energy usage, and typically implements a particular trade-off among these entities. Allowing components of these devices to be controlled by software has opened up opportunities for compilation and operating strategies to reduce power dissipation and energy usage, at the potential cost of performance degradation. Such control includes:

1. Hibernation (i.e., initiating transitions of a component between high-power active states and lower-power hibernating states)
2. Dynamic frequency and voltage scaling, which allows the clock speed and supply voltage to be set explicitly within a range of feasible voltage and frequency combinations
3. Remote task mapping, where power and energy is saved on a mobile device by executing a task remotely on a server

This chapter discusses general issues and challenges related to compilers for power and energy management. A set of compilation strategies are further examined, together with initial results that describe their potential benefits.

18.2 Why Compilers?

Compilers translate a program in a high-level language into a program that can be executed on a target architecture. In other words, compilers support high-level programming models that allow programmers

to describe the solution to their problem at an abstraction level closer to the particular problem domain. As a result, programs are easier to understand and maintain. Porting a program to another target system requires recompilation on the new system instead of reimplementing the program in the new assembly/machine language; however, these benefits may come at the price of a reduction in overall program performance. Typically, the effectiveness of a compiler and its generated code is measured by comparing it against a code that an "expert" assembly/machine code programmer would have written, or even the best machine code possible. For an optimizing compiler, this difference should not be too large, where the acceptable performance gap depends on the particular application domain. What such a comparison does not capture is the effort needed by an "expert" programmer to come up with such a high-quality code. Modern embedded processors have many features previously found only in high-performance processors, including SIMD instructions, VLIW design, and multiple independent memory banks.

The effort to write efficient or even correct programs may be prohibitively high, particularly for embedded systems with short time-to-market cycles. As a result, high-level languages and their optimizing compilers are becoming a necessary alternative to programming advanced embedded processors in machine and assembly code. Instead of rewriting a set of applications for a new target system, a new compiler has to be provided for that new architecture. Researchers in the embedded systems compiler community have developed and are further investigating new compilation infrastructures that allow the effective retargeting of compilers [9]. Although the issue of retargetability is very important, it is not covered in this chapter.

Optimizing compilers perform program analyses and transformations at different levels of program abstraction, ranging from source code and intermediate code, such as three-address code, to assembly and machine code. Analyses and transformations can have different scopes. They can be performed within a single basic block (local), across basic blocks but within a procedure (global), or across procedure boundaries (interprocedural). Traditionally, optimizing compilers try to reduce overall program execution time or resource usage such as memory. The actual compilation process can be done before program execution (static compilation) or during program execution (dynamic compilation). This large design space is the main challenge for compiler writers. Many trade-offs have to be considered to justify the development and implementation of a particular optimization pass or strategy; however, every compiler optimization needs to address the following three issues:

1. Opportunity. When can the optimization be applied?
2. Safety. Does the optimization preserve program semantics?
3. Profitability. When applied, how much performance improvement can be expected?

Clearly, every program transformation should be safe. Compiler writers would be out of their jobs if safety is ignored. Profitability has to consider any overheads introduced by an optimization, particularly runtime overheads. The combination of opportunity and profitability allows the assessment of the expected overall effectiveness of an optimization.

In principle, hardware- and operating system (OS)-based program improvement strategies face the same challenges as compiler optimizations; however, the trade-off decisions are different based on the acceptable cost of an optimization and the availability of information about dynamic program behavior. Hardware and OS techniques are performed at runtime where more accurate knowledge about control flow and program values may be available. Opportunity, safety, and profitability checks result in execution time overheads, and therefore need to be rather inexpensive. Profitability analyses typically use a limited window of past program behavior to predict future behavior. In contrast, in a static compiler, most of the opportunity, safety, and profitability checks are done at compiler time (i.e., not at program execution time), allowing more aggressive program transformations in terms of affected scope and required analyses. Because the entire program is available to the compiler, future program behavior may be predicted more accurately in the cases where static analysis techniques are effective. Purely static compilers do not perform well in cases where program behavior depends on dynamic values that cannot be determined or approximated at compile time. In many cases, however, the necessary dynamic information can be derived at compile time or code optimization alternatives are limited, allowing the appropriate alternative

to be selected at runtime based on compiler-generated tests. The ability of the compiler to reshape program behavior through aggressive whole-program analyses and transformations, which is a key advantage over hardware and OS techniques, exposes optimization opportunities that were not available before. In addition, aggressive whole-program analyses allow optimizations with high runtime overheads that typically require a larger scope to assess their profitability.

The following sections discuss several promising compiler optimization techniques, together with an assessment of their potential benefits. These optimizations include remote task mapping, resource hibernation, and dynamic voltage and frequency scaling.

18.3 Power vs. Energy vs. Performance

Optimizing compilers need underlying performance models and metrics to be able to transform the program code for a specific optimization goal. These models and metrics guide the compiler to make selections among program transformation alternatives. If one optimization goal subsumes another, there is no need to develop separate models and metrics for the subsumed models. This section addresses the question of whether or not power, energy, and performance should be considered separate compiler optimization goals.

18.3.1 Power vs. Energy

Optimizing for minimal power dissipation or minimal energy usage may have different metrics, and therefore result in different optimization strategies. One possible metric for power and energy is that of activity level at any given point during program execution and total amount of activities for a program region, respectively. The more "work" is done at a program point, the more power is dissipated. Given these metrics, is optimizing for power the same as optimizing for energy? The answer depends on the particular definition of "work."

An optimizing compiler may define work as the number of instructions executed at a given point in time. This model assumes that:

1. A fixed amount of power is associated with each executed instruction.
2. The power dissipation of an instruction is independent of its particular operand values or other executing instructions. Figure 18.1 illustrates this case. By reordering or rescheduling instructions, for instance, in a VLIW or superscalar architecture, the initial power profile of a program region as presented on the left of Figure 18.1 may ideally be transformed into the one presented on the right. Although the peak power dissipation is different for both profiles, the energy usage is the same. In other words, activity or work rescheduling can be an effective way to reduce peak power dissipation while having no impact on energy usage. Therefore, peak power reduction may be an optimization objective different from energy reduction.

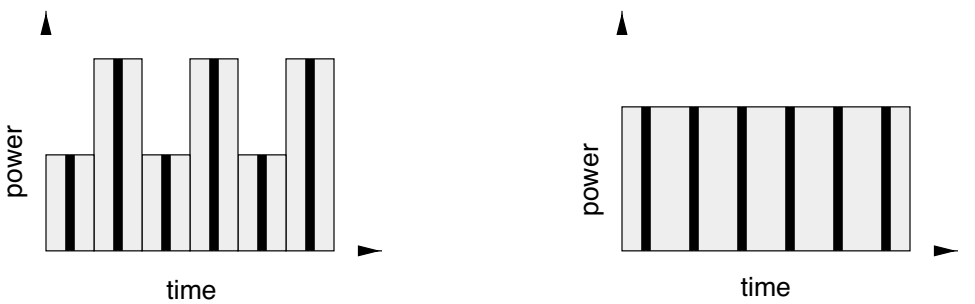


FIGURE 18.1 Optimizing for power vs. energy: two possible power profiles of an example program region.

For power models based on bit-level switching activities as its work notion, rescheduling instructions may also target overall energy usage by grouping instructions based on their particular bit patterns. In addition to instruction scheduling, a careful selection of register names in the code generation phase of a compiler can result in code sequences that have bit patterns with less switching activities, for instance, due to the reuse of "similar" register names [7].

Due to the particular chemical characteristics of some batteries, highly varying discharge rates (i.e., varying power dissipations) may reduce the lifetime of a battery significantly. By "smoothing" the power dissipation profile of an application through instruction scheduling and reordering, the usable energy of a battery can be significantly increased [11].

From now on, we will not distinguish between the optimization objectives of reducing peak power dissipation and overall energy usage unless explicitly stated.

18.3.2 Power/Energy vs. Performance

Early work on optimizing compilers for power and energy management suggested that optimization transformations for performance subsume those for power and energy management. Therefore, power/energy is not an optimization objective in its own right [13]. Traditional optimizations, such as common subexpression elimination, partial redundancy elimination, strength reduction, or dead code elimination increase the performance of a program by reducing the work to be done during program execution [2,12]. Clearly, reducing the workload may also result in power/energy savings. Memory hierarchy optimizations, such as loop tiling and register allocation, try to keep data closer to the processor because such data can be accessed more quickly. Keeping a value in an on-chip cache instead of an off-chip memory, or in a register instead of the cache, also saves power/energy due to reduced switching activities and switching capacitance.

However, a fundamental difference exists between the models and metrics used for performance and those used for power/energy optimizations. Many performance models have the notion of a critical path (i.e., a sequence of instructions or activities that will dominate the overall program execution time). If an optimization introduces activities on the noncritical path, performance is not affected. Therefore, as long as these noncritical activities lead to an overall decrease of the critical path (at least in most cases), the optimization is beneficial. In the context of power/energy optimizations, this is not true. Any activity, whether on or off the critical path, will contribute to the overall power dissipation and energy usage.

Figure 18.2 is an example that illustrates the differences in optimizing for power/energy versus optimizing for performance for a source-level transformation, in this case loop invariant code motion [2,12]. In the example program, the assignment a = b * 2 is assumed to be loop invariant. For a traditional scalar architecture, loop invariant code motion will move the assignment out of the loop, resulting in the code on the right side of Figure 18.2. In a VLIW architecture, the code on the left may be best if empty VLIW instruction slots are available to execute the loop invariant assignment for each iteration of the loop. Although the assignment is done 10 times, it may reduce the overall critical path. Depending on the particular overall compilation strategy used, moving the assignment out of the loop may actually increase the critical path. In the context of power/energy optimization, performing redundant computations should be avoided, and, therefore, moving the invariant assignment out of the loop typically leads to power and energy savings.

Another example where optimizations for power/energy may be different from that for performance is speculative execution. Speculation performs activities "ahead of time" based on some assumptions

```
for (i= 0; i< 10; i++)  {
    a =  b    2;
    c[i] =  d[i] +  2.0;
}
```

```
a =  b    2;
for (i= 0; i< 10; i++)  {
    c[i] =  d[i] +  2.0;
}
```

FIGURE 18.2 Example code fragment to illustrate power vs. performance optimization strategies.

about the future behavior of the program. If these assumptions turn out to be false, additional work may be necessary to undo the impact of the speculative performed activities. Software prefetching is an example of such a transformation. The compiler may insert prefetch instructions for memory accesses across control branches. Assuming that the target machine allows multiple outstanding loads, this optimization can be very effective. Again, as long as the speculative activity can be hidden on the noncritical execution path, no negative impact on performance will occur. In the context of power/energy optimizations every additional, speculative activity has to be compensated for by the overall power/energy benefit of the optimization to make things not worse. In other words, the window of profitability has to be larger for power/energy optimizations than performance optimizations. This does not mean that speculation cannot be applied for power/energy optimizations, but suggests a less aggressive application of such a transformation by restricting it to the cases where the benefit is likely.

18.3.3 Summary

In recent years, reducing the power dissipation and energy consumption of a program have actually become optimization goals, no longer considered byproducts of traditional performance optimizations that mainly try to reduce program execution times. Power and energy optimizations can be implemented in hardware through circuit design, by the operating system through scheduling techniques that consider the power and energy requirements of active processes, and by the compiler through compile-time analyses, code reshaping, and hints to the operating system. The following issues should be considered during the design of an optimizing compiler for power/energy management:

1. You can run but you cannot hide. All instructions, including instructions on the noncritical path contribute to the overall power dissipation and energy consumption. As a result, power/energy optimizations have a higher threshold for profitability than performance optimization if they require additional instructions to be executed.
2. Keep the overall picture in mind. A power/energy optimization with a slight performance penalty may be profitable for a single system component (e.g., cache, CPU, and memory), it may not be profitable for the overall system due to its impact on the power/energy requirements of other system components. In addition, the power/energy characteristics of other active processes have to be considered in a multi-programming environment.
3. You cannot beat hardware. If an operation is implemented in hardware, and an application can take advantage of this hardware (e.g., floating point unit), a compiler should try to generate code for it. If the hardware dissipates power while idle, the compiler needs to be able to disable it during such idle periods.

18.4 List of Optimizations

The following section discusses three compiler optimizations. These optimizations are just examples, and are presented to illustrate the potential benefits of compile time power/energy management. This list is by no means complete.

18.4.1 Dynamic Voltage and Frequency Scaling

Dynamic voltage scaling (DVS) is recognized as one of the most effective power reduction techniques. It exploits the fact that a major portion of power of CMOS circuitry scales quadratically with the supply voltage [3]. As a result, lowering the supply voltage can significantly reduce power dissipation. For noninteractive applications, such as movie playing, decompression, and encryption, fast processors reduce device idle times, which, in turn, reduce the opportunities for power savings through hibernation strategies. In contrast, DVS techniques are still beneficial in such cases (i.e., DVS reduces power even when these devices are active); however, DVS comes at the cost of performance degradation. An effective DVS algorithm is one that intelligently determines when to adjust the current frequency-voltage setting

(scaling points) and to which frequency-voltage setting (scaling factors), so that considerable savings in energy can be achieved while the required performance is still delivered.

One possible compiler-directed algorithm identifies program regions where the CPU can be slowed down with negligible performance loss [6]. It is implemented as a source-to-source level transformation using the SUIF2 [1] compiler infrastructure. Physical measurements on a laptop with a 600–1200-MHz AMD Athlon 4 processor demonstrate that total system energy savings of up to 23% can be achieved with performance degradation of less than 5% for the SPECfp95 benchmarks. On average, the energy and energy-delay products are reduced by 11% and 9%, respectively, at the cost of the performance slowdown of 2%. It was also discovered that the energy usage of the programs using this DVS algorithm is within 6% from the theoretical lower bound.

18.4.2 Resource Hibernation

A common approach to increase energy efficiency puts idle resources or entire devices in low-power (hibernation) states until they have to be accessed again. The transition to a lower power state usually occurs after a period of inactivity (an inactivity threshold), and the transition back to active state usually occurs on demand. Unfortunately, the transitions to and from the low-power state can consume significant time and energy. Nevertheless, this strategy works well when there is enough idle time to justify incurring such costs.

Source-level transformations can be used to reshape the program behavior such that inactivity thresholds of a device or component are extended, allow hibernation to be more effective. By allowing the compiler to give hints to the operating system about expected idle times of these components and devices, the OS is able to issue deactivation directives earlier and activation directives just in time before the device or component is used again. In addition, the operating system can use these hints to implement the most efficient policy for the set of active processes. The results reported in Heath et al. [5] demonstrate that on a set of streamed and nonstreamed application, the reshaped programs can achieve disk energy reductions ranging from 55% to 89% (70% on average) under a sophisticated energy management policy with only a small performance degradation.

18.4.3 Remote Task Mapping

Mobile devices come in many flavors, including laptop computers, Webphones, pocket computers, PDAs, and intelligent sensors. Many such devices already have wireless communication capabilities, and we expect most future systems to have such capabilities. Two main differences exists between mobile and desk-top computing systems, namely the source of the power supply and the amount of available resources. Mobile systems operate entirely on battery power most or all the time. The resources available on a mobile system can be expected to be at least one order of magnitude less than those of a "wall-powered" desk-top system with similar technology. This fact is mostly due to space, weight, and power limitations placed on mobile platforms. Such resources include the amount and speed of the processor, memory, secondary storage, and I/O. With the development of new and even more power-hungry technology, we expect this gap to widen even more. Remote task mapping is a technique that tries to off-load computation to a remote server, thereby saving power and energy on the mobile devices [8,10].

A possible compilation strategy that generates two versions of the initial application, one to be executed on the mobile device (client), and the other on a machine connected to the mobile device via a wireless network (server) [8]. The client and server codes have to be able to deal with disconnection events. The proposed compilation strategy uses checkpointing techniques to allow the client to monitor program progress on the server, and to request checkpoint data to reduce the performance penalty in case of a possible server and/or network failure.

The reported results have been obtained by actual power measurements of an image processing application (face detection and face recognition) on three client systems:

1. The StrongARM-based, low-power SKIFF system developed at Compaq's Cambridge Research Laboratory.
2. Compaq's commercially available StrongARM-based iPAQ H3600.
3. A Pentium-II-based laptop. Initial experiments demonstrate that energy consumption can be reduced significantly, in some cases, up to one order of magnitude, depending on the selected characteristics of the mobile device, remote host, and wireless network.

18.5 Future Compiler Research for Power/Energy

Compiler research for power and energy management is still in its infancy. Such research requires platforms that expose power and energy management features to higher software levels such as the compiler through standardized interfaces (APIs). Although efforts have been made in some areas (e.g., ACPI [4]), more work needs to be done.

In addition, the lack of a reliable and effective evaluation infrastructures for power and energy optimizations has significantly hampered compiler research. The compiler community relies mostly on physical measurements on existing target systems for a set of representative benchmarks to evaluate the benefits of a given optimization or set of optimizations. Simulation results are accepted as an indication of a potential benefit of an optimization, but are typically not considered sufficient proof that the optimization is worthwhile in practice. What is needed is an evaluation infrastructure for power and energy optimizations that consists of a combination of physical measurements and performance modeling. Physical measurements need to include current and voltage measurements, as well as temperature measurements. Performance models are needed for the CPU, memory subsystems, controllers, communication modules, and I/O devices such as the disk and screen. This technology is crucial to be able to understand and assess the benefits of a proposed optimization for the entire target system, subsets of system components, or single system components.

18.6 Acknowledgment

This work has been partially supported by National Science Foundation (NSF) CAREER Award No. 9985050. Any opinions and conclusions expressed in this chapter are those of the author, and do not necessarily reflect the view of the NSF.

References

[1] National Compiler Infrastructure (NCI) project. Overview available online at http://www-suif.stanford.edu/suif/NCI, Co-funded by NSF/DARPA, 1998.

[2] A.V. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools, 2nd ed.* Addison-Wesley, Reading, MA, 1986.

[3] T. Burd and R. Brodersen. Energy-efficient CMOS microprocessor design. *28th Hawaii Int. Conf. on System Sciences (HICSS-95),* pp. 288–297, January 1995.

[4] Advanced Configuration and Power Interface Specification. Compaq, Intel, Microsoft, Phoenix Technologies, Toshiba, Revision 2.06, October 11, 2002. http://www.ocpi.info.

[5] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application transformations for energy and performance-aware device management. *Int. Conf. on Parallel Architectures and Compilation Tech. (PACT '02),* Charlottesville, VA, pp. 121–130, September 2002.

[6] C.-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. *ACM SIGPLAN Conf. on Programming Languages, Design, and Implementation (PLDI '03),* San Diego, CA, pp. 38–48, June 2003.

[7] M. Kandemir, N. Vijaykrishnan, M.J. Irwin, W. Ye, and I. Demirkiran. Register relabeling: a post-compilation technique for energy reduction. *Workshop on Compilers and Operating Syst. for Low Power (COLP '00),* Philadelphia, PA, October 2000.

[8] U. Kremer, J. Hicks, and J. Rehg. A compilation framework for power and energy management on mobile computers. *Int. Workshop on Languages and Compilers for Parallel Computing (LCPC '01),* Cumberland, KY, pp. 115–131, August 2001.

[9] R. Leupers. Compiler design issues for embedded processors. *IEEE Design Test of Comput.,* 19(4):51–58, July/August 2002.

[10] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. *Int. Conf. on Compilers, Architectures, and Synthesis for Embedded Systems (CASES 2001)*, Atlanta, GA, pp. 238–246, November 2001.

[11] T. Martin and D. Siewiorek. The impact of battery capacity and memory bandwidth on CPU speed-setting: a case study. *Int. Symp. on Low-Power Electron. and Design (ISLPED),* San Diego, CA, pp. 200–205, August 1999.

[12] S.S. Muchnick. *Advanced Compiler Design Implementation.* Morgan Kaufmann Publishers, San Franscisco, CA, 1997.

[13] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. Instruction-level power analysis and optimization of software. *J. VLSI Signal Process.,* 13(2/3):1–18, 1996.

19

Design of Low-Power Processor Cores Using a Retargetable Tool Flow

Gert Goossens
Target Compilers Technologies

Peter Dytrych
Dirk Lanneer
Philips Digital Systems Laboratories

19.1 Introduction

With process geometries shrinking to nanometers, unprecedented levels of silicon integration are now available. This has fuelled the design of complete electronic systems on a single multimillion-transistor chip. To master the design complexity of such systems on chip (SoC), the reuse of processor cores has become an important design paradigm. Different types of predesigned and preverified processor cores can be instantiated and connected as building blocks in a heterogeneous chip architecture; however, power consumption is becoming a major hurdle in the successful design of future SoCs.

This chapter describes a methodology for designing low-power processor cores in SoCs. Its key component is the ability to quickly and adequately customize the instruction-set architecture of the processor core, to match the characteristics of the application. It is demonstrated that this allows for a drastic reduction of the power consumption of the processor, while retaining sufficient design flexibility as offered by a programmable processor. The methodology is supported by a retargetable tool-suite, offering architectural exploration, software development, and verification capabilities. The practical applicability of the methodology and tool-suite is demonstrated by the design of an industrial ultra-low power digital signal processor (DSP) core for audio coding applications, named CoolFlux DSP.

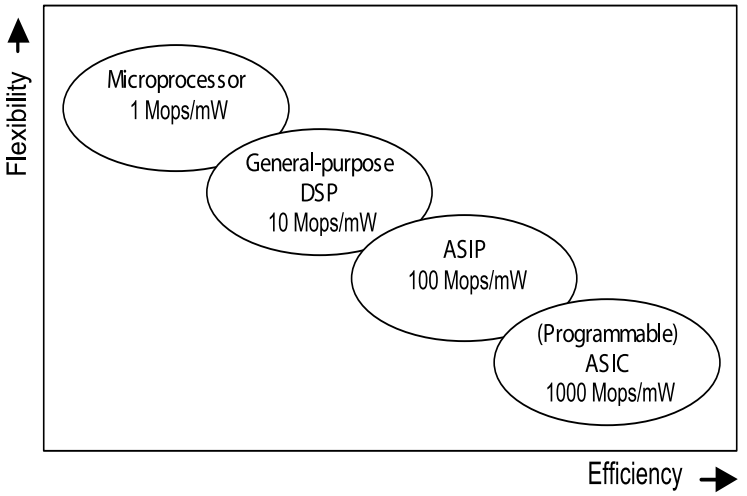


FIGURE 19.1 Classification of processor cores in SoC design.

19.1.1 Processor Cores in SoC Design

Figure 19.1 depicts a classification of processor cores used in SoCs. On the one hand, general-purpose microprocessor and DSP cores, as available from intellectual property (IP) vendors, offer most flexibility. On the other hand, application-specific cores are being designed to implement system functions that are critical in terms of computational throughput and power dissipation. Due to their application-specific nature, specialized semiconductor or system companies often design these cores in-house. Traditionally, these application-specific cores take the form of a fixed-function application-specific integrated circuit (ASIC) block, designed in a hardware description language such as Verilog or VHDL. Such an ASIC core consists of a data-path with special-purpose functional units and interconnections. Control flow is typically restricted, and can be implemented in a small finite-state machine (FSM).

In competitive markets such as telecom and consumer electronics, the flexibility to take into account rapidly changing functional requirements, and the efficiency to cope with high computational throughput and low-power dissipation requirements, are both important. New processor cores must combine the best of both worlds. DSP cores are becoming more application-specific, by extending a general-purpose instruction-set architecture (ISA) with specialized functional units and instructions. This is referred to as application-specific instruction-set processor (ASIP) cores [10]. At the same time, new-generation ASIC cores offer a small layer of software programmability on top of a specialized data-path, to allow for limited functional changes that are crucial to extend the lifetime of these cores. This is referred to as programmable ASIC cores.

Usually, the design of an embedded processor core is significantly influenced by the overall SoC architecture. SoC design addresses the system partitioning, which determines the balance of low power and parallelism, including task parallelism, data parallelism, and instruction-level parallelism (ILP). This defines the broad specification points of the different embedded processor cores in the SoC, which each typically only provide a solution for part of the computational load of the complete system. In this chapter, we assume that the system partitioning and analysis have been performed upfront, although in practice the procedure is rarely this cleanly decoupled and the optimization of a processor core architecture is likely to be done in a complete system context.

As an example, two rather different application scenarios could be a digital hearing instrument and an SoC platform for portable consumer audio. The hearing instrument will have very exacting power and area constraints, but can usually be rather application-specific and have a small application code base (hundreds of assembly lines), for which an ASIP or programmable ASIC (Figure 19.1) is most adequate. The audio platform will tolerate less demanding power constraints and have a very broad code

base (hundreds of thousands of assembly lines), requiring a more general-purpose, 24-bit DSP. The audio platform may also be configured as a multiprocessor to allow scalability.

19.1.2 SoC Integration and Low-Power Design

The requirement of low-power dissipation is becoming an important, if not *the* most important, motivation for making application-specific processors. Whereas from an area and gate-count perspective, putting together tens or even hundreds of processor cores on a single chip is not a problem anymore, controlling the heat dissipation and the energy consumption of such a chip becomes a major issue.

To control the power characteristics of the SoC, the overall system architecture, the architectures of the composing processor cores, and the circuit-level implementation are all important [11]. This chapter primarily focuses on the processor core architectural level. The main idea that is explored is that by making the processor architecture application-specific (i.e., by designing an ASIP or a programmable ASIC instead of a general-purpose processor) [Figure 19.1], its power consumption can be reduced drastically.

In the authors' opinion, the following observations are cornerstones of low-power architecture design:

1. Optimizing for minimum cycle count is beneficial for power consumption. The main part of the dynamic power consumption of a circuit is due to the capacitance effect, and is proportional to the average switching frequency (i.e., clock frequency times an activity factor), and to the square of the supply voltage [5,9]:

$$P = C \times (f_{Clock} \times Act) \times V_{dd}^2$$

   Processor architectures that can implement a given software program in a small number of instruction cycles, will generally exhibit better power characteristics. Indeed, a lower cycle count will allow for scaling down f_{Clock}, and especially V_{dd} (known as voltage scaling).

   A low cycle count can be achieved by introducing specialized functional units to accelerate the critical functions of the algorithm, and by providing instruction-level parallelism. These are key features of application-specific processor architectures.
2. Optimizing for reduced memory access is beneficial for power consumption. A substantial portion of the power consumption in a processor is due to memory accesses. This is both related to the switching activity on data and address busses, and to the loading of word lines in the memories [16]. Processor architectures that can implement a given software program using a small number of data and program memory accesses, will generally exhibit better power characteristics.

   Important power savings can be achieved by providing a storage hierarchy. For example, by providing a loop cache, one can avoid excessive program memory fetches for programs containing loop structures. To reduce the required number of data memory accesses, the architecture should be allowed to maintain variables as much as possible in registers local to functional units. Program memory accesses can be reduced among others by designing an application-specific instruction-set with only a small number of instruction bits, as well as by using techniques such as variable-length encoding and instruction compaction. Again, these are key features of application-specific processor architectures.
3. Minimalistic architectures are beneficial for power consumption. An effective architectural design strategy for low power must be minimalistic [6]. By including only those hardware resources that are really needed by the target applications, power consumption can be reduced significantly. Once again, this leads to application-specific processor architectures.
4. Low-power architectural design is holistic. An effective architectural design strategy for low power must be holistic [6]. To effectively reduce the switching activity and capacitance, all aspects of a processor architecture are important, and only the combination of all elements will result in an overall power-efficient architecture.

19.1.3 Architectural Tool Support for Low-Power Processor Design

Instead of attempting to develop an automatic optimization tool for low-power architecture design, an interactive and iterative methodology is proposed that allows architecture designers to explore different architectural trade-offs and obtain rapid feedback about the quality of architectural decisions.

This methodology, which takes into account the holistic nature of low-power architecture design (see Section 19.1.2), is based on a retargetable tool-suite for processor design available from Target Compiler Technologies called CHESS/CHECKERS [2]. Key features of this technology are the following:

- Whereas other architectural design environments are based on a predefined but parameterizable template of a processor architecture [1,3,4,13], one of the key objectives when developing the CHESS/CHECKERS tool-suite was to provide maximum architectural freedom to the designer. In this way, the designer can find an optimal balance between architectural flexibility and specialization, to obtain the best power dissipation characteristics for his or her application.
- THE CHESS/CHECKERS architectural exploration capabilities effectively allow for the discovery of the architectural sweet spots that result in low-power dissipation.

The retargetable tool-suite must be coupled to a complete power-aware very large scale integration (VLSI) design flow. This allows us to simulate instead of to speculate about power consumption. Good architectural candidates can be determined in the retargetable tool flow by using Chess/Checkers retargetable C compiler and getting profiling data from the retargetable instruction-set simulator (ISS). These can then be pushed through the VLSI design flow to ensure that a good, low-power implementation can actually be achieved. Our experience has been that this process is very revealing and really helps to build efficient processor architectures, taking into account that low-power architecture design is holistic.

The Chess/Checkers tool-suite has been applied successfully to design power-efficient, application-specific processor cores for critical applications in wireless and wireline telecommunications, consumer electronics, and medical devices such as hearing aids. The Chess/Checkers tool-suite, and its abilities for low-power architectural exploration, is described in Section 19.2.

Section 19.3 of this chapter surveys a number of important architectural optimizations that make part of a holistic strategy for low-power architectural design. These optimizations are typically explored in the architectural design phase, using Chess/Checkers.

As an illustration of the methodology, the industrial design of an ultra-low power DSP core for audio coding applications is described in Section 19.4. This processor, called CoolFlux DSP, has been designed by Philips Digital Systems Laboratories [6], with the help of the Chess/Checkers tool-suite.

19.2 A Retargetable Tool-Flow for Designing Power-Efficient, Application-Specific Processors

19.2.1 The Chess/Checkers Retargetable Tool-Suite

Chess/Checkers is a retargetable tool-suite that supports the different phases of designing application-specific processor cores, developing application software for these cores, and verifying the correctness of the design. An outline of the Chess/Checkers tool-suite is listed in Figure 19.2. Chess/Checkers consists of the following tools:

- Chess. A retargetable C compiler that translates C source code into machine code for the target processor. Different from conventional compilers such as GCC [12], the Chess compiler uses graph-based modeling and optimization techniques [15], to deliver highly optimized code for specialized architectures exhibiting peculiarities such as complex instruction pipelines, heterogeneous register structures, specialized functional units, and instruction-level parallelism. Chess produces machine code in the Elf object file format, with source-level debug information in the Dwarf 2.0 format.

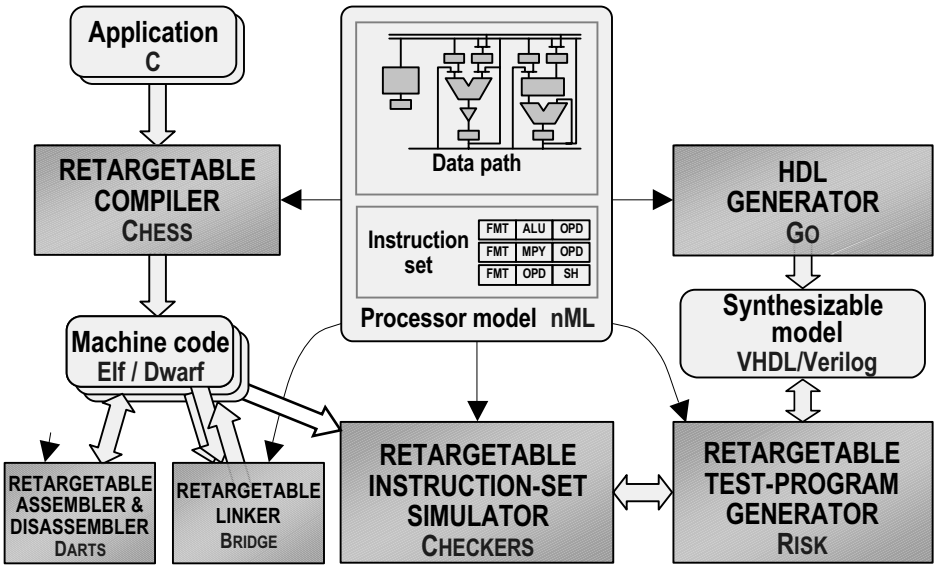


FIGURE 19.2 Outline of the CHESS/CHECKERS tool-suite.

- Bridge. A retargetable linker that builds executable programs from separately compiled Elf/Dwarf object files and libraries.
- Darts. A retargetable assembler and disassembler that translates assembly code into binary Elf/Dwarf object files and back. The assembly language syntax is user-defined.
- Checkers. A retargetable ISS generator that produces a cycle and bit accurate ISS for the target processor. The ISS can be run in a stand-alone mode or be embedded in a co-simulation environment through an application programming interface (API). Checkers comes with a graphical debugger that can connect both to the ISS, as well as to the available processor hardware via a JTAG or debug port for on-chip debugging. Source-level debugging is supported.
- Go. A hardware description language (HDL) generator that produces a synthesizable register-transfer level HDL model of the target processor core. Through APIs, users can plug in their own HDL implementations of functional units and of the memory architecture.
- Risk. A retargetable test-program generator that allows for the quick generation of a large number of assembly-level test-programs for the target processor. These test programs can then be executed both in the ISS and in the HDL model of the processor to check for consistency of both models.

A unique feature of the Chess/Checkers tool-suite is its architectural retargetability, based on the *nML* processor description language. nML is a high-level language that captures a programmer's model of the target processor [7]. This is the abstraction level commonly found in a programmer's manual of a processor. Using nML, an architecture designer can quickly define the ISA of a processor. After reading the nML description, the different Chess/Checkers tools are automatically targeted to the specified architecture.

Figure 19.3 depicts a part of an nML description of a processor. Structural information about the processor is introduced by declaring its storage elements (i.e., memories, registers, and pipeline registers) and its interconnections. The instruction-set is defined using an attributed grammar. The grammar breaks down the instruction set into instruction classes (e.g., `alu_inst`, `mac_inst`, and `shift_inst` in Figure 19.3). The behavior of instructions is specified in action attributes of the grammar rules, using a register-transfer model. In these register-transfer actions, user-defined primitive functions can be called (e.g., `add()`, `sub()`, `and()`, and `or()` in Figure 19.3). To enable instruction-set simulation, the user adds bit-true simulation models for each primitive function. Likewise, to enable hardware generation,

```
// Declaration of storage elements and interconnections:
mem DM[1024]<num,addr>;
reg R[4]<num>;
pipe C<num>;
trn A<num>; trn B<num>;
...

// Definition of instruction set (using attributed grammar):
opn my_core (alu_inst | mac_inst | shift_inst);
...

opn alu_inst (op:opcod, x:c2u, val:c16s, y:c2u) {
  action {
    stage EX1:
      A = R[x];
      B = val;
      switch (op) {
      case add : C = add(A, B) @alu;
      case sub : C = sub(A, B) @alu;
      case and : C = and(A, B) @alu;
      case or : C = or(A, B) @alu;
      }
    stage EX2:
      R[y] = C @alu;
  }
  syntax : op " R" y ", R" x ", " val;
  image : "0"::op::x::y::val;
}
```

FIGURE 19.3 Excerpt of an nML processor description.

the user adds HDL models for each primitive function. Grammar rules also have `syntax` and `image` attributes, defining the assembly language syntax and the binary encoding of the instructions.

19.2.2 Architectural Scope

CHESS/CHECKERS supports a wide range of processor architectures. Retargetability is supported within this range. The following parameters indicate the current architectural scope of the Chess/Checkers tools:

- Data types. Chess/Checkers can support the built-in data-types of the ANSI C language. In addition, users can also introduce any custom data-type. This is useful for application-specific processors, which often contain a variety of specialized data-types. Chess/Checkers allows for the definition of application-specific data types as *C++* classes. The defined classes can then be used in the nML processor description, to specify the data types of the processor's memories, registers, and interconnections. The same class definitions can also be used in the source program for the Chess compiler.
- Arithmetic functions. Chess/Checkers can cope with standard arithmetic instructions found in general-purpose processors, as required for compiling ANSI C code. Users can, however, also define specialized arithmetic instructions in nML, and specify a mapping from the C source code to these instructions using the concept of intrinsic function calls.
- Memories. Chess/Checkers supports von Neumann and Harvard architectures. The processor may have any number of data memories. Each memory may have one or multiple ports. In case of multiple memories, the user can assign static variables in the C source program to specific memories using a memory qualifier. Several addressing modes are supported for data memories. This includes indexed (or offset), direct, and indirect addressing — optionally with postmodification of address pointers. Special addressing operations, such as modulo and bit-reversed addressing, are supported through intrinsic function calls.

- Instruction format. Chess/Checkers supports a wide range of instruction formats, from orthogonal to highly encoded formats. An orthogonal format consists of fixed control fields that can be set independently from each other. In an encoded format, the interpretation of the instruction bits as control fields is dependent on the instruction. Very long instruction word (VLIW) processors have an orthogonal instruction format. The tools support variable-length instructions, as well as instruction compaction to encode small sequences of instructions in a single instruction word.
- Registers. Chess/Checkers supports a wide variety of register structures, ranging from a homogeneous structure with a single, general-purpose register-file to a heterogeneous structure with special-purpose registers that are dedicated to store operands and results of specific instructions. Chess/Checkers also supports various constraints on the utilization of registers. For example, one may specify that the selection of multiple operand or result registers of an instruction be controlled by a single selection-field in the instruction word. Such register coupling constraints often occur in application-specific processors to save opcode space.
- Instruction pipeline: Chess/Checkers supports instruction pipelines of any depth. Different instructions do not need to have the same number of pipeline stages. Chess/Checkers also supports multi-cycle instructions, multi-word instructions, and instructions with delay slots. The Chess compiler can ensure that pipeline hazards, which are specified in nML, are resolved in the generated code.
- Control flow. Chess/Checkers provides support for subroutines and interrupt service routines. Several mechanisms are available to support the concept of a software stack for storage of automatic variables. Chess/Checkers also supports the concepts of hardware do-loop instructions and of mode bits that determine the behavior of instructions.

19.2.3 Architectural Exploration

As explained in the previous section, Chess/Checkers supports a wide architectural design space. Through architectural exploration, a designer can quickly determine a power-efficient architecture for a specific application domain. As a starting point for exploration, the designer will collect the following inputs:

- Application code for the critical functions that need to run on the processor. A large range of possibilities are available, depending on the nature of the design. At one extreme, only a small number of quite similar algorithms may need to run on an application-specific processor. At the other "general-purpose" extreme, one has to consider a large number of potentially highly diverse applications and attempt to optimize from this "sea of C" to ultimately some fully laid out VLSI design and the corresponding object code for an application. For complete DSP applications, such as an MP3 decoder, the code is usually characterized with a 20/80 rule where 80% of the cycles are spent in 20% of the code. This gives a good spread between unstructured control code and DSP loop kernels.
- Architectural design constraints in terms of area, timing, and power budgets, as well as time scales and other project-related factors. Other more difficult constraints may be present as well, such as backward compatibility to an existing processor architecture with a large legacy code base, scalability to allow coverage of multiple price/performance points, and how application-specific or general-purpose to make the processor architecture. These overall specifications will limit the scope of some architectural choices, such as the number of functional units and the choice of initial ISA.

Based on the preceding inputs, the designer typically makes an initial proposal of an ISA, which is described in nML. Once this starting point is chosen, a more refined architectural exploration can be performed, which will lead to the final design. Note that the initial architecture can be a subset or superset of the finished design, although it is probably more common to start with a subset and add features

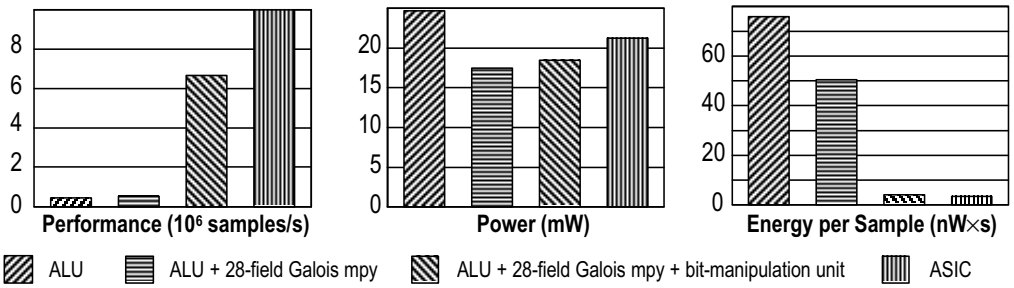


FIGURE 19.4 Performance, power, and energy per sample measurements for different processor architectures for a Reed–Solomon encoding function in an ADSL modem chip.

as required. Section 19.3 discusses a number of important architectural choices for a low-power processor design.

When using Chess/Checkers, a designer can afford to make many iterations. Of each intermediate architecture, the performance can be evaluated by compiling critical C functions with the Chess compiler, and simulating and profiling the resulting machine code with the Checkers ISS. In the first place, the ISS's profiling capabilities allow to evaluate the cycle and instruction count for the application; however; it is also possible to introduce high-level power models in the ISS and to make a comparative power analysis of different architectures (see Section 19.2.4). Using this feedback from the tools, the designer can compare different ISAs and optimize the architecture in nML to obtain a good match between flexibility, throughput, and power characteristics. At some intermediate points, the designer may want to generate synthesizable HDL using the Go HDL generator and enter the VLSI design flow for more accurate measurements.

Figure 19.4 illustrates the architectural exploration capabilities of the Chess/Checkers tool-suite, during the design of an application-specific processor core for Reed-Solomon encoding, for use in an asymmetric digital subscriber line (ADSL) modem SoC. The Reed–Solomon encoding algorithm was described in C source code. As a starting point, a simple microprocessor architecture was used, with a single 32-bit arithmetic and logic unit (ALU). The C code was compiled on the architecture and profiled using the Chess/Checkers tool-suite. The different diagrams depict the computational performance, the power consumption, and the energy that is needed to process one data sample. After profiling the machine code for the single-ALU architecture, it was clear that too many cycles and program memory accesses were spent in the calculation of critical functions such as Galois-field multiplications and the bit-manipulation operations in the Reed-Solomon algorithm. These functions were initially implemented in software on the ALU. In a second design iteration, the designer extended the architecture with a dedicated functional unit capable of computing Galois-field multiplications in a single cycle. This resulted in a moderate increase of the computational performance and an important reduction of the power consumption. In a third iteration, the designer additionally allocated a dedicated functional unit for bit manipulation. This allowed the offloading of the ALU significantly, resulting in a major performance improvement and, likewise, a reduction of the energy needed per data sample.

For comparison, Figure 19.4 also illustrates the characteristics of a hardwired ASIC core for Reed-Solomon encoding, developed in the same process technology. As can be observed, the ASIC core's characteristics are close to the third alternative designed with the Chess/Checkers tools. This comparison illustrates that the Chess/Checkers tool-suite can span a wide range of processor architectures, from general-purpose microprocessors to programmable ASICs. The designer can perform a true architectural exploration and get rapid feedback about the quality of the intermediate results.

19.2.4 Power-Conscious Architectural Design

As explained previously, Chess/Checkers supports an interactive methodology for architecture design. This approach is based on the assumption that automatically generated architectures can never approach

the specialization of a human designer. Instead of automating the architecture generation phase within a restricted architectural scope, Chess/Checkers relies on the designer's creativity while supporting a wide scope of processor architectures. This section elaborates on how the Chess/Checkers tool-suite can be used to design power-efficient processor architectures.

When defining an architecture, the designer makes the basic decisions that influence the power efficiency of the architecture. Optimization for power is supported by the tools in the following ways:

- The Chess compiler primarily aims at optimizing the cycle count of the program, with instruction count or code size as the secondary optimization goal. As explained in the introduction, this generally contributes to low-power consumption. With a low cycle count, it is easier to fit a low V_{dd} and f_{Clock}, while a low instruction count reduces the power dissipated in program memory accesses.
- Note that the length of a clock cycle is not known *a priori,* when modeling an architecture in nML. Typically, the designer can make an estimate, but this needs to be verified by running the HDL generator Go and performing logic synthesis on the generated description.
- The architectural scope of the Chess/Checkers tools and of the nML language is wide enough so that the designer can experiment with different architectural techniques for low power. These techniques are described in Section 19.3. In particular, the cycle and instruction count can be reduced by exploiting instruction-level parallelism, by bundling multiple functions in a single instruction, by exploiting special-purpose registers, and by designing highly encoded instruction sets. The Chess compiler contains various optimization phases to make efficient use of these features.
- The tools can give early feedback about cycle count and instruction count. This is mainly obtained through the profiling capability of the ISS. In addition, the designer can check the effective utilization of functional units and registers, and strip those that are not frequently used.
- It is possible to have the ISS automatically calculate an approximate power consumption figure when executing a program. Based on the nML processor model, the Checkers tool generates an ISS in the form of a C++ source program. The generated model is open enough so that the user can integrate instruction-level power models in the ISS.
- Obviously, such power models are library and technology dependent. To construct and tune these models, a basic architecture can be specified in nML and small programs can be run that repeatedly execute specific instructions or instruction sequences, both in the ISS and in the derived HDL model using a tool such as the Synopsys Power Analyzer. The following experimental observations may serve as guidelines when constructing power models for application-specific DSPs and programmable ASICs:
  - In case of an orthogonal instruction format (see Section 19.2.2 on the architectural scope of Chess/Checkers), power models may be constructed per orthogonal subclass of the instruction word. By adding the power consumption of the orthogonal subclasses, a sufficiently accurate figure for the overall power consumption is obtained.
  - Within an instruction class (e.g., `alu_inst` in Figure 19.3), the specific choice of opcodes (e.g., `add`, `sub`, `and`, and `or` in Figure 19.3) has a dominant effect on the relative power consumed by the instruction. In contrast, the choice of operands or results (e.g., `R[0]` vs. `R[1]` as the source or destination register, and the exact bit-pattern of the immediate constant `val` in Figure 19.3) is much less relevant. Therefore, the choice of opcodes is an important parameter in a power model, while the choice of operands or results may be neglected more easily.
  - Power models are best defined for small sequences of instructions, instead of for individual instructions [8]. Experiments have demonstrated that power calculations in the ISS based on power models for pairs of instructions can be within 30% of the actual power of the circuit (as obtained in a gate-level simulations). In case only power models for individual instructions are used, however, the results can differ as much as 80%. To reduce the complexity of the model, the order of the sequence may be neglected (i.e., one may assume that the power consumed by the sequence A|B is the same as for B|A).

- The HDL generator Go contains a number of optimizations that contribute to a power-efficient hardware implementation of the processor core. For example, Go can generate write-enable signals for selected registers, which allows commercially available logic synthesis tools to introduce clock gating to reduce power dissipation. In addition, Go is able to latch the inputs of unused functional units, to prevent toggling of unused logic and thus save power.

19.3 Low-Power Processor Architecture Design

This section addresses some common issues covering the design of low-power processors and introduces how such a design may proceed in practice. The general area of low-power processor architecture design is potentially a very broad subject, and we will limit the discussion to that of embedded, low-power, DSP design and specifically focus on the processor core itself. It should be pointed out that the retargetable tool-suite presented in Section 19.2 is not limited to this domain, and this choice is driven by our actual design experience with a relatively general-purpose audio processor called CoolFlux DSP, which is described in Section 19.4.

19.3.1 General Characteristics

When considering different low-power processor core architectures, it is worth having some sort of power-aware metric with which to compare them. We have used simple metrics that consider some key aspects of a processor core. An example of such a figure of merit for a specific application could be:

$$cost = (m_{app}/m_{max}) \times P \times A$$

where m_{app} is the minimum clock frequency required for the application to achieve real-time operation, m_{max} is the maximum clock frequency of which the processor core is capable, P is the power per MHz, and A is the complete area, including memory. For this particular metric minimizing the cost would be a goal. It is worth noting that this formula contains conflicting factors, so that, for example, increasing parallelism and thus A is likely to decrease m_{app} due to the ability to exploit ILP. Thus, in many respects, an optimized processor core architecture has to find some good compromises among conflicting requirements. This section covers some key aspects regarding the broad architectural choices that have to be made at an early stage.

- Parallelism. Here, we have to consider the type and amount of parallelism a single processor core node will support. A low-power processor core design should try to approach the ideal power/parallelism characteristic as illustrated in Figure 19.5. This is based on minimizing control overhead. The two main types of parallelism exploited here are ILP and data parallelism. ILP is related to the number of operations an instruction can issue to functional units and the pipeline depth of the various functional units. These two factors will define the number of operations in flight at any one time. Given the need for efficient compiler support and the ILP potential of the applications, parallelism needs to also be well balanced in the architectural exploration. We have obtained efficient compilation results for a machine that issued up to eight basic operations per instruction and had four pipeline stages. For many DSP applications data parallelism, as supported in a single-instruction multiple-data (SIMD) machine, is a particularly efficient approach to use as the processor core control overhead is further amortized over several operations on sub-words. Thus, a 32-bit base architecture could also define instructions that perform four 8-bit operations in parallel. This is easily supported by the retargetable tool-suite by the definition of vector data types and operations.
- Pipeline structure. This is a key aspect of low-power processor core design, and many factors need to balance well here. We spent a lot of architectural exploration time in this area when designing our audio DSP (see Section 19.4). We have found that relatively short, simple pipelines (i.e., three to five stages) with limited interlocks and bypassing have given us good results, with

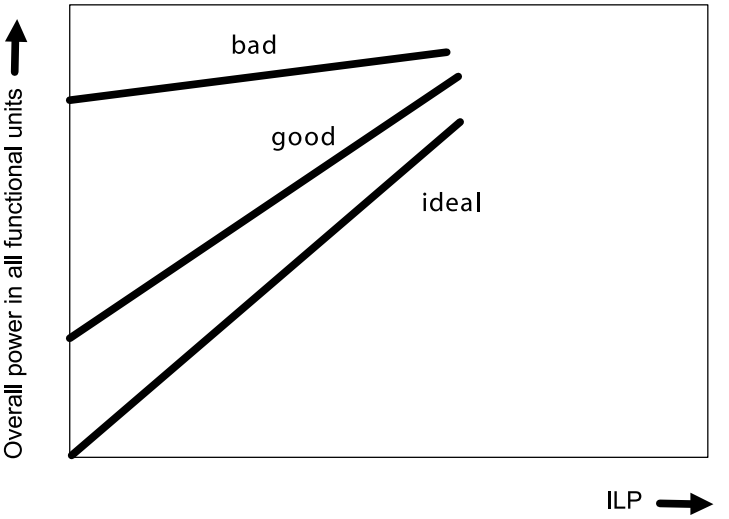


FIGURE 19.5 General relationships between power and ILP used in a processor core (assuming V_{dd} and f_{Clock} are kept constant).

minimal design and verification effort. Instruction issue policy is strictly in order. Generally, pipeline depth has a quickly diminishing improvement on performance. At the same time, cost factors, such as design and verification effort, tend to increase rapidly, as is illustrated in Figure 19.6. As an example of many of the factors concerned, a search for the best pipeline depth would have to consider the following elements: the speedup possible for the system clock (this gives a power advantage from the larger potential voltage scaling with its quadratic power reduction), reduced cycle efficiency due to extra delay slots (or the need for extra bypasses), the potential deglitching effects of pipeline registers, the extra power cost of the clock tree, and the pipeline schedule and length of control transfers. A final comment on pipeline structure is that the CHESS compiler is good at resolving static pipeline scheduling issues. Thus, the onus is on providing a rather exposed pipeline and allowing a minimal hardware solution that is good for low power.

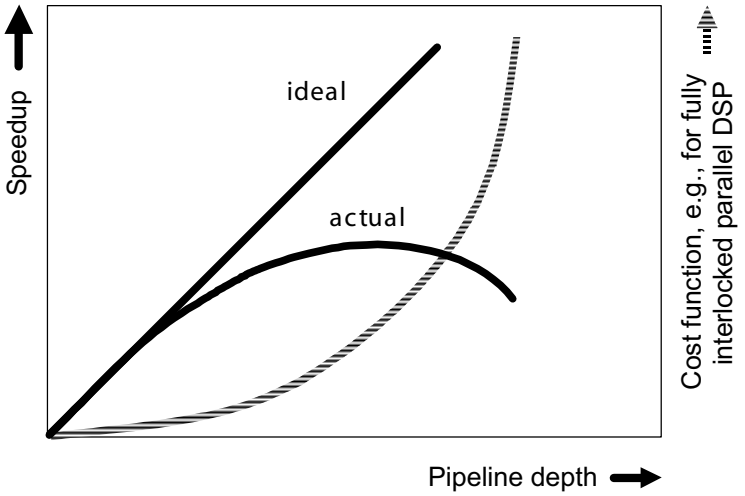


FIGURE 19.6 Speedup and cost as a function of pipeline depth.

- Register file structure. The register file of a processor core can be homogeneous (a central structure) or may be fully heterogeneous (distributed) at the two extremes. For low-power DSP applications, a distributed register file is desirable because this reduces the number of register ports and the sizes of the various files, as well as leveraging locality of storage. The costs associated with a single central register file serving many functional units over many (probably bypassed) pipeline stages are usually prohibitive for a low-power machine. With distributed register files, particularly within the main datapath, however, comes the problem of efficient register allocation and scheduling. Our experience here is that Chess is particularly well positioned to solve this problem and has allowed us to use this low-power distributed register file feature in our designs.
- Memory architecture. Memory access can now be over 50% of the power budget of an embedded DSP, and is likely to rise as geometry shrinks and applications grow. Several issues must be addressed here including aspects of memory hierarchy and separate memory data spaces. The second issue is a standard feature of most DSP designs and will not be elaborated here. Most low-power machines should exploit a data memory hierarchy of some sort, which may span main storage, local tight coupling between memory and register files [16]. Here, intelligent prefetch techniques can have significant advantages over traditional caches, particularly for applications such as video that combine statically known addressing patterns and large data objects.
- For program memory, a small cache, or at least a loop buffer, will reduce overall program memory power consumption for the 20/80 code encountered in DSP applications, as will any techniques used to reduce program memory size. Compact instruction encoding should be sought to reduce the program memory's size. In addition, we have used compression of the program memory contents coupled with the good code density already produced by the Chess compiler to aggressively reduce the program memory footprints.
- Memory addressing. For DSP applications, a well-established set of extended addressing modes are used, usually operating as a post modify on the relevant pointer register. These greatly contribute to reducing the cycle count required for the application, and thus indirectly to low power. Several trade-offs that balance the number of addressing modes against the complexity of the addressing units are possible here. The likely address modes include facilities for cyclic addressing as well as bit reversed addressing for fast Fourier transforms (FFTs) and other butterfly-based computations. To support a C compiler and to maintain efficient data structures that minimize use of data memory, good support of a software stack is also desirable. Thanks to the software stack concept, both the data memory size and the cycle count can be reduced, which contributes to low power. Stack support can be provided in a number of ways. We have typically used a fully indexed stack with a dedicated stack pointer.
- ISA. DSP designs encompass a wide variety of ISA design styles, from orthogonal to highly encoded. For low-power applications, we have favored highly encoded ISA design styles. This approach minimizes program memory size while providing enough parallel instruction classes for the ILP extracting Chess compiler to operate efficiently. Although we provided symmetry across the various parallel views of the machine, parallel operations were only introduced for the common forms of parallelism in DSP applications, such as multiply-accumulate (MAC)-based inner loop kernels. This has also allowed us to have a rather asymmetric datapath where most of the functionality is in the primary ALU, thus allowing a relatively compact design. Some of these issues are further expanded in the Section 19.3.2.

19.3.2 Instruction-Set Architecture

A low-power processor has a carefully optimized ISA, which attempts to strike a fine balance among code size, encoding, instruction decoder complexity, and compiler efficiency in scheduling ILP. This is quite a difficult balance to achieve due to the requirement to maintain enough parallelism in much of

the ISA to keep compilation efficient, while maintaining a short instruction word and thus small program memory footprint.

Because DSP code is characterized by the 20/80 rule, there is a need to support rather diverse requirements. The ISA of the processor core can be thought of as having several distinct facets, in the form of instruction classes that implement various styles of computation. For example, in our audio DSP design, there is a relatively nonparallel microcontroller like facet as well as one that codes for maximum parallelism in DSP kernels.

Considering the design of the ISA has repercussions throughout the processor core design, and as far as producing a low-power instruction decoder is concerned, it is important to use regular formats where possible to minimize the amount of field extraction multiplexing that is needed. In addition, to maintain a good pipeline timing balance, certain encoding styles can be used that allow the fast production of time critical control signals and a spread of distributed decoding functions across pipeline stages.

Another issue we addressed aggressively is the potential inefficiencies due to flow control instructions, such as branches. As control flow instructions can occur up to about once every five instructions in general compiled C code, it is very important to maintain high cycle efficiency here. A number of techniques can be used that minimize the power consumption of the processor core. We typically provide a good mixture of flow control instructions with and without exposed delay slots. This allows the Chess compiler to make good code selection choices based on whether delay slots can be scheduled efficiently. We also provide zero overhead hardware looping to maintain high efficiency within inner loop constructs. Again, the compiler handles this automatically and forms software-based loops when the hardware loop stack is fully utilized. Another feature we use is conditional execution of instructions, which eliminates the use of a branch construct and have no exposed delay slots.

Generally, it is best to try to avoid the explicit coding of no-operation (NOP) instructions. This is partly aided by the options that have been provided on flow control instructions, but we have also added a form of NOP compression to some of our designs, which reduces our program memory size by up to 25% for typical compiled applications in our audio DSP. These savings have a significant impact on power and area: we measure an average 25.3 bits/instruction for typical compiled code for our CoolFlux DSP, while the instruction width is 32 bits.

19.3.3 Micro-Architecture

A low-power micro-architecture should attempt to minimize control overhead while keeping the main datapath as efficient as possible, within the bounds of the technology used. This means that pipeline interlocks and bypass networks should be used only when necessary. It will also be useful to run candidate designs right through placement and routing to ensure that cell row utilization during chip layout can be maintained as this will reduce area and thus the capacitance and power associated with many nets while improving timing.

From a clocking perspective, a single edge clocked synchronous design can achieve a better timing balance and clock tree efficiency than designs using both clock edges. We have always tended to carefully limit the number of overall registers in a design in order to keep control of the clock tree size and its significant power consumption (up to 40% of processor core power for semi-custom VLSI design flows). A low-power design will use the standard techniques of micro-clock gating and operand isolation that are now available with many synthesis tools. The Go HDL generator in the Chess/Checkers tool-suite is capable of selectively enabling these techniques in the generated HDL design. These are standard techniques and will not be further discussed here.

The pipeline should already be designed so that good timing balance is achievable and when implementing the micro-architecture this goal must be furthered through the VLSI design flow. Usually, for critical sections, this will mean attention at RTL source, synthesis, and back end of the flow. For the memory subsystems, this is particularly important because many critical timing paths are likely to be present here. A common solution to this problem is to use a write-back buffer, which schedules writes

to memory only when free access slots are available. This technique allows a full clock cycle to be allowed for memory access.

The control signals from the instruction pipeline should be held until they are actually needed by a functional unit. From a power perspective it is detrimental to toggle, for example, multiplier input selection lines unless an instruction specifies a valid multiplier operation [14]. If possible, the instruction decoder itself should be designed to minimize internal toggling; this may be achieved by using a distributed design, for example, by instruction class.

Many of the final micro-architectural optimizations are made when the VLSI flow is exercised. This is particularly true for issues such as unnecessarily toggling control logic and optimization of critical timing paths.

19.3.4 Methodology

We have used a design methodology that attempts to provide as much information as possible to the processor core system architect, so that fine design trade-offs can be made using real simulation data. Therefore, we have established a full VLSI design flow as well as having the retargetable compiler tool-suite in place at a very early stage in the project. Thus, RTL design has proceeded in parallel with processor core architectural exploration, and this has allowed very useful insights to be had. These activities have also tended to bond the team together through having access to a common design database.

Most architectural exploration is done within the retargetable tool-flow environment by compiling a suite of applications with Chess and performing profiling with the Checkers ISS. This loop will include changes to the nML processor description, typically things such as ISA, pipeline schedules, and internal computational resources as well as optimization of the application source code are explored here. Good architectural candidates are pushed through synthesis, test insertion, and occasionally full layout to ensure that no problems occur with the realization of the complete processor core. We are particularly interested in maintaining high efficiency through physical design.

This VLSI design flow is power aware, and we use the accurate gate-level power simulator DIESEL, which was developed by Philips. This simulator is driven by actual simulation vectors, and typically reaches accuracy within 10% of final silicon for the technologies in use. We had experimented with RTL power estimators before, but have kept with the gate level simulator because we typically needed early area and timing figures anyway. Once the flow is scripted, it is easy to get some accurate figures in an overnight run. This design flow is also complete in that full layouts are produced, and we use fully extracted (Hyperextract 3D) parasitic data when producing final area, timing, and power figures.

The specification of a design has to be carefully managed. We have tended to take a minimalist approach, where additional features are only added if they demonstrate a significant performance/cost benefit. Another factor in this process has been to lock the specification once confidence has been established; any changes beyond this are handled by a strict change request procedure. A factor we have particularly avoided has been "creeping" specifications.

We have been fortunate in being able to build a small effective team of like-minded engineers run by a single system architect who makes any final design related decisions. Of particular significance has been the synergy between engineers at Philips and Target Compiler Technologies and the close cooperation that was achieved. Perhaps a small, tightly knit team is reflected in a compact power optimal processor core design.

19.4 An Ultra-Low Power DSP for Audio Coding Applications

This section illustrates some of the principles outlined before, by considering a few aspects of the actual design of a low-power C programmable audio DSP, the CoolFlux DSP, developed within Philips PDSL.

19.4.1 Background and Goals

The CoolFlux DSP audio core was designed with two main objectives in mind:

1. The need for very low-power consumption
2. The need to be efficiently programmable with the C high-level language

The extra productivity advantages of programming in C outweighed the minor power increase for efficiently providing C language support in the processor core.

The project was actually also a first introduction into the codesign of the ISA and compiler using a retargetable compiler. In many ways, the question of how low-power a general-purpose high-level language programmable DSP could be made was being addressed (e.g., did we have to include application-specific features (in this case for MP3 coding) or programming restrictions to meet our power goals?).

The motivation behind this is that power consumption continues to be a very important issue in applications. This is due to the increase in portable products on the one hand and their higher processor core needs on the other hand. The portable MP3 players that are appearing in the market are a good example. The processor core throughput requirements for these devices increase rapidly. The application algorithms become increasingly computationally demanding and a larger number of functions need to be executed. At the same time, product lifetimes are decreasing, and this puts increasing pressure on product development cycle times, particularly the software component of these projects.

These factors and a large code base of 24/56-bit fixed-point applications were the starting specification points for the CoolFlux DSP. An MP3 decoder program was used as the main driver application because this gave a good mix of unstructured "control-like" code as well as some tight DSP kernels with high ILP potential.

19.4.2 Architecture

The CoolFlux DSP architecture is depicted in Figure 19.7. It is a dual MAC, dual Harvard machine capable of sustaining two MACs, two memory operations, and two pointer updates per instruction making it highly cycle efficient for computationally intensive DSP applications when scheduled by Chess. Due to the unique ISA design, the CoolFlux DSP is also highly efficient at supporting unstructured "control-type" code as well making it very well balanced for complete embedded applications in which 20/80 code mix is typical.

Finally, much attention has been paid to efficiently supporting the operations needed by ANSI C making the CoolFlux DSP an excellent compiler target while maintaining very low-power consumption through the careful realization of the underlying micro-architecture and the entire design flow.

The datapath consists of an X and a Y data processing side (Figure 19.7). The two sides are highly asymmetric. The main arithmetic components that are available are two multipliers, two full ALUs, and two rounding and saturation units.

The X multiplier is coupled to a preadder, thus the third small ALU. This ALU (ALU0) can perform a nonsaturating addition or a subtraction of two datapath registers. The result of this operation is then multiplied by another datapath register. The X multiplier performs all useful combinations of signed and unsigned multiplication. This allows efficient C type support as well as enabling higher precision arithmetic if needed. The Y multiplier is a lot simpler than the one on the X side. There is no preadder function available and only a signed/signed multiplication is possible.

The X ALU is the main ALU in the processor core. It has full 56-bit precision as well as efficient support for C `long` and `int` types. The operations supported are quite extensive and include DSP specific functions such as absolute and maximum. Division support and full four-quadrant division is also efficiently supported from C. The X ALU also primarily generates the condition code flags. The Y ALU is much simpler and has 56-bit precision. It only supports a subset of the X ALU operations. The main datapath has four distributed register banks: the X, Y registers and the A and B accumulators.

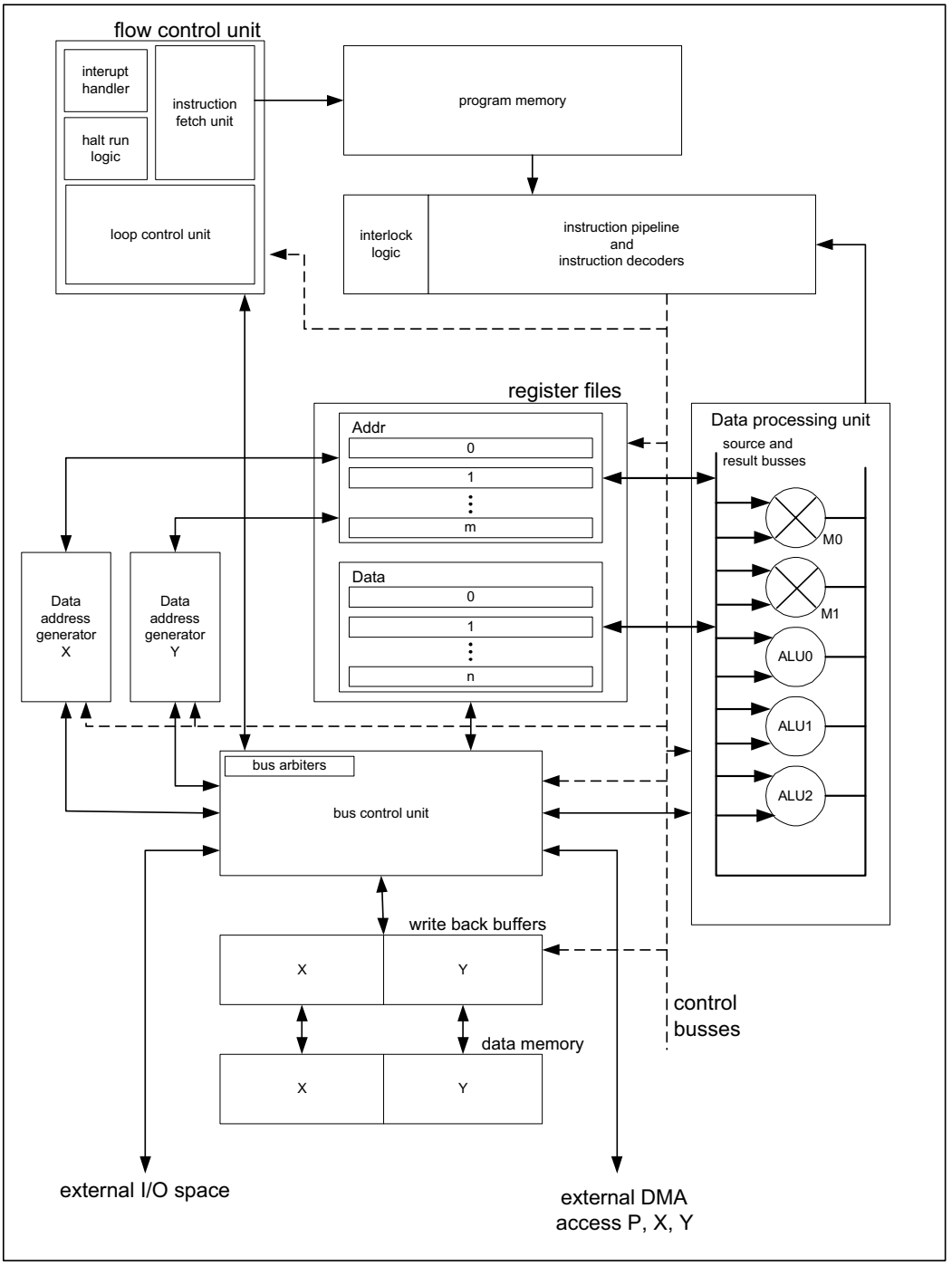


FIGURE 19.7 Block diagram of CoolFlux DSP audio core.

The bus control unit supports moves within the machine. This unit has been designed as a central bus switch used to exchange the contents within and between the X and Y busses as well as supporting move operations on each side. Considerable design effort was spent on this unit to ensure it was power efficient as well as meeting the stringent timing requirements needed.

Besides the main datapath unit, an address generation unit is also used. Two separate units are available for the X and Y memories, respectively. These units contain the address registers, the modulo-protected, and indexing ALUs, as well as the bit reverse addressing logic.

The flow control unit controls the processor core. This fetches the instruction stream, decodes it, and schedules the pipelines by issuing control signals to the entire machine. Interrupts are also processed here, and we have implemented a low latency, vectored interrupt system. This system guarantees interrupt response within a limited number of cycles thus easing the buffering needs of external devices. We support fully interruptible hardware loops, even if they only contain one instruction.

A very important unit is the loop control unit, which provides zero-overhead hardware looping. A maximum of four nested loops is supported, although this can be set as a parameter. This loop control unit is very efficient, both in software overhead and in real hardware parameters such as area and power.

19.4.3 Low-Power Techniques

The processor core uses all of the standard techniques for low-power design, mainly based around micro-clock gating, operand isolation, and general unnecessary toggle reduction techniques. These will not be elaborated here. We also use a technology library, including SRAM, which allows use of aggressive voltage scaling to below 1V.

A pipeline structure was developed that allowed use of a single edge clock while giving memory access a full clock cycle. This will tend to maximize the execution clock rate, thus allowing the most scope for voltage scaling, which gives approximately quadratic power reduction. The pipeline structure is also finely balanced with the rest of the micro-architecture design resulting in minimal and simple interlocks, minimal bypassing, and minimal length control transfers among pipeline segments.

The pipeline also leverages the ISA structure by utilizing a small number of distributed instruction decoders as opposed to one large one, thus reducing unnecessary logic toggling. In addition, some instruction encoding techniques were used to attempt to minimize unneeded logic toggling [17]. Finally, instructions are included that can put the core into various sleep modes, including a deep sleep where the clock can be deactivated.

The processor core uses distributed register files, local to their respective computational resources, to reduce power consumption. The ISA also includes some coupling mechanisms that attempt to reduce the need for copies among register files. Another factor is that the pipeline structure of the processor core was designed so as to minimize the need for bypassing mechanisms. The advantages all add up when compared with using a single multi-ported central register file. For a machine with the parallelism that is available within our processor core, this is a large advantage in both power and maximum clock frequency.

Significant area (cost) and power consumption is now evident in the memory subsystems of processor cores. Several techniques have been used to reduce both the size and power consumption of these memories. Globally, all memory spaces are made up of smaller physical segments such that only one is active in any space at any cycle.

Data memory utilization is optimized by allowing use of efficient data structures that are supported by powerful addressing modes within the processor core. Both cyclic and bit reversed addressing are supported as well as other common DSP addressing modes. Particularly efficient stack support is provided allowing efficient linkage and local storage for functions. These techniques ensure that data memory requirements are minimized as well as execution cycles. On the architectural front, a good balance has been sought between memory subsystem costs and the ability to provide sufficient memory bandwidth for the parallel computational units.

To increase the efficiency of the program memory, innovative techniques for the code size reduction have been included, on top of the very efficient code that is generated by the Chess C compiler anyway (i.e., an efficient method for code compression has been included). A major trade-off in designing the ISA was the width of the instruction word against the amount of encoding used while still leaving enough degrees of freedom for the compiler to perform code selection well. An ISA with key areas that were orthogonal was developed. This ISA was enhanced with a form of NOP compression, which used very little control logic. This has been leveraged by the use of scheduling techniques within the compiler, which favor the generation of instruction sequences, which allow maximal compression to be used. We have measured savings between 20 and 25% in code size for typical applications.

The processor core supports extensive I/O facilities allowing easy, efficient interfacing to other systems. Particularly, interrupts are implemented in a complete and robust way. The interrupt system is characterized by very low latency so that even single instruction hardware loops are fully interruptible. This allows a minimal amount of specific buffering to be implemented, thus keeping system costs low. This flexible I/O system and the ability of the retargetable environment to support intrinsic functions also mean that more application-specific accelerators can be easily added to increase the system power efficiency if needed.

The processor core is implemented in VHDL at RTL level and a power aware semi-custom ASIC VLSI design flow is used. The requirements for low power are carefully considered in all stages of the VLSI design flow as well, including the coding, synthesis, and layout areas.

19.4.4 Results

The CoolFlux DSP design has been initially realized in Philips' 0.18-μ CMOS technology. All the figures given are for simulation data; the timing, area, and power results are using fully extracted parasitic three-dimensional data from the layout database and not wire load estimates.

The power breakdown of the processor core is depicted in Figure 19.8. It demonstrates a low control overhead of approximately 14% in the processor control unit (PCU) logic and power that is dominated by the datapath components (DCU). The overall control overhead with program memory fetches

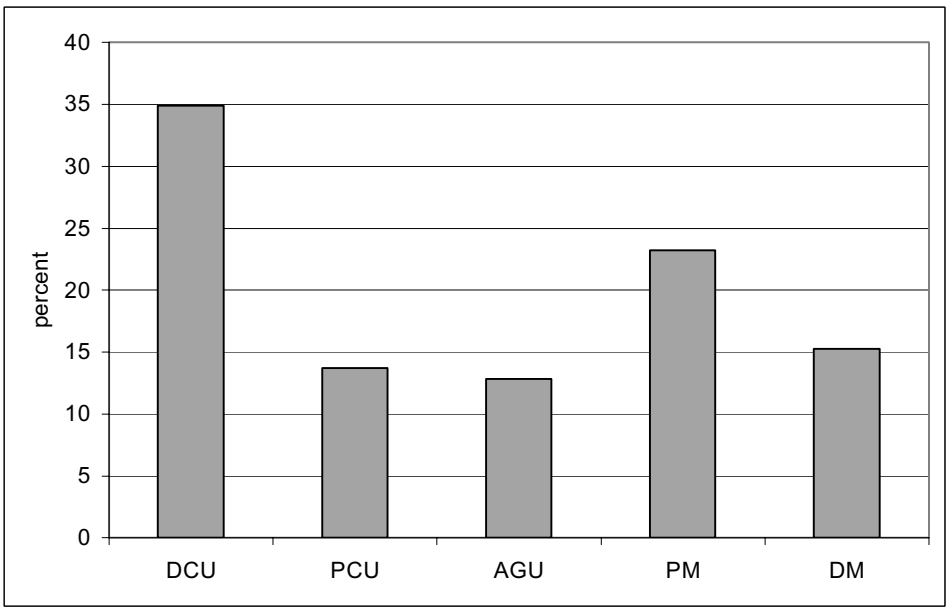


FIGURE 19.8 CoolFlux DSP power breakdown.

TABLE 19.1 CoolFlux DSP Performance Figures for MP3 Decoder

MP3 decode computation load needed	14.9 MIPS (128-Kbps, 44.1-KHz stereo material)
Program memory size	4.6 Kwords × 32 bits
Data memory size	10 Kwords × 24 bits (total)
MP3 core power consumption	1 mW (Philips 0.18 μ standard cell low-leakage CMOS, V_{dd} = 0.9 V, f_{Clock} = 15 MHz)
Maximum clock frequency	135 MHz (worst-case commercial)
Core size	Approximately 45 kGate (NAND2 equivalents)

included is some 37% of the total power consumption. The memories are represented by the program memories and data memories (PM, DM) and the address calculation units are in the AGU module.

Table 19.1 presents some figures obtained for the CoolFlux DSP for an MP3 decoder application and some general core area and timing parameters.

19.5 Conclusions

This chapter described a methodology for the design of low-power programmable processor cores in SoCs. The main idea that is explored is to customize the instruction-set architecture of the core. By making the core application-specific, efficient architectures with minimal overhead, running at lower clock frequencies and exploiting lower supply voltages, can be obtained while retaining much of the flexibility and programmability of a general-purpose processor.

The key infrastructure for making this methodology work in an industrial environment is a retargetable tool-flow that allows for quick and thorough exploration of the architectural design space as well as for efficient software development starting from C source code. In this chapter, the Chess/Checkers tool-suite from Target Compiler Technologies has been introduced for this purpose. In addition, a survey has been given of various architectural techniques that are beneficial for designing low-power processor cores. Most of the presented architectural optimizations are within the architectural solution space of the Chess/Checkers tool-suite.

The effectiveness of the methodology has been demonstrated through the design, by Philips, of an ultra-low power processor core for audio coding applications called CoolFlux DSP. As an example, this core can run MP3 decoding from compiled C code in less than 15 MIPS, which results in a processor core power consumption of about 1 mW in 0.18-μ standard cell technology.

19.6 Acknowledgment

The authors thank their colleagues at Easics N.V. for sharing some of the data on low-power design experiments.

References

[1] ARCtangent-A4 Core — A Technical Summary, ARC International Inc., http://www.arc.com, 2003.
[2] Chess/Checkers: a retargetable tool-suite for embedded processors, Technical white paper, Target Compiler Technologies, http://www.retarget.com, June 2003.
[3] Xtensa Architecture and Performance, Tensilica Inc., http://www.tensilica.com, Sept. 2002.
[4] R. Camposano and J. Wilberg, Embedded system design, *Design Automation for Embedded Syst.,* Vol. 1, No. 1–2, pp. 5–50, Jan. 1996.
[5] A. Chandrakasan and R. Brodersen, *Low-Power Digital CMOS Design,* Kluwer Academic Publishers, Boston, 1995.
[6] P. Dytrych, M. Adé, J. Coninx, J. David, and P. Vandebroek, The design of a very low-power MP3 decoder accelerator, *DSP Valley Annu. Res. and Technol. Symp.,* Leuven, Belgium, Oct. 2002.

[7] A. Fauth, J. Van Praet, and M. Freericks, Describing instructions set processors using nML, *Proc. European Design and Test Conf.,* pp. 503–507, March 1995.
[8] M.T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, Power analysis and minimization techniques for embedded DSP software, *IEEE Trans. on VLSI Syst.,* Vol. 5, No. 1, pp. 123–133, March 1997.
[9] J.M. Rabaey and M. Pedram, *Low-Power Design Methodologies,* Kluwer Academic Publishers, Boston, 1995.
[10] J. Sato, M. Imai, T. Hakata, A. Alomary, and N. Hikichi, An integrated design environment for application specific integrated processor, *Proc. Int. Conf. Comput. Design,* pp. 414–417, Oct. 1991.
[11] D. Singh, J. Rabaey, M. Pedram, F. Catthoor, S. Raigopal, N. Seghal, and T. Mozdzen, Power conscious CAD tools and methodologies: a perspective, *Proc. IEEE, Special Issue on Low-Power Electron.,* Vol. 83, No. 4, 1995.
[12] R.M. Stallman, Gnu compiler collection internals, http://gcc.gnu.org, Dec. 2002.
[13] J. Sato, A. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, and M. Imai, PEAS-I: a hardware/software codesign system for ASIP development, *IEICE Trans. on Fundamentals,* Vol. E77-A, No. 3, March 1994.
[14] C. Su, C. Tsui, and A. Despain, Low-power architecture design and compilation techniques for high-performance processors, *Proc. IEEE COMPCON,* Feb. 1994.
[15] J. Van Praet, D. Lanneer, W. Geurts, and G. Goossens, Processor modelling and code selection for retargetable compilation, *ACM Trans. on Design Automation of Electron. Syst.,* Vol. 6, No. 3, pp. 277–307, July 2001.
[16] F. Catthoor, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design,* Kluwer Academic Publishers, Boston, 1998.
[17] S. Woo, J. Yoon, and J. Kim, Low-power instruction encoding techniques, School of Computer Science and Engineering, Seoul National University, undated.

20

Recent Advances in Low-Power Design and Functional Coverification Automation from the Earliest System-Level Design Stages

Thierry J.-F. Omnès
Philips Semiconductors

Youcef Bouchebaba
University of Nantes

Chidamber Kulkarni
University of California-Berkeley

Fabien Coelho
Ecole des Mines

20.1 Introduction

To meet the cost, power, performance, and programmability constraints of next-generation multimedia devices and platforms in a reasonable design and verification time, introducing a system specification cleaning engine so-called the software washing machine by IMEC's Hugo de Man is key [23]. At this highest level (and first step) in the overall system-level design and verification flow, automation is a very

difficult problem because system architects typically prefer the expressiveness of C/C++ to the powerful semantics of synchronous languages [6] such as Esterel [16], Lustre [32], and Signal [49], using tools such as Esterel Studio [26], Polychrony [42], and Simulink. They also prefer the expressiveness of C/C++ to the powerful mathematics of geometric data-flow modeling available from languages such as Fortran, Alpha [21], and tools such as paralléliseur interprocedural de programmes scientifiques (PIPS) [37] and MMAlpha [47]. Because data access and transfer have the biggest impact on the cost, power, and performance of embedded systems, bridging the gap between geometric data-flow modeling and C/C++ requires special attention. Section 20.2 tours basic geometric transformations to motivate the introduction of a novel and advanced low-power optimization engine, based on PIPS, with the ability to take restricted but still C code as input. Beyond the productivity gain achieved by automation, we observe superior power savings than typically obtained using the systematic but manual code rewriting techniques that stand for best practice in this field today.

Another well-known source of cost, power, and performance optimization is parallelism. Having performed our novel and advanced transformations from the earliest system-level design stage, we move to task- and data-level parallelism exploitation. On an Intel IXP1200 network processing platform, we partition an IPv4 forwarding application written in C into 4 tasks (so-called micro-engines) and 16 threads for exploiting data parallelism using a YAPI (Y-chart) approach. As illustrated in Section 20.3, this results again in important productivity gains while maintaining near-optimal performance on large packet lengths.

Because our advances rely on sophisticated C/C++ source code transformations and manual partitioning, it is essential to verify their functional correctness. Section 20.4 describes the co-verification methodology currently used within Philips' System and Software Design Environment (SSDE). It is the last but essential step to bridge the gap between advanced low-power techniques and production-quality embedded system design and verification.

20.2 Advanced Loop Transformations for Low Power

Research in the program transformation field has drawn much attention for several years. It consists in finding new techniques that allow the compiler to transform source code to optimize some criteria, such as parallelism, execution time, or data locality, which have a direct effect on the reduction of energy consumption [13,38]. The transformations described here aim at improving data locality to switch costly transfers from the main memory to cheaper cache or register memory. During program optimization, the greatest profit comes from the loop nest optimizations because they use the most time in computation of scientific programs. For many years, several techniques have been proposed to transform these nests. Among these techniques, tiling [36], fusion [39,65], and memory reallocation [24,28] can be cited. Tiling is a good technique for increasing the data locality, but most work of this technique is only dedicated to code with single loop nest. This chapter demonstrates how we combine all these techniques to apply them to sequences of nested loops [7–9]. The codes considered here are signal processing applications, which are sequences of loop nests of equal but arbitrary depth. Each of these nests uses a stencil of data produced in the previous nest, and the references to the same array are equal, up to a shift.

20.2.1 Input Code

As mentioned previously, the input code includes signal-processing applications, which are a sequence of loop nests as depicted in Figure 20.1. Note that the dependencies of this code form a directed acyclic graph, and each of these nests uses a stencil of data produced in the previous nest and represented by a set: $V^k = \{\overrightarrow{v_1^k}, \ldots \overrightarrow{v_{mk}^k}\}$.

Domain D_0 associated with array A_0 is defined by the user. To avoid illegal accesses to the various arrays, the domains D_k $(1 \le k \le p)$ are derived in the following way: $D_k = \{\vec{i} \,/\, \forall\ \vec{v} \in V^k : \vec{i} + \vec{v} \in D_{k-1}\}$.

We suppose that the vectors of the various stencils are lexicographically ordered, so that $\forall\ k : \overrightarrow{v_1^k} \le \cdots \le \overrightarrow{v_{mk}^k}$ ($\le$ is a lexicographic operator).

```
do i⃗ ∈ D0
  A1(i⃗) = F1(A0)
enddo
⋮
⋮
do i⃗ ∈ Dk
  Ak(i⃗) = Fk(Ak−1)
enddo
```

FIGURE 20.1 General form of input code.

20.2.2 Loop Fusion

Loop fusion is a transformation technique that combines several loops into one loop. It has several advantages including:

1. Lower cost of loop bound testing [19]
2. Synchronization reduction when loops are distributed among different computation units [12]
3. Data locality increasing [44]

This chapter focuses on this last point, which consists of reducing data transfers among various levels of memory hierarchy, which has a direct effect on energy consumption [13,38].

Generally, the fusion of two loops is valid if and only if they have the same iteration domains and in the merged nest we do not create dependencies from instruction of the second nest to instruction of first nest such as: flow dependence, output dependence or anti-dependence [63]. To merge all nests of code in Figure 20.1, we should make sure that all elements of array A_{k-1} that are necessary for the computation of an element $A_k(\vec{i})$ at iteration $\vec{i}$ in the merged nest have already been computed by previous iterations. To satisfy this condition, we shift [34] the iteration domain of every nest by a delay $\overrightarrow{h_k}$. Our fusion will be valid if and only if these various delays satisfy the following condition [7–9]: $\overrightarrow{h_k} \geq \overrightarrow{h_{k+1}} - MAX_i(\overrightarrow{v_i^{k+1}})$.

The merged code after shifting the various iteration domains is given in Figure 20.2. S_k is the instruction label and $D_{iter} = \bigcup_{k=1}^{p} (D'_k)$, where D'_k is domain D_k shifted by vector $\overrightarrow{h_k}$. As instruction S_k might not be executed at each iteration of domain D_{iter}, we guard it by condition: $C_k(\vec{i}) = if(\vec{i} \in D'_k)$.

```
do i⃗ ∈ Diter
  S1: C1(i⃗)  A1(i⃗ − h⃗1) = F1(A0)
  ⋮
  S1: C1(i⃗)  A1(i⃗ − h⃗1) = F1(Ak−1)
  ⋮
  S0: C0(i⃗)  A0(i⃗ − h⃗0) = F0(A0−1)
enddo
```

FIGURE 20.2 Merged nest.

```
do i⃗ ∈ D_iter
  S_1: C_1(i⃗)  B_1(F_1(i⃗)) = F_1(A_0)
  ⋮
  S_2: C_2(i⃗)  B_2(F_2(i⃗)) = F_2(B_2-1)
  ⋮
  S_k: C_k(i⃗)  A_k(i⃗ − h⃗_k) = F_k(B_k-1)
enddo
```

FIGURE 20.3 Merged nest with buffer allocation.

20.2.3 Fusion with Buffer Allocation

In practice, we only need the first and last arrays (A_o and A_p) because all others arrays only hold temporary data. Thus, we replace arrays $A_1 \ldots A_{p-1}$ by circular buffers $B_1 \ldots B_{p-1}$. Buffer B_i is a one-dimensional array that will contain array A_i live data. This transformation saves memory space and avoids loading the same element several times.

20.2.4 Live Data

An element of array A_{k-1} is said to be live at iteration $\vec{i} \in D'_k$ if and only if:

1. It is produced in the domain D'_{k-1} at iteration $\vec{i}_1 \leq \vec{i}$.
2. It exists in the domain D'_k at iteration $\vec{i}_2 \geq \vec{i}$, which will consume it.

The memory volume $M_k(\vec{i})$ corresponding to an iteration $\vec{i} \in D'_k$ is then given by the number of elements of array A_{k-1} that are live at iteration $\vec{i}$. We can verify easily that $M_k(\vec{i})$ is bounded by a constant, which will be noted Sup_k.

20.2.5 Code Generation

As mentioned it earlier, each array A_k will be replaced by a buffer B_k , which will be managed in a circular and sequential way. The size of buffer B_k is given by Sup_{k+1} defined previously. To store and load the elements of array A_k in the buffer B_k we associate with it an access function: $F_k : D'_k \rightarrow N$ such that:

$$F_k(\vec{O}_k) = 0$$

$$F_k(SUCC(\vec{i}_k)) = (F_k(\vec{i}_k) \bmod Sup_{k+1}$$

The merged code after the replacement of different arrays by buffers will have the form of the code in Figure 20.3.

20.2.6 Tiling

Tiling is one of the most important techniques in the program transformation domain. Generally, it transforms a nest of depth n into another nest of depth $2n$. Much work has been done on tiling [36,62], but most of it is only dedicated to a single loop nest. This chapter presents a simple and effective method that simultaneously applies tiling with fusion to a sequence of loop nest. We are interested only in data that live in the cache memory, thus our tiling is at one level. Our tiling is used as loop transformation and is represented by two matrices: a matrix $A(n, 2n)$ that gives the various coefficients of tiles and a

```
do i
  S1: C1(Ap^-1 l)  A1(Ap^-1 l - h1) = F1(A0)
  ⋮
  S2: C2(Ap^-1 l)  A2(Ap^-1 l - h2) = F2(A2-1)
  ⋮
  Sk: Ck(Ap^-1 l)  Ak(Ap^-1 l - hk) = Fk(Ak-1)
enddo
```

FIGURE 20.4 Tiled code.

permutation matrix $P(2n, 2n)$ that allows for the specification of the organization of tiles and the iterations inside these tiles. As with fusion, the first step before applying tiling with fusion is to shift the iteration domain of every nest by a delay $\overrightarrow{h_k}$. We note by D'_k the shift of domain D_k by vector $\overrightarrow{h_k}$.

20.2.7 Matrix $A(n, 2n)$

Matrix $A(n, 2n)$ defines the tile size and allows us to transform every point $\vec{i} \in Z^n$ into a point $\vec{i}\,' \in Z^{2n}$. In the vector, $\vec{i}\,'$ we have two types of loops:

1. Loops that iterate over tiles
2. Loops that iterate over iterations inside tiles

All the elements of the i^{th} line of this matrix are equal to zero except:

1. $a_{i,2i-1}$, which represents the size of tiles on the i^{th} axis
2. $a_{i,2i}$, which is equal to 1

The relationship between $\vec{i}$ and $\vec{i}\,'$ is given by $\vec{i} = A\vec{i}\,'$.

20.2.8 Matrix $P(2n, 2n)$

Matrix $A(n, 2n)$ has no impact on the execution order of the initial code. Permutation matrix $P(2n, 2n)$ allows for:

1. The placement of all the loops that iterate over tiles before the loops that iterate over iterations inside tiles
2. The specification of the order in which the iterations will be executed

This matrix transforms every point $\vec{i}\,'$ into a point $\vec{l} \in Z^{2n}$ such as $\vec{l} = P\vec{i}\,'$. After the application of matrices $A(n, 2n)$ and $P(2n, 2n)$, we obtain the code in Figure 20.4.

20.2.9 Tiling as a Loop Transformation

Contrary to fusion without tiling, the computation of the various delays $\overrightarrow{h_k}$ in the case of fusion with tiling requires more effort. Our fusion with tiling will be represented as transformation:

$$\omega : Z^n_{\vec{i}} \rightarrow Z^{2n}_{\vec{l}}$$

As mentioned in our previous work [7–9], the simultaneous application of tiling with fusion to the input code is valid if and only if:

$$\forall k, \forall \vec{i} \in D_{k+1}, \forall \vec{v} \in V^{k+1} : \omega(\vec{i} + \vec{v} - \overrightarrow{h_{k+1}} + \overrightarrow{h_k}) \leq \omega(\vec{i}).$$

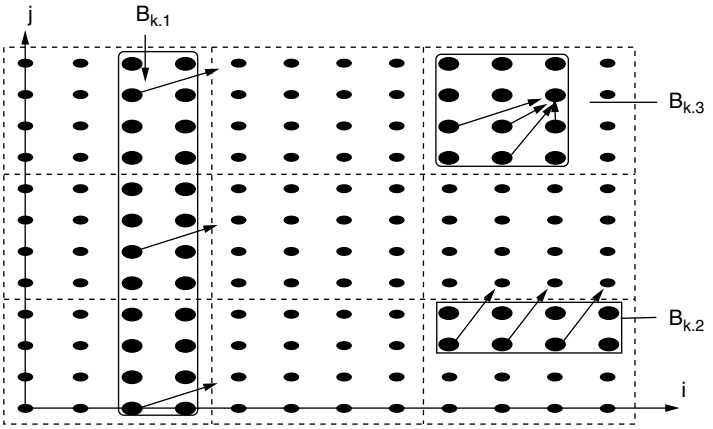


FIGURE 20.5 Multiple buffer allocation.

This condition means that each data-producing iteration must be computed before the iteration that consumes it.

20.2.10 Buffer Allocation

In the case of fusion, we suggest replacing arrays $A_1 \ldots A_{p-1}$ by circular buffers $B_1 \ldots B_{p-1}$. A buffer B_i is a one-dimensional array that contains the live data of array A_i. In the case of fusion with tiling, this technique has two drawbacks:

1. Dead data are stored in these buffers to simplify access functions
2. The size of these buffers increases when the tile becomes large

To eliminate these two problems, we replace every array A_k by $2n+1$ buffers. As mentioned earlier, tiling allows transforming a nest of depth n into another of depth $2n$. The n external loops iterate over tiles, while the n internal loops iterate over iterations inside these tiles. For every external loop m, we associate two buffers $B_{k,m}$ and $B'_{k,m}k$ (corresponds to array A_k); for all internal loops, we define single buffer $B_{k,n+1}$, which contains the live data inside the same tile. For example, if the depth of the nests is two, every array A_k will be replaced by five buffers:

- Buffer $B_{k,1}$ contains the data produced by a column of tiles, which will be consumed by the following column.
- Buffer $B_{k,2}$ contains the data produced in a tile, which will be consumed by the following tile.
- Buffer $B_{k,3}$ contains the live elements in the same tile, and it is managed as the circular buffer for the fusion.

In a given tile, we use data that are produced in the previous tile, and we produce other data that will be consumed in the following tile. To avoid destroying data in the buffer $B_{k,2}$, we duplicate it by another buffer $B_{k,2}$. For this same reason, we duplicate the buffer $B_{k,1}$ by another buffer $B_{k,1}$.

20.2.11 Implementation and Tests

As mentioned in the introduction (Section 20.1), our goal is to reduce the energy consumption in signal processing applications, which strongly depends on data transfers between the various levels of the memory hierarchy. The tests presented here are the numbers of cache misses and execution times of various transformations. We have considered only the external cache because it is the only one that generates data transfers between the processor chip and its environment. The two measurements are carried out for Sun Blade 1000, based on a microprocessor UltraSPARCIII with and 8-MB external cache

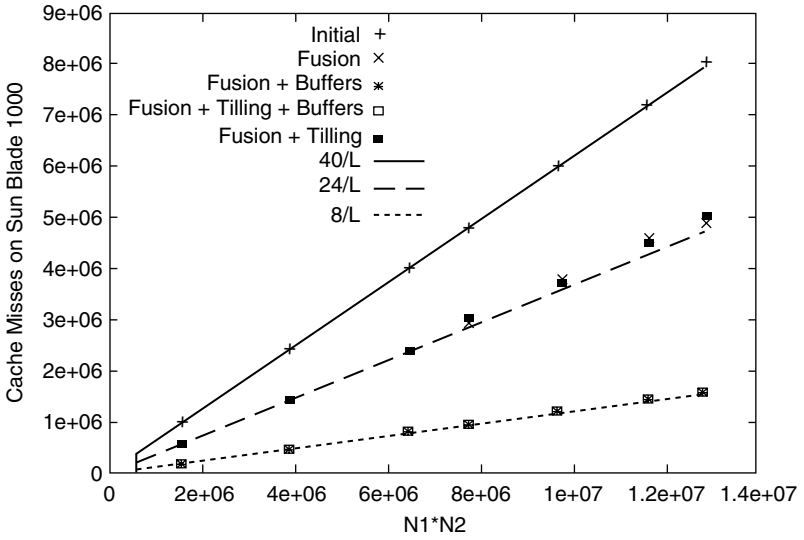


FIGURE 20.6 External cache misses.

and 750 MHZ of clock frequency. The code on which we made our experiments is real application; cavity detection code provided by IMEC, which is a sequence of five nested loops.

20.2.12 Cache Misses

The Figure 20.6 plots cache misses according to array size on Sun Blade 1000. The lines of rate 40/*L*, 24/*L*, and 8/*L* (L is the size of external cache lines) represent theoretical values for cache misses respectively of the initial code, the merged code (tiled and merged code), and the merged code with buffer allocation (merged and tiled code with buffer allocation). As one can observe from Figure 20.6:

- Buffer allocation in fusion and in fusion with tiling decreases considerably the number of external cache misses by almost a factor of 5 when compared with the initial code.
- All tests follow asymptotically well-defined lines that correspond to expected theoretical results.

20.2.13 Execution Time

The objective of our research was to increase data locality. We nevertheless measured the execution times of different transformations of our application. Figure 20.7 gives the execution times according to the array size on Sun-Blade-1000. Notice that the merged code with buffer allocation increases considerably the execution time. This increase is foreseeable because the modulo used in access functions is time-consuming. To improve the execution time of this code we can either use powers of 2 as buffer sizes or eliminate the modulo by unrolling the loops. Figure 20.8 contains the execution times of the various transformations when the buffer sizes are a power of 2.

20.2.14 Conclusion

Section 20.2 discussed the reduction of the energy consumption of signal processing applications executed in embedded systems. The reduction of the energy consumption requires a reduction of memory accesses as demonstrated by IMEC's work [14,30,64]. We have studied several program transformations improving the data locality, and we have extended them.

Much work on loop transformation has been done, but most of it is only dedicated to codes with single loop nests. This section combined loop fusion, tiling, loop shifting, and memory reallocation to apply them to sequence of nested loops. Our method consists in shifting each iteration domain by a delay

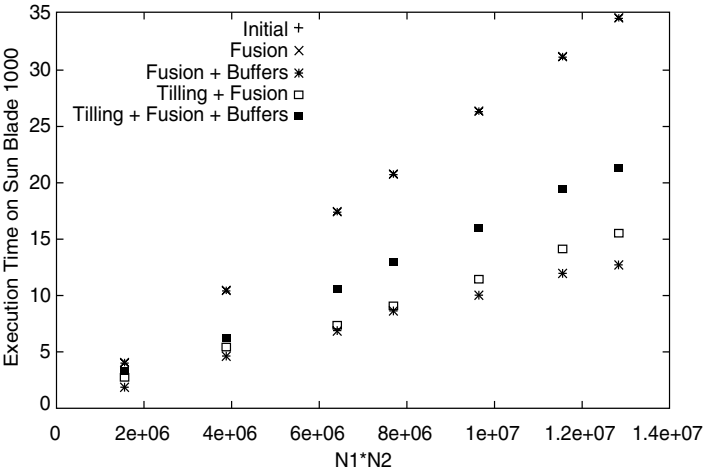


FIGURE 20.7 Execution time of various transformations on Sun-Blade1000.

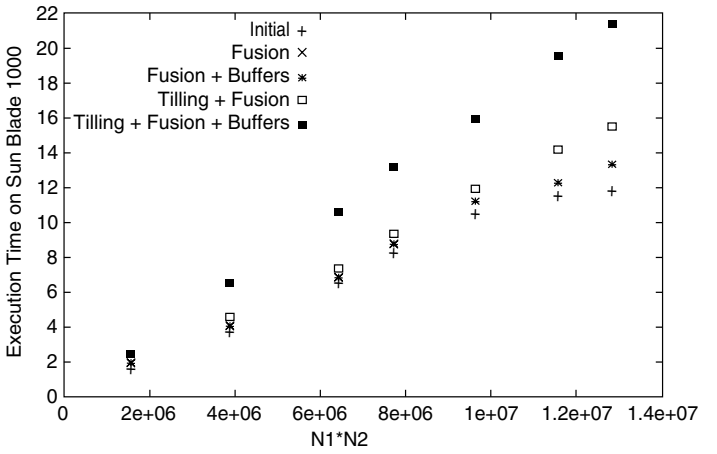


FIGURE 20.8 Reduction of time execution by changing the buffer size (size = power 2).

to ensure the legality of the simultaneous application of fusion with tiling. We then used the concept of live data to replace each array by:

1. Sequential and circular buffers in case of fusion
2. A set of buffers in case of fusion with tiling

All transformations described in this section have been implemented in our source-to-source prototype compiler PIPS, and we have carried out two measurements (cache misses and execution time) on target Sun Blade 1000 machines. These measurements demonstrate that the number of cache misses can be reduced by a factor of up to 5.

20.3 Exploiting Task-Level and Data-Level Parallelism on the Intel IXP1200

Network processors exploit task and packet level parallelism to achieve high throughput. To date, this has resulted in a huge diversity of architectures for similar applications. Driven by practical implementations. This section explores the different trade-offs in network processor design and implementation.

20.3.1 Introduction

Contemporary network processors (NPUs) exhibit a wide range of architectures for performing similar tasks: from simple reduced instruction set computer (RISC) cores with dedicated peripherals, in pipelined or parallel organization, to heterogeneous multiprocessors, based on complex multi-threaded cores with customized instruction sets. Although so diverse, all NPUs exploit task-level concurrency in applications by means of parallel programmable processing elements (PEs) to meet line speed requirements. Thus, the inter-PE communication and the topology of PEs are performance critical aspects of any NPU architecture.

Programming such concurrent systems remains an art. The programmer is not only required to partition and balance the load of the application manually among multiple PEs, it is also necessary to implement each task, often in assembly, before a reliable performance estimation can be obtained. Thus, a robust application mapping strategy for such architectures requires a balance between thread partitioning, scheduling, memory accesses, and input/output (I/O). With current tools, this task becomes time-consuming and error prone, due to trial-and-error methods employed by system implementers based on simulation runs. Therefore, topology, inter-PE communication, and the ease of mapping are likely to be key aspects of the quest for a natural programming model.

For the next generation of network processor based system implementations, we strongly believe that considerable emphasis will be put on performance per cost (e.g., power consumption) aspects and on support of appropriate programming models. Therefore, it is essential to identify and investigate limitations and bottlenecks in system implementation without going all the way down to complete implementations, as is the current practice. Thus, the use of high-level design space exploration and verification tools is required that support a wide range of heterogeneous architectures and enable precise reasoning about different implementation styles and their performance. The main goal of this section is to clearly understand the performance/cost trade-offs for network processor based implementations. In this process, we have implemented two differently mapped versions of our IPv4 benchmark [26] on IXP1200 to gain detailed insight into programmability of existing NPU architectures.

20.3.2 Performance Modeling and Evaluation

Our approach to design space exploration is based on the Y-chart approach. Separate descriptions of the application (workload) and the micro-architecture are bound to each other in an explicit mapping step, describing bindings of tasks and communication onto micro-architecture building blocks. The following evaluation of the system may manually or automatically trigger adaptations of the workload, the allocation of architecture building blocks, or the mapping of the application onto the architecture.

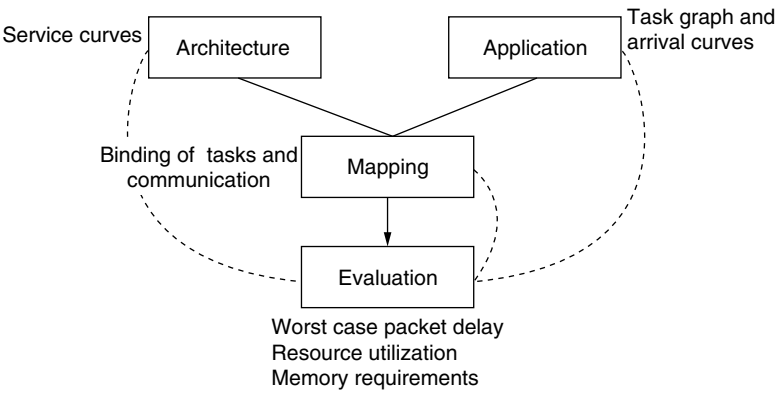


FIGURE 20.9 Design exploration using a Y-chart.

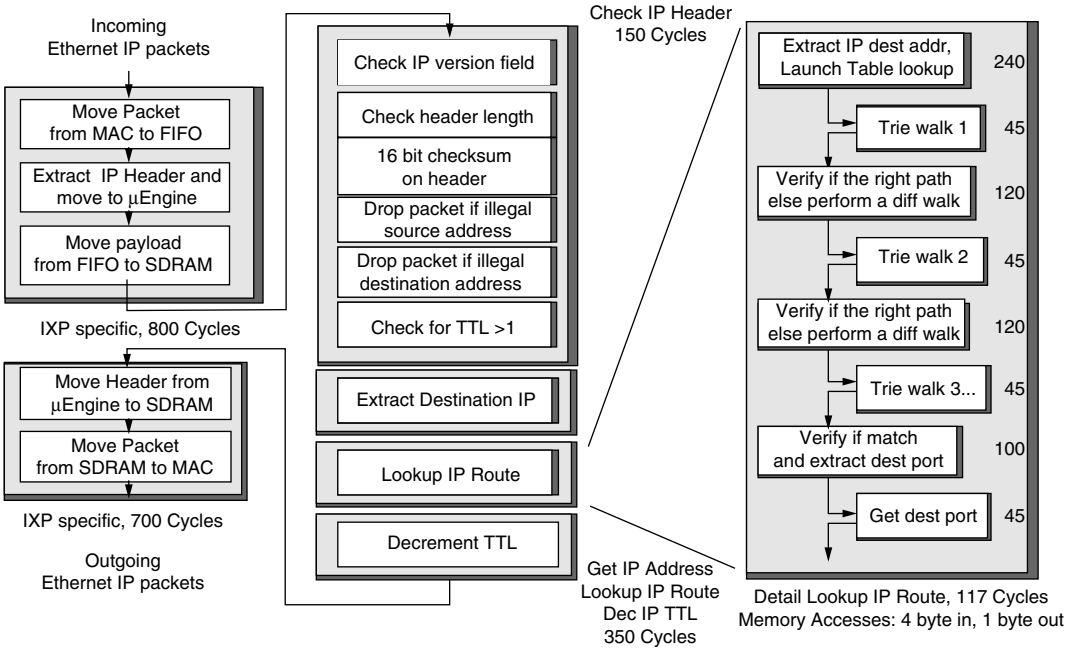


FIGURE 20.10 Instance of annotated IPv4 task graph derived from the application for analysis.

20.3.3 Modeling IPv4 Forwarding Application

A 16 fast Ethernet port IPv4 forwarding switch application is used in this work. Our functional specification of the application is based on RFC 1812 [59]. Figure 20.10 illustrates the main components in the functionality of our benchmark annotated with cycle counts for 64-byte packet size. It is important to note that in addition to the core functionality a number of steps are required to receive the packets from the external media access control (MAC) unit into the IXP1200 and extract the packet header, on which the previously stated operations are performed. Last, the modified packet header and the packet payload need to be written back into the external MAC unit via the IX bus unit. These additional operations, in fact, result in most of the programming effort for our application. For example, 14 detailed tasks are required to perform the core functionality of our benchmark, whereas we need 42 detailed tasks to perform the ingress and egress operations on each packet.

20.3.4 Modeling IXP1200 Architecture

The Intel IXP1200 network processor [48] is targeted for applications performing packet forwarding and classification at layer three and below of the open system interconnection (OSI) model. This section introduces only the main components of the IXP1200 utilized by our application as needed for modeling. The IXP1200 comprises six micro-engines, with four threads on every micro-engine, for computation. There are four unidirectional on-chip buses connecting both the off-chip memories (SRAM and SDRAM) to the micro-engines. External MAC units are connected to the IXP1200 via the IX Bus. The IX bus interface unit has the required logic and memories to receive and transmit packets from/to the external MAC unit. The IX bus unit has a scratchpad memory (SRAM) and two FIFO memories, with each having 16 entries of size 64 bytes. In addition, the SDRAM unit is connected to the IX bus unit via a separate on-chip bus, used to transfer packet payloads directly based on micro-engine commands, and last, an on-chip command bus carries events and signals between micro-engine and the IX bus unit.

This section focuses only on the data plane of the IXP1200 network processor. Thus, aspects related to the StrongARM processor are not modeled. In addition, we have not modeled the PCI bus interface

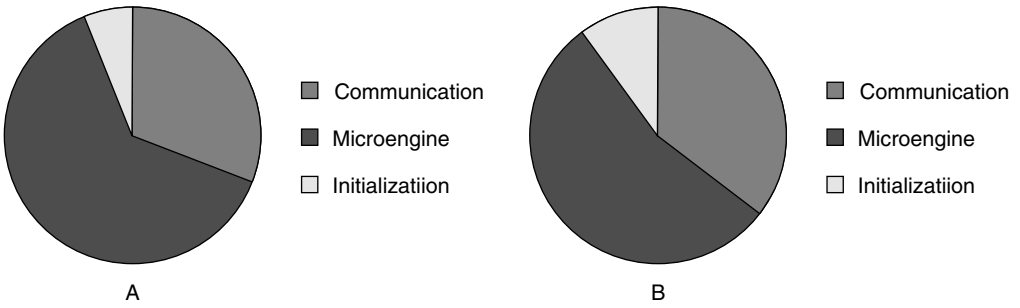


FIGURE 20.11 Assembly and u Engine C code distribution.

as well as the hash engine because we do not utilize these peripherals. A typical packet flow through the IXP1200 follows the steps given in Figure 20.10. Comparing results from analysis and simulation, Section 20.3.5 describes the experimental setup that we used for our implementation.

20.3.5 Experimental Setup

In our case study, we first developed the application in micro-engine C, following the preceding specification based on the Intel reference code. We made a few modifications to the transmit threads to improve the performance and to make the code stable and usable across different packet streams. In addition, we have used the assembly implementation from Intel reference code because it is hand tuned and most performing. The application was partitioned so that 16 threads on 4 micro-engines were assigned 1 port each on the receive (and forwarding) part. The transmit part of the application was assigned eight threads on two micro-engines. This partitioning holds because the end-to-end delay for a packet on the receive part is more than twice that on the transmit part. This implementation was used to derive the per-packet profiling information used to build the task graph for the network calculus-based approach.

Performance on the IXP1200 was measured using version 2.01 of the developer workbench assuming a clock frequency of 200 MHz; the IX bus is 64-bit wide and has a clock frequency of 80 MHz. Two IXF440 external MAC units (with eight duplex fast Ethernet ports each) are connected to the IX bus and Ethernet IP packets are streamed from this unit to the IXP1200 and back. The packets for the application contain destination addresses evenly distributed across the IPv4 32-bit address space. We employ different packet sizes, namely from 40 bytes to 256 bytes. A single packet source for each input port generates an evenly distributed load. In addition, the range of destination addresses and associated next-hop destinations provide an evenly distributed load on every output port.

20.3.6 Results and Analysis

We now present some of our results from the analysis of this implementation.

20.3.6.1 Distribution of Code

Figure 20.11 depicts the total distribution of the IXP1200 assembly and uEngineC code in terms of lines of code devoted to either communication (between different memories and interface units), computation related (includes code for register transfers in micro-engine) and initialization code (related to initialization of ports, etc.). The main observation being the percentage distribution of code is not drastically different compared with the assembly, however, the number of lines of code in uEngineC is more compared with the assembly (620 lines compared with 500 lines).

20.3.6.2 Per-Packet Time Distribution

Figure 20.12 depicts the percentage time distribution based on the previous classification for each packet. Observe that the micro-engine spends a significant amount of time in the idle state. In addition, the

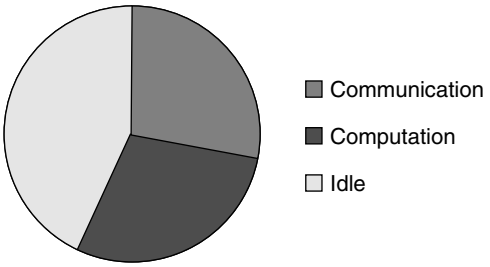


FIGURE 20.12 Per-packet time distribution.

amount of time spent in communication is almost equal to that of computation (Note: Computation here does not indicate arithmetic and logical computations only, but any micro-engine activity including register loads and stores.)

20.3.6.3 Implementation Results

To have a reference set of design points we determined the maximum possible throughput for IPv4 forwarding without packet loss by simulation. We varied the packet length to account for payload storage versus header processing trade-offs. The results are presented in Figure 20.13. Observe that we approach

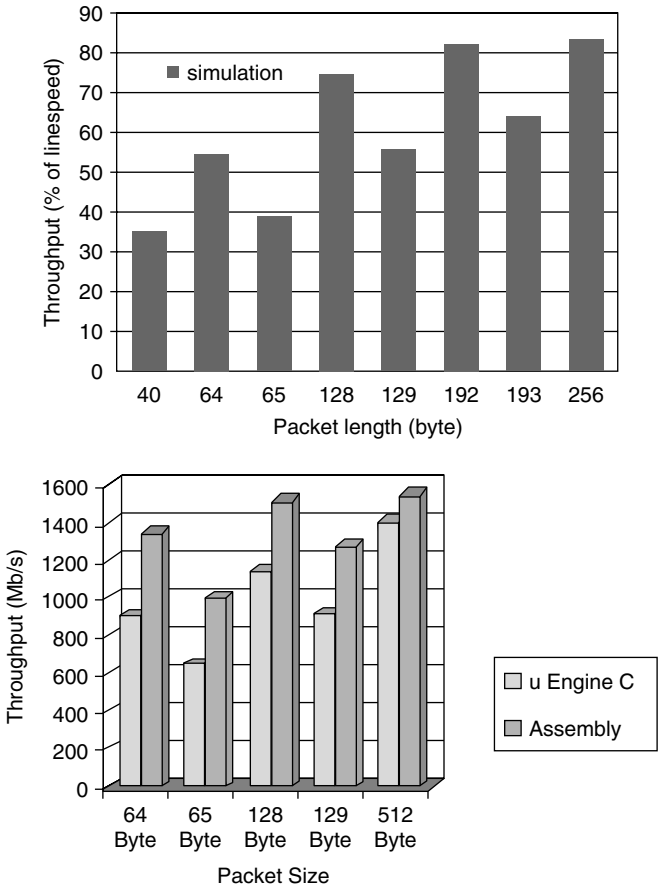


FIGURE 20.13 Throughput for IPv4 forwarding on IXP.

line speed only for larger packet sizes where the micro-engines can keep up with the processing demand of the reduced number of packet arrivals (compared with small packet lengths). We can also recognize the influence of the 64-byte receive and transmit FIFOs in the IX bus unit. As soon as an additional 64-byte segment is needed, the throughput drops, due to the basic unit of data transfer between synchronous dynamic random access memory (SDRAM) and FIFOs being 64 bytes. Thus, for a given delay of two 64-byte transfers we are transferring only 65 bytes (instead of 128 bytes).

Figure 20.13 also presents the comparison between the throughputs obtained by the assembly and uEngineC versions of IXP1200 code. We observe a decrease in performance between uEngineC code and that of assembly code by an average of 25 to 30%. For packets with larger sizes (in our experiments 512 bytes), however, this difference reduces to 5 to 6%. This is because larger packets result in higher throughput for both the cases; however, for the uEngineC code this increase is significant and offsets the difference in performance between the assembly and uEngineC code.

20.3.6.4 Exploring the Implementation Space

Figure 20.14 depicts the variation in throughput with increasing buffer sizes in the external MAC unit (IXF440). We observe that increasing buffer sizes does not contribute to a large increase in the throughput and thus buffer sizes are not the main bottleneck in our set up. Figure 20.14 also presents the variation in the throughput for different IX bus clock frequencies for 64-byte packets. We observe that for both the 80-MHz and 133-MHz, not much change occurs in the throughput; however, by increasing the clock frequency to 200 MHz (the same as the IXP1200 clock frequency), we obtain line rate performance.

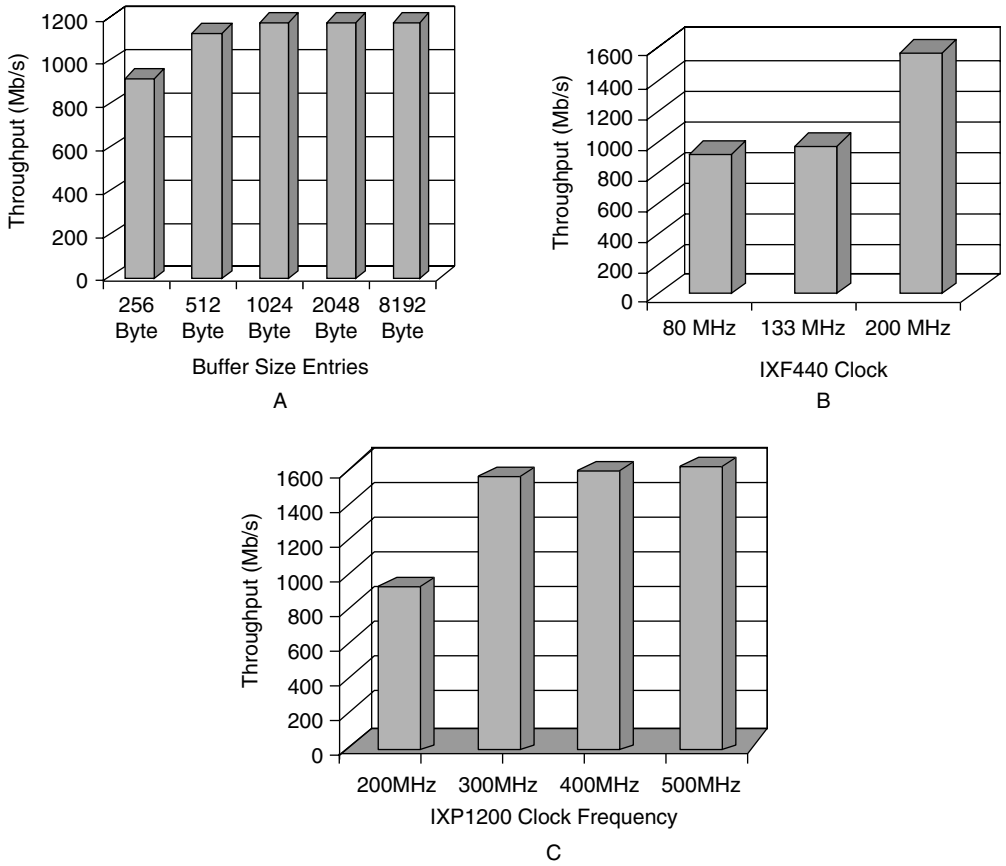


FIGURE 20.14 Throughput for IPv4 forwarding on IXP.

Thus, the IX bus is the bottleneck to attaining line rate performance. Finally, Figure 20.14 depicts the variation in throughput with increasing clock frequency of IXP1200 for IX bus frequency of 80 MHz. We observe that for clock frequencies of 300 MHz and above, we achieve almost a line rate performance. Thus, based on these experiments, we conclude that the two main reasons for performance degradation in IXP1200 are:

1. The speed of IX bus
2. The latency of the IXP1200 itself

Improving either of the two aspects above results in a line rate performance for the given set up.

20.4 Advanced Functional Coverification Using SSDE

Traditionally, software and hardware design activities are clearly separated, as represented by a wall in Figure 20.15. Because of this separation, handing off the hardware IP to the software activity is postponed to the availability of a sufficiently stable and detailed hardware implementation datasheet for the software implementation phase to start. From that point, software IP is developed independent of hardware and hardware engineers can work on finalizing their hardware IP and preparing a rapid prototyping platform like (e.g., an FPGA board). Only after the rapid prototyping platform and the software IP are both available can software and hardware coverification start, usually performed by a separate design integration team.

This traditional approach has two major disadvantages:

1. The software activity remains idle for half of the process, which significantly increases time-to-market.
2. The coverification happens only as a last step in the flow, which does not offer enough room for verifying the system properly.

Moreover, traditional rapid prototyping platforms (e.g., simulation accelerators, FPGA boards, emulators, and early silicon) operate at the Gate-Level (GL), which does not offer the necessary level of visibility for debugging back to the Register-Transfer Level (RTL) and the software programming language level efficiently. As a result, systems are poorly verified; the few bugs found take weeks to fix; several

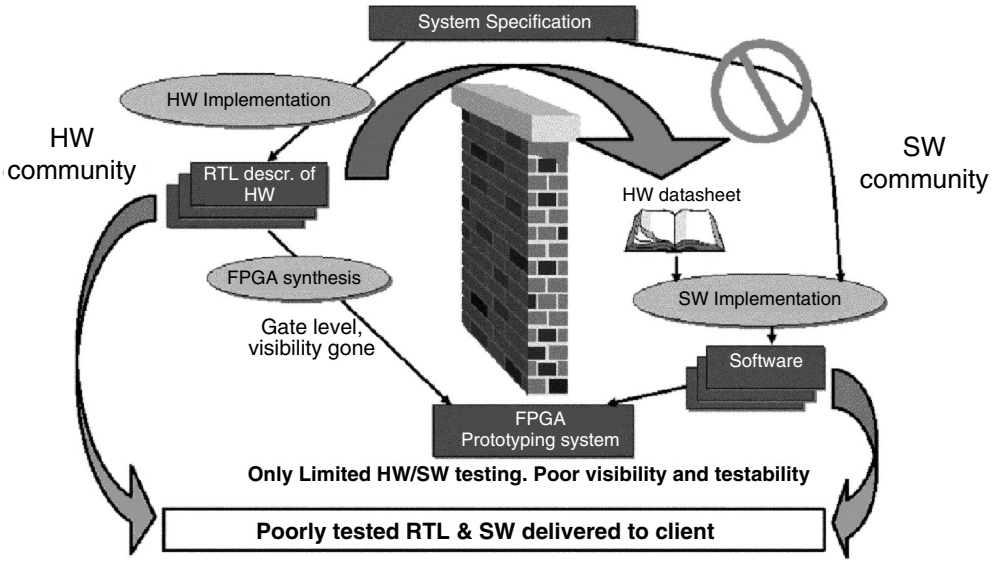


FIGURE 20.15 Traditional wall between the hardware and software communities.

expensive silicon respins are needed before final tape-out; and frustration propagates among the hardware, the software, and the IP integrator communities, not to mention the frustration of the users.

20.4.1 Coverification Using Our System and Software Design Environment (SSDE)

Our proposed SSDE environment alleviates the disadvantages of separating the software and hardware design activities, which was proven on a real USB 2.0 high speed (HS) [29,43], IP9021, business case in cooperation with the Re-Use Technology Group (RTG) Interconnectivity Software Design (ISD) section of the Philips International Technology Center Leuven (ITCL). We illustrate this pilot project in Figure 20.16. Because Seamless from Mentor Graphics, Inc. enables embedded code execution on a simulation model of the hardware, we can use the early RTL-level description of the hardware to start embedded software development at a much earlier stage in the design cycle. From that point, cross-compiling the embedded software into Seamless enables the cycle-accurate software and hardware co-debug with excellent visibility both of the embedded software aspects (i.e., the Seamless XRay* interface provides a conventional software debug interface) and of the hardware signals (i.e., Seamless links to your preferred RTL simulator and waveform viewer). As a result, bugs are found at a much earlier stage and are fixed within a few hours, if not minutes. Because Specman Elite from Verisity, Inc. enables advanced coverage-driven functional verification while offering an e-verification component (eVC) for the USB protocol and protocol checkers for the AHB bus protocol from ARM Ltd., we can verify our full implementation of a USB 2.0 HS device in a realistic working environment — including an AHB bus connecting the ARM processor to the USB device and a USB host generating traffic into the USB device — from the early stages of software development. As a result, the system is verified with a much higher confidence.

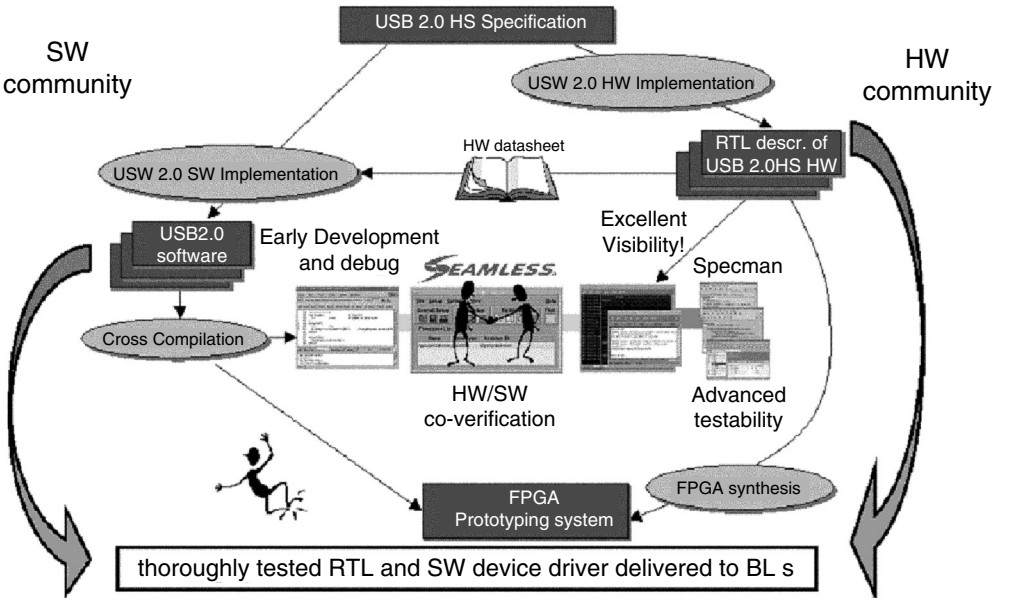


FIGURE 20.16 Breaking the wall between the software and hardware communities by using Seamless from Mentor Graphics, Inc. Boosting the functional verification productivity by using Specman Elite from Verisity, Inc.

*Note that XRay is only one of the software debuggers that may be supported by a processor support protocol (PSP). Others include the ARM debugger and gdb (mainly used for MIPS PSPs). However, only Xray currently supports the interface with Specman via Seamless CVE.

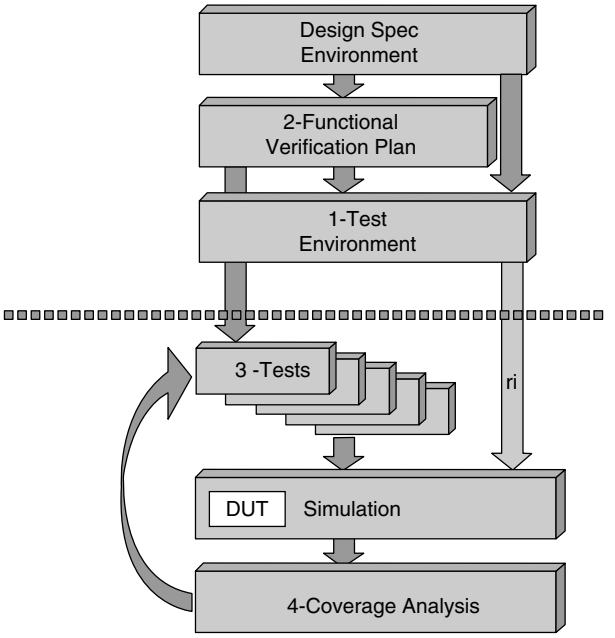


FIGURE 20.17 SSDE Verification Flow: four steps into advanced coverage-driven functional verification.

Because this Seamless/Specman rapid system-level prototyping environment cannot reach very high-speed or even real-time execution, it is still useful to rely on traditional prototyping platforms such as FPGA boards, simulation accelerators, and emulators after the functional specification is verified.

20.4.2 Should I Consider Using SSDE?

Because functional coverification is a broad and weakly defined topic, we need to separate concerns before proceeding further with the presentation of our SSDE methodology. As illustrated in Figure 20.17, we are addressing four steps in the coverification flow:

1. SSDE Test Environment (or Database): the delivery of a proven software and hardware test environment, currently based on Seamless from Mentor Graphics, Inc. and Specman Elite from Verisity, Inc.
2. Functional Test Plan: the tools and methodology support for capturing a functional coverification plan
3. Tests: the tools and methodology support for generating appropriate system tests
4. Coverage Analysis: the tools and methodology support for measuring the coverage quality of the generated tests

20.4.3 Our Generic SSDE Setup

Using our generic SSDE setup in full, Seamless from Mentor Graphics, Inc. reads in the software and fires the cosimulation session. Specman Elite from Verisity, Inc. reads in the e-language test-bench; automatically generates random signal-level tests; performs "on the fly" data and temporal checks; and measures functional coverage.

20.4.4 Overview of Seamless

Starting from a programmable platform design under test (DUT), which may, for example, consist of a CPU, a memory, a direct memory access (DMA) controller, and an I/O block, we use Seamless from

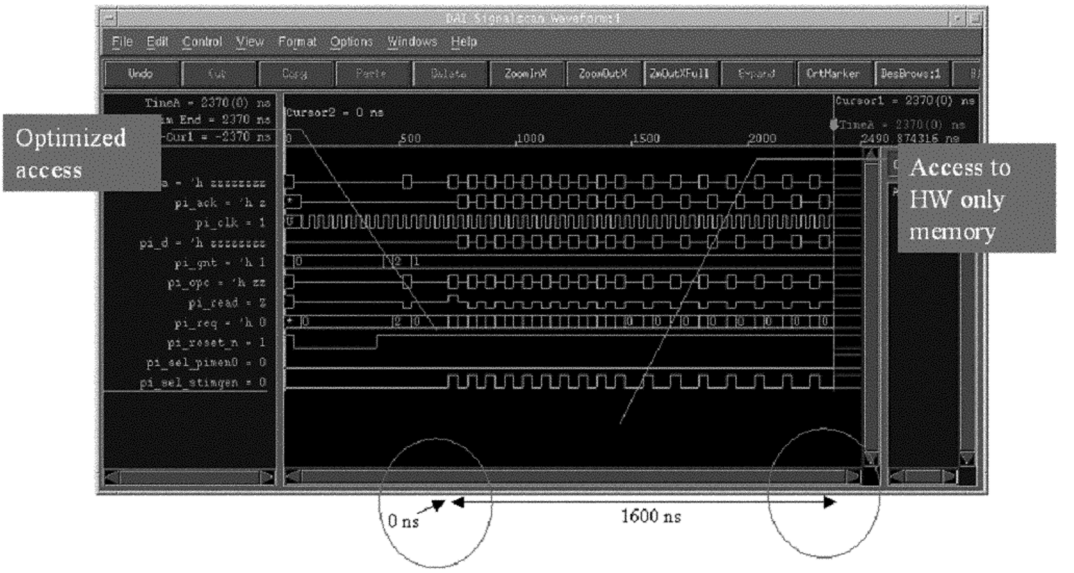


FIGURE 20.18 Optimized software and hardware co-verification using Seamless from Mentor Graphics, Inc.

Mentor Graphics, Inc. to abstract away the CPU and the memory from the DUT. In practice, this requires purchase of a so-called Processor Support Package (PSP) for the processor being used (e.g., ARM or MIPS) along with the basic Seamless license. Please note that Seamless does not support multi-core debugging, which will be addressed by future SSDE releases only. From that point, software can be directly cross-compiled into Seamless that will automatically take care of applying tests to the DUT while allowing cycle-accurate co-debug with excellent visibility of the software and the hardware altogether. As a result, tests can be abstracted away from the detailed signal-level implementation of the test-bench, enabling an easier test-suite portability across, for example, various RTL simulators, various FPGA prototyping boards, emulators, and final silicon [10] while leveraging engineering productivity in the tedious process of test-bench creation. Currently, no standard Transaction Simulation Language (TSL) has been defined to capture the abstracted test-suite, but discussions are under way within the Philips Semiconductors Advanced Functional Verification Workgroup (AFV-Wg) and the SystemC Verification Working Group [54]. Within Philips Semiconductors, many business lines (BLs) already have experience in developing their own standard language and test-suite reuse infrastructure, which should offer an ideal transition path to SSDE.

Figure 20.18 illustrates a typical waveform produced by a Seamless-based cosimulation. Within such a Seamless run, we distinguish three types of events:

1. Access to optimizable memory: These are accesses to memories that support high-level modeling within the Seamless environment.
2. Access to unoptimizable memory: These are accesses to memories that do not support high-level modeling within the Seamless environment, which is often the case for memories embedded inside the hardware part.
3. Standard event: This is any other type of event.

As illustrated in Figure 20.18, when switching Seamless to the high-level modeling mode, accesses to optimizable memories are removed from the cycle-accurate cosimulation. As an effect, optimized Seamless cosimulations can easily be 10× faster than a conventional cosimulation, in the range of 100 kcycles/sec.

20.4.5 Overview of Specman Elite

Starting either from a Seamless setup or from the RTL description of a hardware block alone, Specman Elite from Verisity, Inc. offers all the necessary support for introducing an advanced coverage-driven

verification methodology [60]. The idea is to iteratively constrain the test-suite to match the needs captured in the functional test plan with 100% accuracy, so-called coverage. This process is in detail. Using the fully integrated Seamless/Specman environment, Specman has visibility over the full software and hardware design. It thus becomes possible to verify any piece of functionality that was intentionally implemented partly in software and partly in hardware.

20.4.6 Functional Verification Plan

20.4.6.1 Specification-Based Verification

Because bugs are typically hidden in design corner cases that humans and sophisticated design tools do not capture effectively, extensive functional verification is about checking (or proving) that a system behaves correctly for all possible corner cases. To achieve this task effectively, best practice recommends starting with a definition of a functional test plan from the actual system functional specification [57] instead of a detailed paper or RTL implementation, which makes the task of extracting corner cases even harder. In the case of a standardized protocol, such as USB, this information is publicly and freely accessible in documents such as Ganssle [29] and Greef et al. [43]. For more specific developments, it is very important that this information is clearly defined in the system specification document presented in the SSDE1.0 flow from Figure 20.17. Compared with a design-centric flow, an advanced functional verification strategy requires that much more attention is spent on defining (or just collecting) a clean and Golden System Specification from the early stages of the design cycle, which Verisity, Inc. calls "specification-based verification" [57]. An important task (and second step in our SSDE1.0 flow presented in Figure 20.17) is to derive the functional test plan from this system specification.

20.4.6.2 e-Based Executable Test Plan

Many approaches are used for capturing a functional test plan. The traditional approach consists in making an explicit list of items to be tested. This approach clearly does not scale as the number of items grows exponentially for complex systems. Therefore, engineers need to compromise either for an incomplete list (i.e., a low coverage list) or for an ambiguous list (i.e., a list that does not really say what should be done). Both compromises are of course not recommendable. The research approach consists of automatically generating tests from a formal and, therefore, unambiguous specification [3,5]. This approach scales much better than the latter, but it still cannot address the complexity of today's SOC designs. It can be used for specific needs only. Between these two approaches is the e-language from Verisity, Inc., which allows for the capture of a higher-level (and therefore incomplete) list of items to be tested from a verification language, which is nicely complemented by automatic random test generation [61]. Provided the system under consideration is well suited for random test generation [1], scalability and coverage can be very high, which cannot be achieved by any other approach. Examples of a good fit are core datapath verification, where computed data can take any random value, and telecommunication application verification, where transmitted packets can carry any random information. Examples of a bad fit are complex protocols requiring high order coverage [2], which cannot be reached by chance.

As illustrated in Figure 20.19, engineers typically think about electronic and software design in two orthogonal dimensions. The first dimension relates to pieces of information, such as a packet, a channel, an Ethernet frame, and a memory buffer, that are ideally modeled using object-programming techniques [27,45]. The second dimension relates to pieces of functionality such as basic definitions, interfaces to the DUT, assertion checkers, coverage computations, and specific corner case tests. Such design aspects typically span over several objects and are therefore not captured effectively by standard object programming. By offering a clean separation of objects and aspects [33], the e-language enables mixed object-aspect programming. To extend an aspect definition, you do not need to edit several base objects and vice versa. This is extremely useful for capturing many corner cases in the least amount of development effort and time.

More precisely, let us study some mixed object-aspect programming examples. Imagine that you want to functionally verify a communication agent that sends transactions over a network. These transactions

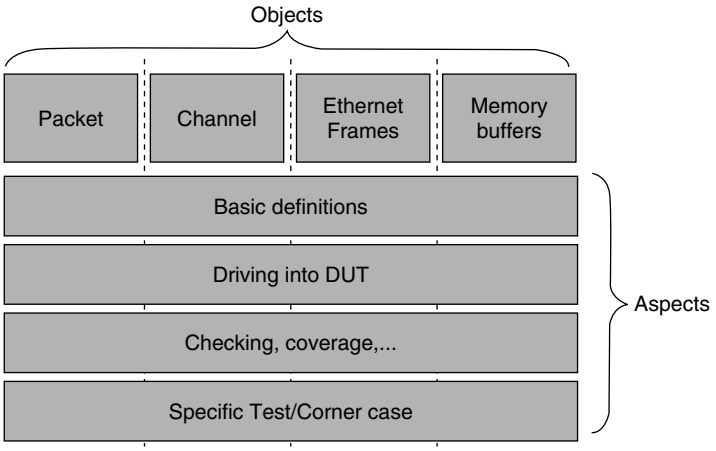


FIGURE 20.19 Objects and aspects are orthogonal programming paradigms.

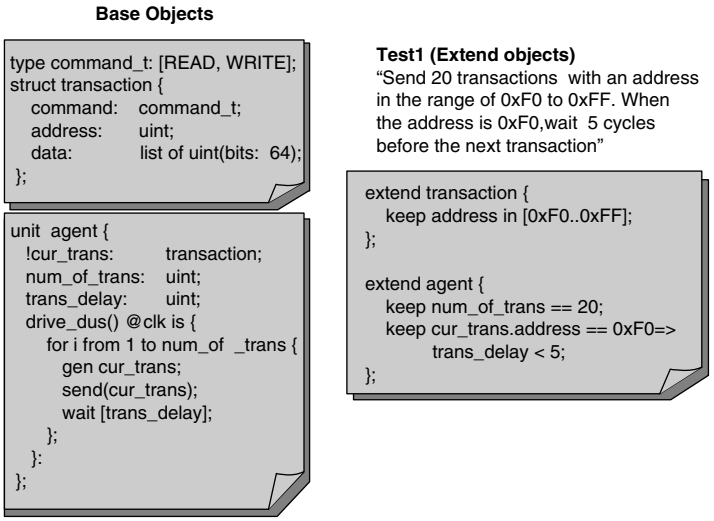


FIGURE 20.20 Using object-aspect programming, file editing is minimized for refining the functional test plan with corner conditions.

consist of a command, a destination address, and some data, which can be easily captured from an object description, as illustrated in Figure 20.20. As a first incomplete approximation of your functional test plan, you wish to generate a large number of such transactions and drive them through the bus on the rising edge of a clock. You also want to wait a number of cycles between two transactions. A first refinement of this functional test plan is to constrain the number of transactions, the address space, and the delay. This is ideally captured by an aspect extension using the extend construct in the e-language.

Imagine that you now want to add a debug feature by capturing the time and content of each transaction sent. As presented in Figure 20.21, this is made possible by extending the transaction object with a start time — the agent object with a list of transactions and the send method from the agent object with a time stamp and history addition.

Finally, imagine that a new derivative of the DUT supports burst mode accesses for transactions of size 4, 8, 16, 32, and 64. As presented in Figure 20.22, this new feature can be tested by constraining the command type to include a BURST mode and the data instance variable from the transaction object to match the required size for burst execution. In conclusion, using mixed object-aspect programming for

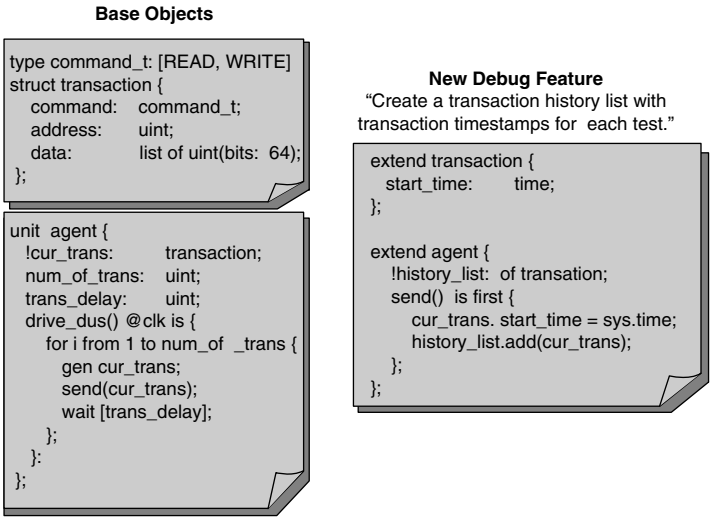


FIGURE 20.21 Using object-aspect programming, debugging features are easier to introduce and manipulate.

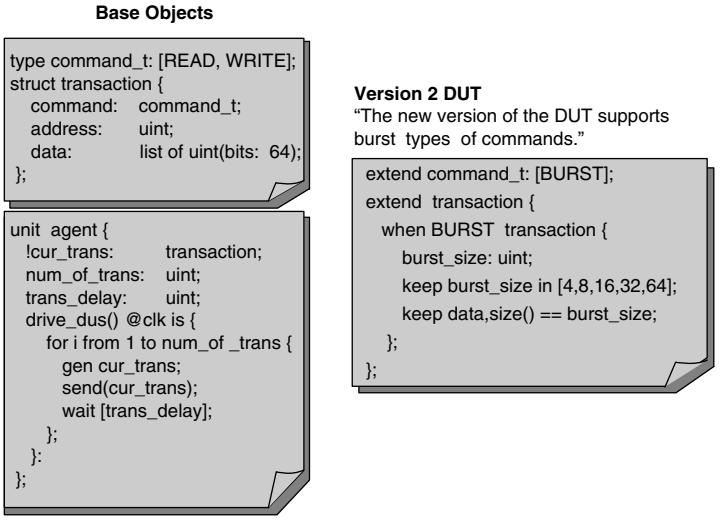


FIGURE 20.22 Using object-aspect programming, file editing is minimized for testing a derivative design.

capturing the functional test plan is superior not only to conventional verification techniques for productivity, but is also superior for maintainability and debugability purposes.

20.4.7 Random Test Generation

As soon as the functional test plan is available, random test generation can start [61]. When captured using the object-aspect programming capabilities of the e-language from Verisity, Inc., this process is completely automatic. Section 20.4.8 describes the advantages and disadvantages of this technology.

20.4.8 Manual Tests Development

As already mentioned, the traditional approach of making an explicit list of items to be tested and implementing them does not scale. This verification effort is proportional to the square of capacity in the best case, and, therefore, at least doubles every 6 to 9 months according to Moore's Law. Moreover,

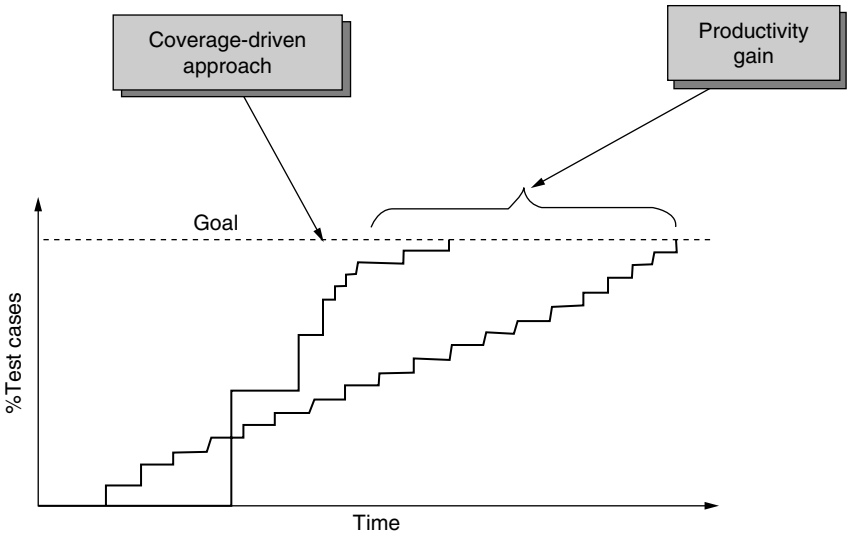


FIGURE 20.23 Coverage-driven verification reuse process.

when performing this task manually in an ad-hoc way, verification reuse across similar verification needs and portability across verification platforms and environments is hard, if not impossible, further extending the waste in nonreusable engineering (NRE) cost. Figure 20.23 illustrates the process of developing tests manually to achieve a goal specified in the functional test plan. Whenever this goal falls beyond the human perception, the only possible behavior for the designer is to write several small tests hoping that the global verification objective will be solved. In the best case, this process is tedious and slow. In the worst case, the designer quickly gets confused and introduces highly redundant tests that may never converge to the wanted goal. This redundancy typically accounts for a large part of the NRE overhead for complex systems. Moreover, by artificially growing the verification complexity, it may also account for the need to rely on very expensive high-capacity verification infrastructure such as emulators.

20.4.9 Automatic Test Pattern Generation

Instead of writing tests by hand, it is possible to rely on automatic test pattern generation (ATPG). Several kinds of ATPG exist, but we focus here on the random type [1,61], using a simplified three-step version of the methodology evolution concept from Verisity, Inc. [59]. The first step consists in relying on fully automatic random test generation from the functional test plan captured in the e-language. Because tests are generated from an intelligent engine, their statistical repartition is homogeneous, which tends to achieve less redundancy than conventional tests written by hand. Because design corner cases are by definition unfit for human perception, randomly generated tests tend to reach them more easily. As a result, automatic random test generation is likely to converge to verification objectives in less time and effort than the manual approach. This productivity gain is illustrated in Figure 20.23.

The second step consists of introducing coverage metrics to drive the quality of tests [60]. Because tests are steered by a measured objective, most of the redundancy can be removed and productivity is largely improved especially toward the end of the verification project, as illustrated in Figure 20.23.

The third step consists in structuring the verification IP in a reusable form. Whenever this IP is reused, a significant portion of the verification effort is saved. This results in an earlier time-to-first-test and a higher productivity win-win situation, as illustrated in Figure 20.23.

In conclusion, by properly exploiting the capabilities of random test generation, it is possible to improve both the availability of first results and the overall project productivity. Whenever coverage-driven reuse applies, the initial tool support and training investment can be recovered from the first day compared

with traditional manual and RTL-level practice. In this situation, there should be no hesitation in adopting our SSDE methodology.

References

[1] V. Agrawal and R. Mercer. Deterministic versus random testing. *Int. Test Conf.,* Los Angeles, CA, pp. 718–718, September 1986.

[2] G. Apostol (Brecis Communications, Inc.). Network processor designer tackles verification nightmare. *EE Design,* November 5, 2001.

[3] L. Arditi, H. Boufaied, A. Cavanie, and V. Stehle. Coverage directed generation of system-level test cases for the validation of a DSP system. *Int. Symp. on FME 2001: Formal Methods for Increasing Software Productivity,* LNCS, Vol. 1, 2001.

[4] F. Baker. *Requirements for IP Version 4 Routers. RFC1812,* Internet Engineering Task Force (IETF), June 1995.

[5] M. Benjamin, D. Geist, A. Hartman, G. Mas, R. Smeets, and Y. Wolfsthal. A feasibility study in formal coverage driven test generation. Technical Report, IBM Haifa Laboratories, Haifa, Israel, June 1999. http://www.haifa.il.ibm.com/projects/verification/gtcb/.

[6] G. Berry. *The Foundations of Esterel.* G. Plotkin, C. Stirling, and M. Tofte, Eds., MIT Press, 2000.

[7] Y. Bouchebaba. Optimisation des transferts de données pour le traitement du signal. Ph.D. thesis, Ecole Nationale Supérieure des Mines de Paris, 2002.

[8] Y. Bouchebaba and F. Coelho. Pavage pour une séquence de nids de boucles. *J. Technique et science informatiques. Parallélisme et systèmes distribués* 21, numéro 5, pp. 579–603, 2002.

[9] Y. Bouchebaba and F. Coelho. Tiling and memory reuse for sequences of nested loops. *Euro-Par 2002, Parallel Process., 8th Int. Euro-Par Conf. Proc. Lecture Notes in Computer Science 2400,* pp. 255–264, 2002.

[10] R. Brackebusch, S. Muller, G.S.-Y. Sokomak, F. Grassert, and D. Timmermann. A new synthesizable architecture approach for verification environments applying transaction-based methodology. *Proc. 40th Design Automation Conf. (DAC '03),* Anaheim, CA, June 2003.

[11] A. Bruce and J. Goodenough (ARM, Ltd.). Re-usable hard-ware /software co-verification of IP blocks. Verisity Design, Inc. Club Verification, June 2002.

[12] D. Callahan. A global approach to detection of parallelism. Ph.D. thesis, Rice University, Houston, TX, 1987.

[13] F. Catthoor et al. *Custom Memory Management Methodology — Exploration of Memory Organization for Embedded Multimedia System Design,* Kluwer Academic Publishers, Dordrecht, 1988.

[14] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. DeMan. Global communication and memory optimizing transformations for low power signal processing systems. *IEEE Workshop on VLSI Signal Process.,* pp. 178–187, 1994.

[15] H. Chang, L. Cooke, M. Hunt, G. Martin (Cadence Design Systems, Inc.), A. McNelly and L. Todd (Simutech, Inc.). *Surviving the SOC Revolution: A Guide to Platform-Based Design.* Kluwer Academic Publishers, Dordrecht, 1999.

[16] A. Chatelain, Y. Mathys, G. Placido (Motorola, Inc.), A. La Rosa and Luciano Lavagno (Politecnico di Torino). High-level architectural co-simulation using Esterel and C. *CODES '01,* pp. 189–194.

[17] Intel Corporation, *Intel IXP1200 Network Processor Family: Hardware Reference Manual, Revision 8,* Intel Corporation, Santa Clara, CA, pp. 225–228, 2001.

[18] P. Crowley, M. Fiuczynski, J. Baer, and B. Bershad. Characterizing processor architectures for programmable network interfaces, *Proc. 2000 Int. Conf. on Supercomputing,* Santa Fe, NM, May 2000.

[19] A. Darte. On the complexity of loop fusion, *Parallel Computing,* Vol. 26, No. 9, 2000, pp. 1175–1193.

[20] D. Dempster and M. Stuart (TransEDA, Ltd.). *Verification Methodology Manual: Techniques for Verifying HDL Designs.* Kluwer Academic Publishers, Dordrecht, 2002.

[21] F. de Dinechin, P. Quinton, and T. Risset. Structuration of the Alpha language, in *Massively Parallel Programming Models,* IEEE Computer Society Press, Berlin, Germany, pp. 18–24, 1995.

[22] E.A. de Kock, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, K.A. Vissers, and G. Essink. YAPI: application modeling for signal processing systems. *Proc. 37th Conf. on Design Automation (DAC-00),* NY, pp. 402–405, June 5–9, 2000.

[23] H. de Man. Washing machine: the key to low-power. *EE Times,* March 6, 2002, http://www.electronicstimes.com/.

[24] C. Eisenbeis, W. Jalby, D. Windheiser, and F. Bodin. A strategy for array management in local memory, *J. Mathematical Programming: Series A,* Vol. 63, No. 3, pp. 331–370, 1994.

[25] C. Eisner and D. Fisman. Sugar 2.0: an introduction. Technical Report, IBM Haifa Research Laboratory, Haifa, Israel, 2002.

[26] Esterel Technologies, S.A. Esterel technologies develops top-level validation methodology for STMicroelectronics: Speeds functional verification of chips with multiple design blocks. Available at http://www.esterel-technologies.com, June 2002.

[27] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Elements of Reusable Object-Oriented Software, Professional Computing Series,* Addison-Wesley, Reading, MA, 1994.

[28] D. Gannon, W. Jalby, and K. Gallivan. Strategies for cache and local memory management by global program transformation. *J. Parallel and Distributed Computing,* Vol. 5, No. 10, pp. 587–616, 1988.

[29] J.G. Ganssle. An introduction to USB development. Embedded Systems Programming, 2002. Available at http://www.embedded.com.

[30] E. de Greef, F. Catthoor, and H. de Man. Reducing storage size for static control programs mapped onto parallel architectures, presented at *Dagstuhl Seminar on Loop Parallelisation,* Schloss Dagstuhl, Germany, April 1996.

[31] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Comparing analytical modeling with simulation for network processors: a case study, *Design Automation and Test in Europe (DATE),* Munich, Germany, March 2003.

[32] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proc. IEEE*, Vol. 79, No. 9, pp. 1305–1320, September 1991.

[33] Y. Hollander, M. Morley, and A. Noy. The e-language: a fresh separation of concerns. *Proc. Technology of Object-Oriented Languages and Syst. (TOOLS) Europe,* Zurich, Switzerland, March 1999, pp. 41–50.

[34] G. Huard. Algorithmique du décalage d'instructions. Ph.D. thesis, École Normale Supérieure de Lyon, 2001.

[35] International Technology Roadmap for Semiconductors (ITRS). Design chapter of the 2001 edition. Technical Report, EECA, JEITA, KSIA, TSIA, SIA, and International SEMATECH, 2001. Available at http://public.itrs.net.

[36] F. Irigoin and R. Triolet. Supernode partitioning. *Proc. 15th Annu. ACM Symp. on Principles of Programming Languages,* San Diego, CA, pp. 319–329, 1988.

[37] F. Irigoin, P. Jouvelot, and R. Triolet. Overview of the PIPS project. *Proc. Int. Workshop on Compilers for Parallel Computers,* Paris, France, November 1990.

[38] M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and H.S Kim. Experimental evaluation of energy behavior of iteration space tiling, *LCPC 2000,* Yorktown Heights, NY, pp. 142–157, 2000.

[39] K. Kennedy. Fast greedy weighted fusion. *Int. J. Parallel Programming,* Vol. 29, No. 5, pp. 463–491, 2001.

[40] B. Kienhuis, E. Deprettere, K.A. Vissers and P. Van Der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. *Proc. Int. Conf. on Application-Specific Syst., Architectures and Processors (ASAP '97),* pp. 338–349, 1997.

[41] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click Modular Router. *ACM Trans. on Computer Syst.,* Vol. 18, No. 3, pp. 263–297, August 2000.

[42] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann. Polychrony for system design. Technical Report RR-4715, IRISA, Environnement de spécification de programmes réactifs synchrones (ESPRESSO), June 2003.

[43] C.-W. Leong. Understanding the universal serial bus (USB). USB developer, 2002. Available at http://www.USBDeveloper.com.

[44] K. McKinley and K. Kennedy. Maximizing loop parallelism and improving data locality via loop fusion and distribution, languages and compilers for parallel computing, *6th Int. Workshop,* Portland, Oregon, pp. 301–320, 1993.

[45] Bertrand Meyer. Object oriented software construction. In C.A.R. Hoare, Ed., *Series in Computer Science.* Prentice Hall International, Inc., Englewood Cliffs, NJ, 1988.

[46] A. Mihal, C. Kulkarni, M. Moskewicz, M. Tsai, N. Shah, S. Weber, Y. Jin, K. Keutzer, C. Sauer, K. Vissers, and S. Malik. Developing architectural platforms: a disciplined approach, *IEEE Design and Test of Computers,* Vol. 19, No. 6, pp. 6–16, November/December 2002.

[47] A. Mozipo, D. Massicote, P. Quinton, and T. Risset. Automatic synthesis of a parallel architecture for Kalman filtering using MMAlpha. *IEEE Canadian Conf. on Electrical and Comput. Eng.,* Edmonton, Canada, May 1999.

[48] J. Nickolls, L.J. Madar III, S. Johnson, V. Rustagi, K. Unger, and M. Choudhury, Broadcom Calisto: a multi-channel multi-service communication platform, *Hot-Chips Symp.,* 2002.

[49] T. Pascalin Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten. Signal. *Lecture Notes in Computer Science,* Vol. 891, pp. 113–, 1995.

[50] Philips Semiconductors, B.V. Nexperia pnx8500: Home entertainment engine. Functional Overview, 2000. Available at http://www.semiconductors.philips.com/nexperia.

[51] S.K. Roy (Synplicity, Inc.), S. Ramesh, S. Chakraborty (IITBombay), T. Nakata, and S.P. Rajan (Futjitsu Laboratories). Functional verification of systems on chip (SOCs) — practices, issues and challenges. *ASP-DAC/VLSI Design 2002,* Bangalore, India, pp. 11–, January 7–11, 2002.

[52] M. Scott, J. Dickerson, and B. Payne. Panel probes SOC problems, solutions. *EE Design,* February 2002. http://www.eedesign.com/news/OEG20000202S0044.

[53] N. Shah, Understanding network processors. Master's thesis, Department of Electrical Engineering and Computer Sciences, University of California–Berkeley, September 2001.

[54] SystemC Verification Working Group. SystemC verification standard specification. Technical Report, Open SystemC Initiative (OSCI), November 2002. Available at http://www.systemc.org.

[55] Teja Technologies, IPv4 forwarding application performance, White Paper, July 2002. Available at http://www.teja.com/library/ip4_whitepaper.html.

[56] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, and K. Keutzer, A benchmarking methodology for network processors, *First Workshop on Network Processors at the 8th Int. Symp. on High Performance Computer Architecture (HPCA8),* Cambridge, MA, February 2002.

[57] Verisity Design, Inc. Spec-based verification. White Paper, 1999. Available at http://www.verisity.com/resources/whitepaper/.

[58] Verisity Design, Inc. e-reuse methodology (eRM) developer manual: maximizing verification productivity. Technical Report, Verisity Design, Inc., 2001.

[59] Verisity Design, Inc. The evolution of verification methodology. Technical Report, Verisity Design, Inc., 2001.

[60] Verisity Design, Inc. Coverage-driven functional verification: using coverage to speed verification and ensure completeness. White Paper, September 2001. Available at www.verisity.com/resources/whitepaper/.

[61] J.A. Waicukauski, E. Lindbloom, E.B. Eichelberger, and O.P. Forlenza. A method for generating weighted random test patterns. *IBM J. Res. and Dev.,* Vol. 33, No. 2, pp. 149–161, March 1989.

[62] M.E. Wolf. Improving locality and parallelism in nested loops. Ph.D. thesis, Stanford University, Stanford, CA, 1992.

[63] M. Wolfe. High-*Performance Compilers for Parallel Computing,* Addison-Wesley, Reading, MA, 1996.

[64] S. Wuytack, J.P. Diguet, F. Catthoor, and H. De Man. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings, *IEEE Trans. on VLSI Syst., Special Issue ISLPED '97,* Vol. 4, No. 6, pp. 529–537, December 1998.

[65] H.P. Zima and B.M. Chapman. *Supercompilers for Parallel and Vector Computers,* Addison-Wesley, Reading, MA, 1990.