Maarten Ditzel
Ralph H.J.M. Otten
Wouter A. Serdijn

Power-Aware Architecting

for data-dominated applications

Springer

Power-Aware Architecting
for data-dominated applications

Power-Aware Architecting
for data-dominated applications

by

Maarten Ditzel
Netherlands Organisation for Applied Scientific Research TNO
The Netherlands

Ralph H.J.M. Otten
Eindhoven University of Technology
The Netherlands

and

Wouter A. Serdijn
Delft University of Technology
The Netherlands

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN 978-1-4020-6419-7 (HB)
ISBN 978-1-4020-6420-3 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

Preface

The complexity of embedded systems-on-a-chip is rapidly growing. Different experts are involved in the design process: application software designers, programmable core architects, on-chip communication engineers, analog and digital designers, deep submicron specialists and process engineers. In order to arrive at an optimum implementation compromises are needed across boundaries of the different domains of expertise.

Therefore, the authors of this book take the point of view of the system architect who is a generalist rather than an expert. He is responsible for the definition of a high level architecture, which is globally optimal. Finding an optimum requires a proper balance between area, performance and last but not least energy consumption. The challenge is not only the size of the design space but also the fact that the most important decisions are taken during the early design phases. The advantage of an early decision is that the impact on area, performance and energy consumption is large. But the disadvantage is that the available information is often limited, incomplete and inaccurate. The task of the system architect is to take the correct early decisions despite the uncertainties.

This book provides a systematic way to support the system architect in this job. Therefore, an iterative system-level design approach is defined where iterations are based on fast and accurate estimations or predictions of area, performance and energy consumption. This method is illustrated with a concrete real life example of multi-carrier communication. This book is the result of a Ph.D. thesis, which is part of the UbiCom project at Delft University of Technology. I strongly recommend it to any engineer, expert or specialist, who is interested in designing embedded systems-on-a-chip.

Eindhoven, The Netherlands,
April 2007

Prof. dr. ir. Jef van Meerbergen
Eindhoven University of Technology
Fellow Philips Research Eindhoven

Contents

Introduction

Worldwide, the demand for portable, hand-held devices is increasing strongly. More and more cellphones, personal digital assistants (PDAs), MP3 players and Game Boys are bought. At the same time, there is an increase in the number of different tasks such devices are expected to be capable of doing, ranging from image processing to data communications. Thus, an overall trend is visible towards ever smaller devices integrating ever more functions. Furthermore, most processing tasks tend to be data-centric, i.e., most computational effort is put in processing data streams. As such, the operating cores can be seen as data stream processors.

The tendency of cramming more functionality in smaller devices is enabled by the continuing growing capabilities of semi-conductor technology. Ever smaller devices can operate on ever higher frequencies, at lower operating supply voltages. Unfortunately, the advances in semi-conductor technology do not solve all of our problems, because we do not want to do the same with improved devices; we want to do more with less. Therefore, the tools used to design and create the chips have to be improved, so that the increasing design gap between ideas and silicon realizations can be bridged.

The call for portable devices, especially in telecommunication applications, imposes increasingly strict requirements on, for instance, the dimensions and the energy consumption of the apparatus. Ideally, your cellphone should be able to operate for months on a single lightweight battery. Unfortunately, the energy storage capacity of batteries is only being improved slowly [58, 67]. Therefore, already at the design stage, care must be taken to reduce the energy consumption of the devices as much as possible, but this should not affect the performance of the device or the required silicon area. A designer must find a balance between these competing design objectives.

In this book, a high-level design method is presented that aids the designer in finding that balance. Furthermore, design automation tools are implemented to test and verify this method for the design of an OFDM (orthogonal frequency-division multiplexing) transceiver. This chapter introduces the ideas and concepts underlying the method and tools.

1.1 High-level system design

As device sizes shrink, the complexity of a chip increases. To describe the complex designs, designers have to make use of languages at higher levels of abstraction in order to meet time-to-market deadlines. Current hardware description languages (HDLs) such as VHDL (very high speed integrated circuit hardware description language) and Verilog are not equipped to handle these levels of abstraction. As a result, designers resort to general programming languages to start with the functional exploration of their designs.

High-level exploration of the design space is crucial when one must design an embedded system with competing objectives while limited design time is available. Significant changes to the functionality or the architecture at lower levels are extremely expensive and should be avoided in all cases. Furthermore, choices made at a high level tend to have a huge impact on the performance and costs of the final design. For example, given an ultra low-power multiplier, a change in algorithm that doubles or triples the number of multiplications still causes a large increase in the power consumption. In figure 1.1, the trade-offs between abstraction level, the cost of changes, the impact of design decisions and the available information is depicted in a so-called abstraction pyramid, originally introduced in [35]. A similar representation of the "reachability" can be found in [13].

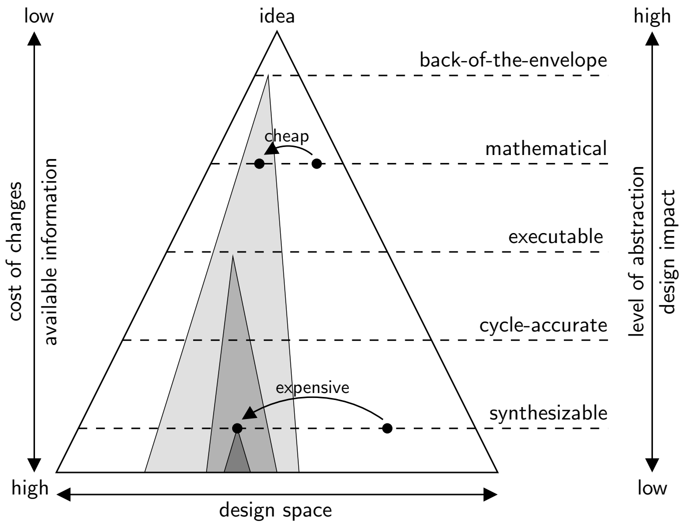


Figure 1.1. The abstraction pyramid represents the trade-offs between the abstraction level, the cost of changes, the impact of design decisions and available information.

Because of their popularity as general-purpose programming languages, and the corresponding abundance of libraries and tools, C and C^{++} are commonly used to construct an executable specification of the functional part of the design. The desired functionality is described by a set of functions that together implement the algorithm. Functions are expressed purely sequentially and an explicit notion of time is lacking. Hence, timing can only be expressed as the order in which the functions are called.

However, the high abstraction level of these programming languages allows for a very terse description of the algorithm. Moreover, changes to the algorithm are easily incorporated and, because of the high simulation speeds, quickly verified. These properties make programming languages like C or C^{++} the de facto standard for the initial, high-level specification of a complex design, even though originally these languages were never intended for this purpose.

1.1.1 Hardware–software partitioning

Once the initial specification of the algorithm is fixed, one of the first questions that arises is, which parts of the algorithm should be implemented in hardware and software in order to find an implementation that best suits the design's requirements (see figure 1.2). A key problem is that at this stage, usually only insufficient or at least inaccurate information to make a proper decision is available to the designer. Moreover, the partitioning problem is a multi-objective optimization problem and although methods for multi-objective optimization exist, normally the designer wants to have a firm control over the decision process. In the end, it is up to the designer to prioritize the design objectives.

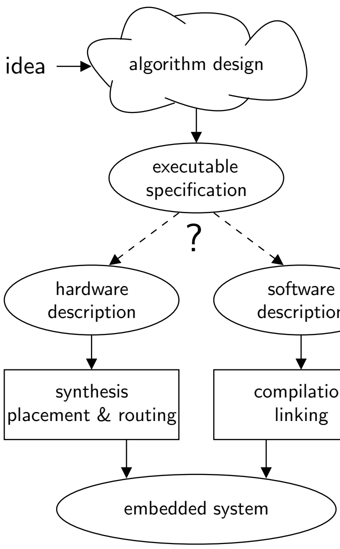


Figure 1.2. Once the algorithm is fixed, the design flow of an embedded system starts with an executable specification. Subsequently, it is decided which parts to implement in software and which in hardware.

The overwhelming number of options and the pace with which they change, render it impossible for a human to make a sound decision about today's complex designs. We simply cannot predict the consequences of a particular choice accurately. Therefore, a tool to quickly identify the trade-offs is indispensable in making well-founded partitioning decisions. Such a tool should be able to cope with the inherent uncertainties due to the high level of design.

In addition to the aforementioned uncertainties, there is another complicating factor: the precise algorithm or the set of algorithms to be implemented may not even be given yet. Several implementations and alternatives may be available, out of which the best should be selected and for which an optimal architecture should be constructed.

In support of the partitioning tool, additional instruments should be available to quickly generate estimates of cost and performance figures, if not absolute then at least relative numbers. If all these tools are available, a designer can quickly explore different partitionings and select the one that best suits his needs. In this book, a solution to the partitioning problem is presented. Its workings are verified by implementing the solution in a partitioning tool and applying it to a test case.

The partitioning tool takes as input a series of algorithms specified as data flow graphs (DFGs). The possible architectures are specified by data processing blocks, such as processing cores and memories, and data transfer elements such as busses. The optimization problem itself is formulated as a mathematical programming problem, more precisely, as a mixed integer linear program. In order to deal with the imprecise nature of the cost and performance figures, the linear program uses fuzzy instead of crisp numbers to represent the coefficients. The necessary fuzzy solver is constructed in such a way that it optimizes the most probable outcome, while minimizing the chances of finding a worse solution and maximizing the chances of a better one.

1.1.2 Hardware specification

After establishing the functional correctness, one has to create a high-level hardware description of the design for the parts of the design to be implemented in hardware. That is, one must model the system as concurrently executing processes. Also the notion of time has to be introduced. To represent the system at this level, one can use a hardware description language.

A hardware description language typically supports different description models for hardware. These commonly include a register transfer level (RTL) model and a behavioral model.

- *Register transfer level:* Formally speaking an RTL model describes hardware as state elements (registers) together with the combinatorial logic connecting them. The resulting description is cycle accurate, both at the interfaces and internally.
  RTL is best suited for a design if the design is best conceived by its structure. The structure is usually divided into a data path and a a finite state machine (FSM) as controller. Because RTL specifies a structural view of hardware, algorithms are difficult to express. However, RTL allows the designer complete control over the architecture, enabling very high-performance designs.
- *Behavioral model:* A behavioral model describes a design algorithmically. Although the internal behavior of the design is not described cycle accurate, the input–output behavior is.
  Behavioral languages commonly allow for higher levels of abstraction with respect to an RTL description. Hence, more concise descriptions are possible.

Today, Verilog and VHDL are the most popular HDLs. Numerous tools are available for simulation and synthesis, either directed to ASICs (application-specific integrated circuit) or FPGAs (field programmable gate array).

Difficulties arise when one tries to translate the executable specification into a hardware description, because of the differences in representation: sequential versus concurrent. Current solutions that automatically convert the executable specification to an HDL either extend the language with specific constructs for hardware description or target a predefined fixed architecture. Examples of these solutions are Ocapi [73] from IMEC, and A|RT Designer from Adelante Technologies (former Frontier Design), respectively. Serious drawbacks of these solutions are that the designer either must learn the specific extensions to standard C or C^{++}, or has to restrict himself to the particular architecture targeted. Other approaches, such as *SA-C* [68] and Handel-C [43] use variants of the C language, thus also forcing the designer to learn a new language.

As a final resort, the conversion can be done manually by rewriting the system model from scratch. The major drawback is the time-consuming and error-prone nature of this process. Furthermore, discrepancies between the executable specification and the hardware description are easily introduced. This significantly increases the risk that the final product does not meet its specifications. In that case, an expensive re-design would be necessary, introducing a delay in the time-to-market of the product.

Moreover, at this stage it is often still unclear which parts of the design should be implemented in software and which in hardware. Therefore, the time and effort invested in the conversion should be kept to a minimum, as some of this work might become obsolete in a later stage.

With the introduction of SystemC as a modeling language, an alternative trajectory has become possible. Now, the executable specification is translated into SystemC instead of into a conventional HDL. SystemC offers the same modeling properties as these, but additionally it provides a higher level of abstraction, effectively offering all features of the high-level programming language C^{++}. Of course not all constructs are synthesizable, but the translated code offers a good starting point for further refinement towards synthesis.

1.1.3 Design flow

In accordance with the arguments raised in the previous sections, we come to the design flow presented in figure 1.3. First, the algorithm to be implemented is described in a high-level programming language, in particular in C. Then, a partitioning has to be made, deciding which parts to implement in hardware and which parts to implement in software. The original C code can directly be used to describe the software, and additional steps are not needed. In contrast, the C code describing the functionality of the hardware parts has to be converted into an HDL, either an RTL or a behavioral description. For both descriptions, we choose to use SystemC.

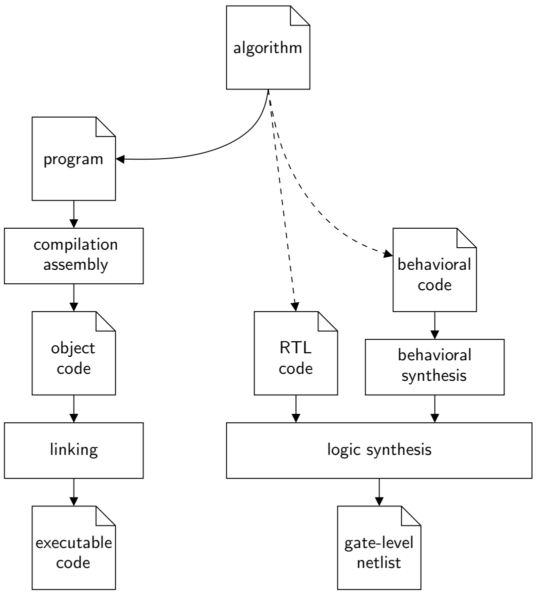


Figure 1.3. Design flow used throughout the book; the dashed arrows denote missing tools in the flow.

It is apparent that a number of steps cannot be done automatically yet. The first is the decision how to divide the design into hardware and software parts. The second is the automatic generation of the hardware descriptions, either using a structural (RTL) or a behavioral representation. The focus of this book is on partial automation of these steps, in order to quickly guide a designer towards a feasible solution that meets the requirements.

To test and verify the ideas and the design flow presented thus far, several design tools have been written to aid a chip designer in making architectural decisions at a high level. Because these decisions are made at a high level, they have a large impact on the performance and costs of the final design. In particular, attention is paid to the energy consumption of a device, and therefore, power is incorporated into the design flow as one of the primary design constraints.

1.2 Power as design constraint

In general, is defined as the conversion rate of energy. In the context of this book, a more restricted definition is used because it focuses on integrated circuits (ICs). In this restricted context, power is defined as the rate at which electrical energy is converted into some other form, usually heat (dissipation). Equivalently, it is defined as the rate at which an IC consumes energy from an electrical power source.

It is important to note that the terms "power" and "energy" are often used interchangeably, although they are not the same quantities. To be precise, if at time instance τ_0 a system has energy $E(\tau_0)$, and at τ_1 only $E(\tau_1)$ is left, then the power P is defined as the amount of energy $\Delta E = E(\tau_1) - E(\tau_1)$ consumed in that particular time interval $\Delta t = \tau_1 - \tau_0$

$$P = \frac{\Delta E}{\Delta t} = \frac{E(\tau_1) - E(\tau_0)}{\tau_1 - \tau_0}. \tag{1.1}$$

Taking the limit $\Delta t \to 0$ leads to the definition of the instantaneous power consumption $p(t)$ at time t

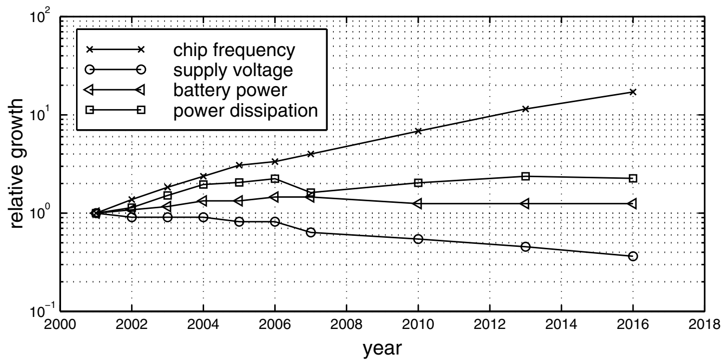


Figure 1.4. Predicted relative growth (data taken from [3]).

$$p(t) = \frac{\mathrm{d}}{\mathrm{d}t} E(t), \tag{1.2}$$

where $E(t)$ is the energy present in the system at time t.

Power has been a constraint in the design of analog systems for a long time, for instance in medical applications such as pacemakers and hearing aids. However, power was not an issue in the design of digital systems. Area and timing were by far the most important design constraints. However, recently power has become a more and more important constraint, for a number of reasons.

- *Rapidly growing complexity:* The current trend towards higher operating frequencies, higher integration densities and higher performance still continues in accordance with Moore's law [46].
- *Application in mobile devices:* The rapidly growing demand for mobile devices, such as personal digital assistants (PDAs) and cellular phones, increases the need for battery-powered devices.
- *Slowly increasing battery capacity:* Unfortunately, progress in battery technology cannot keep pace with advances in electronics circuits [58, 67], thereby increasing the gap between supply and demand.
- *Reduced lifetimes because of heating:* With the higher frequencies and the increasing integration densities, operating temperatures rise, thereby increasing the failure rate of electronic devices.[1]
- *Problematic power supply:* The required energy must be transported from an external source (for instance a battery) onto the chip via its pins and bonding pads. Because of the large currents involved, many pins are required in order to reduce the current per pin.

A number of these effects are depicted in figure 1.4. Plotted are the relative predicted increase of frequency, supply voltage, battery power and power dissipation. The data for the graphs originate from [3]. The increase in dissipation is apparent, as is the lagging of the battery power.

Depending on the application, power consumption varies from a marginal design constraint to the most important one. It can nearly be neglected in low-performance designs connected to a fixed power grid, while in high-performance battery-powered mobile devices, it is crucial. Furthermore,

[1] The failure rate can be estimated using the Arrhenius equation $\lambda = \lambda_0 e^{-\frac{E_a}{kT}}$, where λ is the failure rate, λ_0 a reference value, E_a the activation energy of the failure mechanism, and k Boltzmann's constant [40]. With an increase of 10℃ at an operating temperature of 70℃, the failure rate increases by a factor of 1.8–6.6 for typical values of the activation energy.

power consumption is a dominant factor for systems which are hard to access, such as satellites, and medical systems implanted in the human body.

Various aspects of the power consumption of a system have already been investigated. In [57], a classification is given of current power management solutions for operating systems. A framework is presented, which allows the implementation of cooperating power management policies at this level. Additionally, a novel scheduling algorithm is described that finds an energy-efficient setting for modern microprocessors exploiting voltage scaling. In [44], a classification is given of the sources of power dissipation in deep submicron CMOS logic. Reduction techniques are presented for each source of dissipation. Also a new technique to reduce the weak-inversion current is presented. This book focuses on the effects of design choices on, among others, the energy consumption of a design and how to make these effects quickly visible to the designer.

1.3 Application

To check whether the tools created operate properly, and whether the design flow is complete, we need a representative application to serve as a test case. For this, an OFDM transceiver is selected. The tools and design flow are applied to design the baseband processing of the OFDM modulator and demodulator used in the Ubiquitous Communications (UbiCom) program [37].

In the UbiCom scenario, a user is equipped with a wearable device consisting of a terminal and a see-through display. Text, graphics, images and video can be displayed before the user's eyes, properly aligned and consistent with the real world, thereby enhancing the user's view (so-called augmented reality).

To transfer all this data to and from the user, a high-speed wireless link is necessary using an OFDM transceiver. The design of the baseband processing of this transceiver is selected as a case to test the design flow and tools. It is chosen because the modulation and demodulation are data-dominated operations. Moreover, there are strongly conflicting design constraints, in particular, the required high processing speed versus the severe restrictions on the energy consumption, and hence it serves as a challenging test case.

1.4 Outline

This book describes several methods and tools to come to a consistent design flow, starting with a high-level specification in C. An essential part of this flow is a new tool to automate the hardware–software partitioning. The tool finds an optimal architecture to implement one or more algorithms. The resulting architecture is optimal with respect to the chip area, the execution time, or the energy consumption.

As the successful operation of the partitioning tool depends on the quick availability of accurate estimates for area, latency and energy consumption, we need an additional tool to convert the original C code into a hardware description language, in particular into SystemC. The automated conversion from C to SystemC enables a closed design flow, where tedious manual rewrites are no longer necessary. Furthermore, the automated conversion eliminates the chances of coding errors.

To test and verify the new design automation tools, we will apply the design tools to implement an OFDM transceiver. But first, we investigate several techniques to reduce the energy consumption of the OFDM transceiver beforehand.

In chapter 2, we give an overview of existing estimation techniques for area usage, delay and energy consumption. Then, both the partitioning tool and the conversion tool are described in chapter 3. In chapter 4, we describe the novel energy reduction techniques for OFDM, and in chapter 5, we discuss the implementation of the OFDM transceiver using the presented methods and tools. Finally, in chapter 6, we conclude.

2 Design trade-offs

2.1 Introduction

To prevent expensive re-designs in the final part of the development of a system/product, the designer needs to obtain feedback about the performance and costs of the current design as early as possible in the design flow. Performance feedback is already incorporated in modern design flows, applying formal analysis, high-level functional simulations and lower-level behavioral and structural simulations.

Cost feedback, on the other hand, is only available at the final stages of the design flow, e.g. only when a net list is available. Yet, choices made at a high level may severely constrain the final product. In particular, in the design of a power-limited system, power estimation early in the design cycle is crucial.

This chapter gives an overview of cost estimation techniques, more specifically techniques for determining area, delay and energy consumption. Of course, these three are not independent and therefore their dependencies, i.e., the trade-offs, are discussed in the final section of this chapter.

2.2 Area estimation

In the early days of VLSI design, the area consumption was the decisive limit on what could be integrated on a chip. Over the years, as chip area and transistor densities steadily grew, it remained one of the primary design constraints, because of its impact on processing costs and yield. Therefore, chips produced in large quantities are designed to be as small as possible.

The total area used for a circuit can be divided into two main categories, namely the area used by the gates, and the area due to the interconnect. As the interconnect can be positioned on top of the gates, these two overlap and the total area is not merely their sum. However, not all interconnects can be placed over the gates, so some additional area is needed to accommodate the interconnect.

2.2.1 Gate area estimation

Several techniques exist for estimating the area. Depending on the level of detail available, different estimation methods and techniques are used. These include

- transistor counting,
- complexity-based techniques, and
- Rent's rule.

Each technique is discussed in more detail in the following sections:

2.2.1.1 Transistor count

If a net list is available, the area of the design can be estimated by counting the number of transistors used and by multiplying the result with the area of a single transistor. For every library cell, the number of transistors used is available, thus calculating the total area estimate is simply a summing operation.

Obviously, the area due to interconnect is not accounted for. Moreover, ideal placement is assumed, i.e., gaps between cells are not taken into account. Therefore, the transistor count estimate gives a lower bound of the total area.

If a net list is not available, a synthesis step can be made to obtain a preliminary estimate. Depending on the complexity of the design this synthesis can be expensive in terms of computation time. Therefore, for quick iterations when exploring the design space, this approach is not very attractive.

2.2.1.2 Complexity-based area estimation

With complexity-based estimation techniques, it is assumed that the area needed to implement a certain Boolean function depends on its area complexity, i.e., it is assumed that the area complexity of a Boolean function corresponds to the minimum number of gates required to implement the function and thus can serve as a valid estimate of the gate count. Different measures and approaches exist to express the area complexity in terms of known properties of the function to be implemented. Of course, once the area for the combinatorial part of a design has been estimated, the area needed for the registers and other memory elements should also be accounted for.

In [16], the authors use the number of literals $L(f)$ of a Boolean function f as an area complexity measure. An exponential relation with the function's entropy is found via the so-called computational work of a function. The computational work is a measure for the computing power of a function.

$$L(f) \propto W(f) \tag{2.1}$$

The computational work $W(f)$ of an n-input combinatorial function f with m outputs is given by [27]

$$W(f) = 2^n H(f). \tag{2.2}$$

Here, $H(f)$ is the entropy of the function and it is defined as

$$H(f) = \sum_{i=1}^{2^m} P_i \log_2 \frac{1}{P_i}. \tag{2.3}$$

where P_i is the probability of the ith output vector y_i, given by

$$P_i = \frac{n_{y_i}}{2^n} \tag{2.4}$$

with n_{y_i} the number of input vectors that have y_i as result.

The authors show that this model is suitable for estimating the area for randomly generated functions. However, it is argued in [49] that this measure for the area complexity does not hold for typical VLSI circuits. Therefore, the authors suggest an extension to the model. They argue that the model should be extended by adding information about the structure of the Boolean space of the function, to accommodate typical VLSI circuits.

In order to capture the characteristic structure of a Boolean function, the authors introduce the so-called linear measure $L(f)$ for a single-output Boolean function f. It is defined as

$$L(f) = L_{\text{on}}(f) + L_{\text{off}}(f). \tag{2.5}$$

Here, $L_{\text{on}}(f)$ and $L_{\text{off}}(f)$ represent the linear measure of the on-set and off-set of the function f, respectively.

The linear measure for the on-set[1] of an n-input, single-output Boolean function f is calculated as the weighted sum of the distinct sizes of the prime implicants in a minimal cover of the on-set [45] of f

$$L_{\mathrm{on}}(f) = \sum_{i=1}^{N} c_i \cdot p_i. \tag{2.6}$$

Here, N is the number of distinct sizes of prime implicants in the cover, $\{c_i\}$ is the ordered set of distinct sizes of these implicants for $i = 1 \dots N$ such that $c_1 < c_2 < \dots < c_N$, and p_i is the weight factor on the prime implicants of size c_i.

The weighing factors are defined as follows

$$p_i = \begin{cases} P(f_i) - P(f_{i-1}), & \text{if } i > 1, \\ P(f_i), & \text{if } i = 1, \end{cases} \tag{2.7}$$

where $P(f_i)$ is the probability that $f_i = 1$, assuming an equal probability for all points in the Boolean space of f_i. Sub-function f_i of the original function f is defined such that its on-set solely consists of the prime implicants of the on-set of f with sizes $c_1, c_2, \dots c_i$. With these definitions we get

$$\sum_{i=1}^{N} p_i = P(f). \tag{2.8}$$

Thus $P(f_i)$ is equivalent to the fraction of the Boolean space of f covered by sub-function f_i.

In order to estimate the area of a Boolean function, the authors use almost-exponential relations to translate the linear complexity measure into an area estimate. These relations are derived by randomly generating Boolean functions for different entropies $H(f)$, calculating the linear measure, and finally measuring the gate count after synthesis.

In order to apply this technique to multi-output Boolean functions, a multi-output function is transformed into an equivalent single-output function. The transformation boils down to adding a multiplexer to the circuit and calculating the linear measure for the result. Finally, the additional area of the multiplexer is compensated for.

2.2.1.3 Rent's rule

An empirical method to predict the area of a circuit is based on the observation made by Rent that the average number of terminals T per module is exponentially dependent on the number of gates G used by the module [41]

$$T = t \cdot G^p. \tag{2.9}$$

Here, the Rent coefficient t is interpreted as the average number of terminals required by a single gate and p is known as the Rent exponent. This relationship is commonly known as Rent's rule.

Conversely, the area can be expressed in the number of pins T

$$G = \left(\frac{T}{t}\right)^{\frac{1}{p}}. \tag{2.10}$$

Thus, if the technology and design parameters t and p are known, an estimate of the area can be made based on the number of pins of the circuit.

If the Rent coefficient and exponent are not known beforehand, they have to be estimated. As Rent's rule is an exponential relationship, an error in the prediction of the Rent exponent can have large impact on quantities derived using Rent's rule. It is important to note that the Rent coefficient and exponent cannot be regarded as independent variables [17].

[1] The linear measure for the off-set is calculated similarly.

Current methods to determine the Rent coefficient and exponent are based on two approaches. The first is to use statistical estimates from previous designs that are similar (various estimates can be found in literature [5]). The second is to estimate the parameters from the properties of the net list. This again requires a synthesis step. However, because of the self-similarity within the design, the same properties apply throughout the whole design. Therefore, only small sub-circuits have to be synthesized to obtain a representative net list.

2.2.2 Interconnect estimation

The estimation of interconnect characteristics is important as it influences the three main design constraints directly.

- *Area:* Obviously, the wires have to be placed somewhere on the chip. Especially structures like busses require a large area.
- *Delay:* The propagation delay is directly dependent on the length of the wires (see also section 2.3.1.2).
- *Power:* The parasitic capacitance and resistance of the wires are dependent on the length of the wires, thus longer wires require more energy to transport a signal value.

Estimation methods can be divided into a priori and a posteriori methods. As a posteriori techniques usually require a full placement step, this method is not feasible for quick iterations. Therefore, in the remainder of this section we will focus on a priori estimation methods only.

An a priori estimation technique for interconnects is the method introduced by Donath [19]. The author recursively applies Rent's rule to come up with estimates for the average lengths $\bar{l}_k$ and the expected number of interconnections $\overline{n}_k$ at each level k of the design hierarchy. Using these, the average length $\overline{L}$ of the interconnect is calculated:

$$\overline{L} = \frac{\sum_{k=0}^{K-1} \overline{n}_k \bar{l}_k}{\sum_{k=0}^{K-1} \overline{n}_k}, \tag{2.11}$$

where K is the top level in the hierarchy. Now, with

$$\overline{n}_k = \alpha C 4^K \left(1 - 4^{p-1}\right) 4^{k(p-1)}, \tag{2.12}$$

where C is the total number of gates in the design and p the Rent exponent, and with

$$\bar{l}_k = \frac{2}{9}\left(7\lambda - \frac{1}{\lambda}\right), \tag{2.13}$$

where $\lambda = 2^k$, we have all the information we need to calculate the average interconnection length $\overline{L}$. As the width and the minimum separation distance of the wiring is usually known, the interconnect area can easily be calculated.

In [78], Donath's model is refined to accommodate additional placement information. It is assumed that an optimal placement procedure will preferably place gates that are interconnected close together. This results in a lower estimate of the lengths of the interconnections. Therefore, Donath's model can be used as an upper bound estimate, whereas the refined model gives more accurate predictions.

2.3 Delay estimation

In many systems to be designed, timing is a critical design constraint. This is especially true for real-time systems, i.e., systems that should generate a response within a specified maximum delay after applying a stimulus. In the remaining systems, the temporal behavior may not be critical, but it still is an important performance measure. Therefore, delay estimation is a crucial aspect of a design trajectory.

In this section we focus on delay estimation of synchronous circuits. The delay or execution time of a synchronous circuit can be expressed by two figures of merit, namely the cycle time and the latency of the circuit. The cycle time is the period of the fastest clock that can be applied to the circuit, without introducing errors. The latency is the time required to execute operations, expressed in terms of clock cycles. Herewith the product of cycle time and latency gives the overall execution delay of the circuit.

Delay estimation techniques can be subdivided into two main categories: delay estimation for individual resources, and system delay estimation. The methods in the first category try to estimate the delay of relatively small building blocks. Techniques in the second category use these results to estimate the delay of a complete system consisting of several of these building blocks.

2.3.1 Resource delay estimation

Various methods exist for estimating the delay of individual resources. They are based on different principles.

- *Measurement:* An evident way to obtain a delay estimate is to measure the timing characteristic of a circuit once it has been built. Though accurate, the method is only applicable for relatively small designs, because of the time needed to physically build the circuit and because of the associated costs.
- *Calculus:* A second method to estimate the delay of a circuit is to calculate the waveforms on all nodes of the circuit either analytically or numerically. These waveforms are then used to calculate the timing behavior of the circuit.
  The applicability of this method is heavily influenced by the level of detail required. For instance, in a synchronous circuit, the exact calculation of the minimum cycle time is more difficult than the calculation of the latency of a circuit once the cycle time is known.
- *Simulation:* Simulation-based methods are used at different levels of detail for delay estimation. The underlying models determine the accuracy of the estimation, and range from detailed transistor models to behavioral gate models.

All three methods assume that the implementation of the resource is already known. Thus, a net list or a layout specification of the resource must be available. Of course, for measuring the delay of circuit, the circuit has to be realized already.

Calculus and simulation-based methods require knowledge and subsequent modeling of the delay mechanisms. In digital synchronous circuits, two sources of delay can be identified: gate delay, and propagation delay. The former is the time it takes a gate to switch from one level to another. The latter is the time needed to transport a signal over a certain distance on a chip. Each of these is discussed in more detail in the following paragraphs.

2.3.1.1 Gate delay

As CMOS FETs have a limited current driving capability, charging or discharging of the load capacitance of a gate causes some delay larger than zero. In this section, a lower bound for this delay is presented.

We consider a charged load capacitance which is discharged by a conducting NMOS transistor (see figure 2.1). The amount of electrical charge $Q_\triangle$ that has accumulated on the capacitor equals

$$Q_\triangle = C_L \cdot V_\triangle, \tag{2.14}$$

where C_L is the load capacitance and $V_\triangle$ is the voltage swing.

The total charge transported through the transistor is given by the integral of the transistor current $i_n(t)$. If the capacitor is to be fully discharged, this value should be equal to the accumulated charge $Q_\triangle$

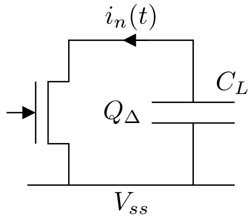


Figure 2.1. Discharging a load capacitance through an NMOS transistor.

$$Q_\Delta = \int_{\tau_0}^{\tau_1} i_n(t)\ \mathrm{d}t. \tag{2.15}$$

To calculate a lower bound on the delay of the gate, we assume that the transistor conducts the maximum current given by the saturation current $I_{D\mathrm{sat}}$. Then substituting $I_{D\mathrm{sat}}$ for $i_n(t)$ in equation 2.15 gives

$$Q_\Delta = \int_{\tau_0}^{\tau_1} I_{D\mathrm{sat}}\ \mathrm{d}t = (\tau_1 - \tau_0) \cdot I_{D\mathrm{sat}}. \tag{2.16}$$

Substitution of equation 2.14 gives the lower bound of the delay $\tau_\Delta = \tau_1 - \tau_0$

$$\tau_\Delta = \frac{C_\mathrm{L} \cdot V_\Delta}{I_{D\mathrm{sat}}}. \tag{2.17}$$

To further explore the delay bound, we need an expression to calculate the saturation current. A widely used transistor model is the long-channel approximation. In this model, the saturation current has a quadratic relation with the gate voltage.

However, the long-channel approximation fails to accurately model recent short-channel FETs, primarily because it does not incorporate carrier velocity saturation effects. To overcome the shortcomings of this model, an alpha-power law model has been developed [72], which models the velocity saturation effects by a velocity saturation index α. The value of the index varies from 2 for long-channel transistors to ultimately 1 (linear relation) when the speed of the carriers is always at saturation speed.

In the alpha-power law model, the saturation current has the following relation with the gate voltage V_G

$$I_{D\mathrm{sat}} = \beta \left(V_\mathrm{G} - V_\mathrm{T}\right)^\alpha, \tag{2.18}$$

where V_T is the threshold voltage of the transistor and β a transistor-dependent parameter.

Substitution of equation 2.18 in equation 2.17 gives

$$\tau_\Delta = \frac{C_\mathrm{L} \cdot V_\Delta}{\beta \left(V_\mathrm{G} - V_\mathrm{T}\right)^\alpha} \approx \frac{C_\mathrm{L}}{\beta \cdot \gamma^\alpha {V_\mathrm{dd}}^{\alpha-1}}. \tag{2.19}$$

Here we used $V_\Delta \approx V_\mathrm{dd}$ and $V_\mathrm{G} - V_\mathrm{T} \approx \gamma V_\mathrm{dd}$. Thus the gate delay is inversely proportional with the supply voltage

$$\tau_\Delta \propto \frac{1}{{V_\mathrm{dd}}^{\alpha-1}}. \tag{2.20}$$

Apparently the gate delay can be lowered by increasing the supply voltage, i.e., by increasing the driving power. However, this has a serious impact on the power consumption of a circuit (see section 2.4.3.1).

2.3.1.2 Propagation delay

With ever decreasing feature sizes and ever higher switching speeds, the propagation delay becomes more and more important. To model an interconnect, resistance–capacitance (RC) trees are commonly used. A popular and widely used RC delay metric is the Elmore delay [21] because of its simple closed-form expression. However, the Elmore delay metric is derived for point-to-point connections, and lacks consistent accuracy for RC trees, especially at the near end, because it ignores resistive shielding.

It has been shown that the Elmore delay corresponds to the first moment of the impulse response of an RC tree [71]. A natural step is to include higher-order moments to increase the accuracy. A simple and accurate metric is the so-called D2M metric [4]. It is an empirical metric based on the first two moments of the impulse response of the RC tree.

Given an RC tree with nodes $\{v_0, \ldots, v_N\}$, where v_0 is the source of the tree, let C_i be the capacitance at node v_i for $0 < i \leq N$. If $p(v_i)$ is the predecessor of node v_i then R_i is defined as the resistance between nodes $p(v_i)$ and v_i. Now define R_{ki} as the total resistance of the portion of the path from v_0 to v_i that overlaps with the path from v_0 to v_k.

For $j \geq 1$, the jth moment m_j^i of the impulse response for node v_i is recursively expressed as

$$m_j^i = -\sum_{k=1}^{N} R_{ki} C_k m_{j-1}^k. \tag{2.21}$$

For $j = 0$ we have $m_0^i = 1$ for all nodes.

Now, the D2M delay metric is defined as

$$D_{D2M} = \frac{{m_1}^2}{\sqrt{m_2}} \ln 2. \tag{2.22}$$

Here, m_j denotes the jth moment for a generic node.

2.3.2 System level delay estimation

As soon as the cycle time and latencies of the resources have been estimated, an estimate can be made of the delay of a system composed of those resources. In most cases, the execution order of the resources is not yet known and therefore a schedule has to be made.

Apart from the resource characteristics, the following aspects of the system should be taken into account while defining a schedule.

- *Data storage:* In synchronous circuits, data should be stored in some form of memory when it has to go from one functional resource to another.
- *Wiring:* To transport data from one resource to another, wires are needed. Naturally these cause some delay.
- *Steering logic:* To properly guide the data from one resource to another some steering logic is needed. Usually it consists of some form of multiplexers.
- *Control unit:* In order to control the resources and the data transports, one needs a control unit. Depending on the type of circuit (data or control dominated) this unit may significantly contribute to the total delay of the system.

In this book, we consider data-dominated systems, i.e., we assume that the control part of the system can be neglected with respect to the data part.

Once the schedule is known, the latency of the circuit is known. Additionally, if the cycle time is known, an estimate is available for the total delay of a system. Of course different scheduling strategies evoke different delays.

The scheduling problem can be formulated as follows. Given a set of tasks and resources, we have to find an ordering of tasks and assignment to resources while satisfying given constraints. Different constraints can be applied separately or simultaneously. Constraints can be anything ranging from minimizing total latency, restricting the number of resources, to hard deadlines for certain tasks. Depending on the constraints, different scheduling strategies are viable. Three scheduling strategies are covered in more detail in the next sections.

2.3.2.1 Unconstrained scheduling

In unconstrained scheduling we want to find the minimum latency schedule. There are no constraints, neither on the number of resources nor on time limits to be met. Unconstrained minimum latency scheduling can be solved in polynomial time by applying the ASAP (as soon as possible) scheduling algorithm.

2.3.2.2 Time-constrained scheduling

In time-constrained scheduling, only time constraints are set. The number of resources is not limited. Time-constrained scheduling comes in two flavors: latency-constrained scheduling and scheduling under timing constraints. Also a combination of the two is possible.

In latency-constrained scheduling, a global maximum latency constraint is specified. This scheduling problem can be solved by running the ASAP algorithm and by verifying that its solution meets the latency constraint.

When scheduling under timing constraints, one can specify additional constraints for the individual resources. These timing constraints can be either absolute or relative. Scheduling under absolute timing constraints can be solved using the Bellman–Ford algorithm.

2.3.2.3 Resource-constrained scheduling

Resource-constrained scheduling arises from resource-limited circuits, typically incurred by constraints on the total chip area. The number of resources necessary to implement a circuit largely determines the area used by the circuit. Therefore, resource-constrained scheduling can be used to find latency versus area trade-offs.

Resource-constrained scheduling is intractable. However, several exact algorithms and heuristics exist to come up with a, possibly approximate, solution in reasonable time. These include integer linear programming algorithms, list scheduling and force directed scheduling.

2.4 Power estimation

In order to properly design for low power, one needs feedback in the form of power estimation techniques. Several power estimation techniques exist for VLSI and can be applied at different levels in the design flow. The following criteria should be taken into account when comparing estimation techniques.

- *Efficiency:* The efficiency of a power estimator indicates whether the estimator can deal with large circuits in reasonable time. Especially at the beginning of the design of a system, feedback must be provided quickly, to aid early exploration of the design space.
- *Accuracy:* The accuracy of a power estimator is determined by the difference between the estimated and the actual (after implementation) power usage. If absolute accuracy is not required, the estimator should at least properly estimate the relative power consumption of design alternatives.
- *Uncertainty:* Especially in an early stage of the design trajectory, not everything is yet known about the design and its implementation. Therefore, a power estimator should be able to cope with these uncertainties and unknowns.

Together these three criteria determine the effectiveness of a power estimator in a particular stage of the design trajectory.

A number of approaches is available to estimate the power consumption of a design [15, 39, 49, 64].

- *Measurement:* An obvious method to estimate the power usage of a circuit is to build it and to measure the current drawn from the power supply. However, this technique is not applicable for other designs than very small ones, as building larger designs rapidly becomes too time consuming and too expensive.
- *Calculus:* A straightforward way to estimate the power dissipation of a circuit is to calculate all currents and voltages of dissipating elements, either analytically or numerically. The product of these gives the instantaneous power of all these elements. Finally, summing the intermediates gives the power consumption of the whole circuit.

  Calculating the exact power consumption is only feasible for relatively small circuits. For larger circuits it is extremely complex, and for some circuits even not possible to find analytical expressions of the currents and voltages and numerical calculation is usually too time consuming.
- *Probabilistic methods:* If the probabilities of the input signals of a network are known, the probabilities of all signals on all nodes of the network can be calculated. Using these probabilities, activity levels can be estimated and based on these the power usage of the network can be estimated with a charged capacitance model [70].

  A drawback of this method is that the inputs must be independent. If they are not, calculating the signal probabilities becomes very complex.
- *Statistical methods:* Another approach similar to the probabilistic methods is to apply a large number of random inputs to the circuit and to statistically find the activities of the nodes. Again a switched capacitance model can be used to estimate the power usage of the circuit.
- *Simulation-based methods:* Finally, a large category of power estimators is based on simulation. These simulate a design and use some power model for its components to estimate the power consumption. Simulation techniques can be applied at various levels; they range from methods using detailed transistor models to techniques estimating the power usage of large sub-systems such as full-blown Fourier transforms.

Except for the technique of direct measurement of the power consumption, all techniques rely on models of the power dissipation in an IC. To get a better understanding of power dissipation in CMOS circuits, we take a look at its origins. Power dissipation in CMOS ICs can be divided into two main categories [62]: static power dissipation and dynamic power dissipation. Both categories will be discussed shortly.

Moreover, several of these methods are based on estimating the activity of the internal signal nodes of the network. Therefore activity-based power estimation will be discussed in more detail in section 2.4.3.

2.4.1 Static power dissipation

Assuming ideal transistors, a CMOS circuit does not dissipate any power when it is not switching. However, this is not a realistic assumption: a certain amount of power is always lost due to a number of mechanisms such as leakage currents and substrate injection currents. These currents cause a static power dissipation component in CMOS circuits.

Static power dissipation can be minimized by choosing a proper technology. Once the designer has chosen one, there is little else he can do to influence this type of power dissipation. Therefore, we will focus on dynamic power dissipation.

2.4.2 Dynamic power dissipation

For CMOS circuits, the dynamic power dissipation can be subdivided into two types.

- *Short-circuit current:* During a switching transient, for a short time, both the NMOS pull-down network and the PMOS pull-up network of the gate will conduct simultaneously. During this time a short circuit is formed between the voltage supply and ground, resulting in a power loss. This loss can be minimized by carefully designing the devices and the gates.

- *Capacitance charging:* Every time when switching occurs, parasitic capacitances are either charged or discharged. The discharging process does not draw any current from the power supply. However, for the charging of the capacitors, energy is needed, which is drawn from the power supply. Hence, power is dissipated when capacitances are charged. The amount of dissipated power depends on the size of the parasitic capacitances and the voltage swing.

Capacitance charging is the dominating factor in power dissipation in CMOS circuits. Furthermore, at the high level of design we are concerned with, we have little influence (except from choosing a proper technology) on the short-circuit currents of individual transistors. Therefore, in the remaining parts of this section, we will focus on power dissipation due to capacitance charging.

2.4.3 Activity-based power estimation

Activity-based power estimation is founded on the assumption that dynamic power dissipation is the dominating factor in the total power dissipation of a circuit. Therefore, it estimates the amount of capacitance being charged every time a node in the network switches. The energy needed to charge these capacitances gives an estimate of the dissipation in the circuit.

2.4.3.1 Capacitance charging

The amount of capacitance charged during a transition must be translated into a power figure. Therefore, we calculate the energy dissipation due to the charging of capacitances in complementary MOS gates.

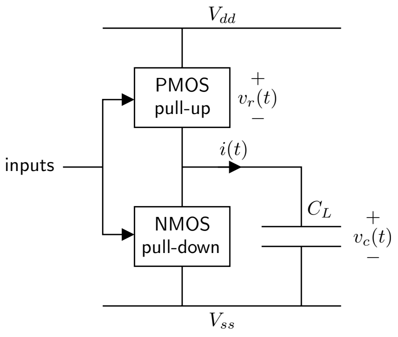


Figure 2.2. Capacitance (dis)charging in CMOS gates.

The current through the capacitive load $i_\mathrm{c}(t)$ of a CMOS gate is given by (see figure 2.2)

$$i(t) = C_\mathrm{L}\frac{\mathrm{d}v_\mathrm{c}(t)}{\mathrm{d}t}, \tag{2.23}$$

where $v_\mathrm{c}(t)$ is the voltage over the load capacitor and C_L the load capacitance.[2]

Switching from a low level at time τ_0 to a high level at τ_1 results in charging the load capacitance with energy E_C

$$E_\mathrm{C} = \int_{\tau_0}^{\tau_1} i(t)v_\mathrm{c}(t)\mathrm{d}t. \tag{2.24}$$

[2] The load capacitance C_L is assumed to be independent of the load voltage $v_\mathrm{c}(t)$.

Then substitution of equation 2.23 gives

$$\begin{aligned} E_\mathrm{C} &= C_\mathrm{L} \int_{\tau_0}^{\tau_1} \frac{\mathrm{d}v_\mathrm{c}(t)}{\mathrm{d}t} v_\mathrm{c}(t)\mathrm{d}t \\ &= \frac{1}{2} C_\mathrm{L} \left(v_\mathrm{c}(\tau_1)^2 - v_\mathrm{c}(\tau_0)^2\right). \end{aligned} \tag{2.25}$$

The energy stored in the load capacitance is lost at the consecutive discharging (switching from high to low) of the gate.

In addition to the energy used to charge the capacitive load, the PMOS circuit dissipates energy E_R in the form of heat

$$E_\mathrm{R} = \int_{\tau_0}^{\tau_1} i(t) v_\mathrm{r}(t)\mathrm{d}t,$$

where $v_\mathrm{r}(t)$ is the voltage drop over the PMOS circuit. Substituting equation 2.23 and using $v_\mathrm{r}(t) = V_\mathrm{dd} - v_\mathrm{c}(t)$ leads to

$$\begin{aligned} E_\mathrm{R} &= \int_{\tau_0}^{\tau_1} i(t) \left(V_\mathrm{dd} - v_\mathrm{c}(t)\right) \mathrm{d}t \\ &= C_\mathrm{L} V_\mathrm{dd} \left(v_\mathrm{c}(\tau_1) - v_\mathrm{c}(\tau_0)\right) - \frac{1}{2} C_\mathrm{L} \left(v_\mathrm{c}(\tau_1)^2 - v_\mathrm{c}(\tau_0)^2\right). \end{aligned} \tag{2.26}$$

Using equations 2.25 and 2.26, we obtain the total energy E lost during a low to high transition, given by

$$E = E_\mathrm{C} + E_\mathrm{R} = C_\mathrm{L} V_\mathrm{dd} V_\Delta, \tag{2.27}$$

where the voltage swing V_Δ of the gate is given by

$$V_\Delta = v_\mathrm{c}(\tau_1) - v_\mathrm{c}(\tau_0). \tag{2.28}$$

Assuming the voltage swing approximately equals the supply voltage, $V_\Delta \approx V_\mathrm{dd}$, leads to

$$E \approx C_\mathrm{L} {V_\mathrm{dd}}^2. \tag{2.29}$$

It is important to note that in this approximation the dissipated energy only depends on the capacitive load and the supply voltage. Gate-specific and transistor-specific parameters, such as the channel width or length of a FET do not influence the energy dissipation.

If we use the exact equation 2.27 instead, gate and device parameters do influence the energy dissipation as they determine the voltage swing V_Δ of the gate.

2.4.3.2 Load capacitance

To estimate the load capacitance C_L of a gate, we use the following model. If an output of a gate switches from a low to a high level, several capacitances have to be charged, namely:

- The output capacitance of the gate
- The input capacitances of the connected gates
- The wire capacitances of the interconnect

An example is shown in figure 2.3.

The total capacitance C_L charged by the gate is given by

$$C_\mathrm{L} = C_\mathrm{out}^O + C_\mathrm{wire} + C_\mathrm{in}, \tag{2.30}$$

where C_out^O is the output capacitance of the driving gate, C_wire the capacitance due to the interconnect and C_in the total input capacitance of the gates being driven.

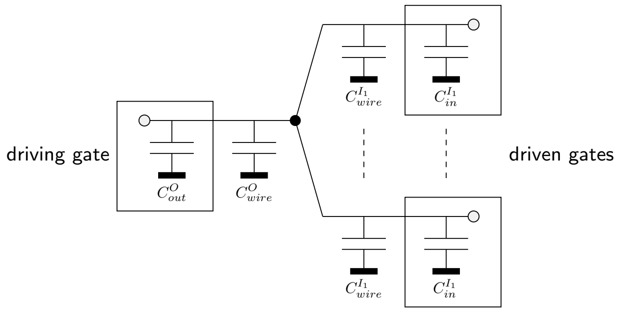


Figure 2.3. Example of the switched capacitance model.

The interconnect capacitance is modeled by an output wire capacitance C^O_{wire} and several input wire capacitances $C^{I_i}_{\text{wire}}$. The total wire capacitance is given by

$$C_{\text{wire}} = C^O_{\text{wire}} + \sum_{i=1}^{N} C^{I_i}_{\text{wire}}. \tag{2.31}$$

The input capacitance of the gates being driven is given by the sum of the input capacitances of every gate $C^{I_i}_{\text{in}}$. Then the total input capacitance is

$$C_{\text{in}} = \sum_{i=1}^{N} C^{I_i}_{\text{in}}. \tag{2.32}$$

Substituting equations 2.31 and 2.32 into equation 2.30 leads to

$$C_L = C^O_{\text{out}} + C^O_{\text{wire}} + \sum_{i=1}^{N} \left(C^{I_i}_{\text{wire}} + C^{I_i}_{\text{in}} \right). \tag{2.33}$$

Here the total switched capacitance is clearly divided into capacitances related to either the driving or the driven side. Therefore, the total switched capacitance of a circuit can be calculated by summing the switched input, output and wire capacitances per gate.

2.4.3.3 Circuit level power estimation

The total energy dissipated in a large integrated circuit is calculated using

$$E = \sum_{i=1}^{N} a_i C_{\text{L}i} {V_{\text{dd}}}^2 \tag{2.34}$$

where N is the number of nodes in the circuit, a_i the number of times the ith node switches from a low to a high level, and $C_{\text{L}i}$ is the load capacitance of node i.

The load capacitance is approximated by the average load capacitance given by

$$C_{Li} \approx \overline{C}_L = \frac{C}{N} \tag{2.35}$$

where C represents the total capacitance. Substituting equation 2.35 in equation 2.34 gives

$$E \approx \frac{1}{N} \sum_{i=1}^{N} a_i C {V_{\text{dd}}}^2 = \alpha C {V_{\text{dd}}}^2. \tag{2.36}$$

Here α is the average switching activity of the circuit

$$\alpha = \frac{1}{N}\sum_{i=1}^{N} a_i. \tag{2.37}$$

It should be noted that equation 2.36 is a rough estimate of the total energy used by the circuit, based on global averages.

2.4.4 System level power estimation

A major obstacle in estimating the energy or power consumption at a high level is that in general, detailed structural information of the design is still lacking. This kind of information is usually only available after scheduling, and the final net list only after a full synthesis step. Although they are more accurate, it takes too much time to compute these estimates so they cannot be used for a quick estimate in an iterative process.

If however, a design is constructed using fixed sub-circuits, the time needed to estimate the cost figures of a sub-circuit once may be acceptable with respect to the overall design time. If the sub-circuits have been used in a previous design, these numbers might even be available beforehand.

Nevertheless, if only the functional description is available, simulation is currently the most obvious way to quickly estimate the power consumption of a design. It typically involves estimating the average activity α and the capacitive load C_L of the circuit.

2.5 Area, delay, power trade-offs

Obviously, the area, delay and power of a circuit are strongly interdependent. As a result, design constraints are often in conflict and trade-offs have to be made. Of course, trade-offs can be made between every combination of two or more constraints. This results in four possible trade-off combinations:

- area – delay,
- delay – power,
- power – area, and
- area – delay – power.

The third option, power versus area, is usually considered the least interesting and also mostly covered in the last combination. Therefore, only the other three are discussed in the next sections.

2.5.1 Area versus delay

For many years area and delay were the primary cost and performance measures of a design. With the recent rise of power usage as an important design issue, the focus has moved a little. However, trade-offs between area and delay are still of prime importance in most designs.

This section starts with a discussion on circuit techniques. Here, a circuit is assumed to be given and techniques are discussed to exchange area for delay and vice versa. This pragmatic approach leads to the more fundamental question of what these techniques can achieve theoretically. Thus, in the remainder of this section some theoretical bounds on area and time (delay), the so-called AT bounds, are discussed.

2.5.1.1 Circuit techniques

Circuit techniques are used to modify the area and delay of a given circuit, without changing the circuit behavior. Two alternatives are discussed: re-timing and pipelining.

Re-timing

A synchronous network can be modified by shifting, adding or deleting registers. This process is called re-timing. The logical behavior of the circuit does not change during or after re-timing.

Re-timing can be applied to achieve different optimization goals.

- *Cycle time minimization:* In cycle time minimization, the delay of the longest path (not interrupted by registers) in the circuit is reduced. This is done by shifting the registers such that the longest path is shortened or replaced by another path, hereby reducing the path delay. Thus after re-timing it is possible to reduce the cycle time to match the newly found longest path delay.
- *Area minimization:* As adding or removing registers respectively increases or decreases the area of the circuit, re-timing can also be used to minimize the area.

Obviously, cycle time and area minimization are not independent. Thus, joint optimization requires another approach. Besides re-timing, combined area and delay optimization requires combinatorial optimization. In this scheme, combinatorial optimization is used to remove combinatorial bottlenecks in the circuit. Afterwards re-timing is applied to meet the cycle time requirements.

Pipelining

Pipelining is a technique used to increase the throughput of a circuit. Throughput is defined as the number of data samples processed per clock cycle. Strictly speaking, pipelining does not reduce the delay of a circuit. However, given an increased throughput after pipelining, the cycle time of the circuit can be lowered to get the same overall processing rate (data samples per time unit).

Of course, pipelining is not for free and usually increases the area of the circuit because of the extra registers and control logic needed.

2.5.1.2 Area-time (AT) bounds

Given a certain implementation of an algorithm, the area and the execution delay are known. When another implementation is used with a different area and delay, the question arises how the area and delay of the first implementation relate to the second and vice versa.

Lower bound arguments for three different relations are summarized in [82]. These are based on three different observations regarding the information processing capabilities of a circuit with a certain area A and a certain execution delay T.

- *Memory limited:* A circuit can only remember a limited number of bits from one time step to the next. This number of bits is proportional to the area. Therefore,

$$A = \text{constant}. \tag{2.38}$$

  Another interpretation of this rule is that the area of the registers in the data path of a circuit does not depend on the delay of the circuit.
- *IO limited:* A circuit can only process and generate a limited number of inputs and outputs. This number is proportional to the product of the area and the delay. Thus,

$$\text{AT} = \text{constant}. \tag{2.39}$$

  For instance to read n-input ports, one needs n-input terminals for a single time unit, or only one input port for n time units.
- *Information exchange limited:* The amount of information that can flow from one part of the area to the other in a single time unit is limited to the square root of the area. Thus, the total information flow is limited to the product of the square root of the area and the delay. Hence,

$$\text{AT}^2 = \text{constant}. \tag{2.40}$$

An equivalent interpretation is that the number of interconnections (wires) is limited to the square root of the area (i.e., the largest possible value for the shortest side of a rectangular area).

It is assumed that the circuits considered contain a large number of operations. Furthermore it is assumed that numerous alternative implementations can be found.

In [56], these bounds were experimentally verified. It appeared that the first two bounds can be used as good approximations. The third, however, was not validated. Still, it was noted that almost all experiments showed the following relationship

$$\mathrm{AT}^r = \text{constant}, \tag{2.41}$$

where r varied between 1.2 and 1.5.

2.5.2 Delay versus power

The average power used by the circuit is defined by

$$P = \frac{E}{T}, \tag{2.42}$$

where E is the total energy consumed by a circuit in T time. The total energy can be approximated by (see equation 2.36)

$$E = \alpha C {V_{\mathrm{dd}}}^2, \tag{2.43}$$

where α is a measure of the the average switching activity of the circuit, C_{av} the average total switched capacitance, and V_{dd} the supply voltage. This expression is similar to equation 2.29, where the load capacitance C_{L} is replaced by the total switched capacitance C.

Substitution of equation 2.43 in equation 2.42 gives

$$P = \alpha \frac{C {V_{\mathrm{dd}}}^2}{T}. \tag{2.44}$$

This expression clearly shows the dependence between the power usage, the supply voltage and the execution delay of the circuit.

Once a circuit is implemented, its power can be reduced by either increasing the execution delay or by decreasing the supply voltage. These methods are known as frequency and voltage scaling.

2.5.2.1 Frequency scaling

Obviously, the power can be reduced by increasing the execution delay T. This corresponds in a synchronous circuit to decreasing the frequency f as $f = \frac{1}{T}$. Substitution in equation 2.44 leads to the relation

$$P \propto f. \tag{2.45}$$

Thus, the power linearly depends of the frequency and it can be reduced by decreasing the frequency.

2.5.2.2 Voltage scaling

In frequency scaling, the power reduction is only linear in frequency. At first sight, a quadratic reduction can be achieved by scaling down the supply voltage of the circuit

$$P \propto {V_{\mathrm{dd}}}^2. \tag{2.46}$$

However, when we lower the supply voltage, the gate delay increases (see equation 2.19). As a result, the power is even more reduced[3], ultimately resulting in a cubic dependency on the supply voltage

$$P \propto {V_{\mathrm{dd}}}^3. \tag{2.47}$$

It should be noted that the voltage cannot be reduced indefinitely, because when the supply voltage approaches the threshold voltages of the transistors, the robustness of the transistors against noise is severely lowered and proper circuit behavior is compromised.

2.5.3 Area versus delay versus power

Above, power reduction techniques were discussed. These techniques are used for already implemented circuits and consequently the area is assumed to be fixed. If we release this restriction, the influence of the area on the capacitance and delay has also to be taken into account.

To relate area to delay, we use the lower bound specified in equation 2.40 in a slightly modified form

$$\mathrm{AT}^r = \gamma, \tag{2.48}$$

where r and γ are constants.

For the power usage we have found (see section 2.5.2)

$$P = \alpha \frac{C{V_{\mathrm{dd}}}^2}{T}. \tag{2.49}$$

Two options emerge; either the supply voltage is kept constant, or it is scaled, thus decreasing the power and increasing the delay. These alternatives are discussed below.

2.5.3.1 Constant voltage

In order to come up with a solvable set of equations, we have to find an expression for the average switched capacitance in the circuit. We assume it to linearly depend on the total area of the circuit. Hence

$$C = \mu_{\mathrm{c}} A, \tag{2.50}$$

with μ_c some constant.

Now combining equations 2.48, 2.49 and 2.50 and solving them gives the area and power as function of the delay

$$A = \frac{\gamma}{T^r},\ P = \alpha \mu_{\mathrm{c}} \gamma \frac{{V_{\mathrm{dd}}}^2}{T^{r+1}}. \tag{2.51}$$

In figure 2.4 this solution is plotted for various values of γ. For the sake of clarity, all other constants are set to unity.

2.5.3.2 Scaled voltage

Using a slightly modified version of equation 2.19, the average delay per gate T_{g} is given by

$$T_{\mathrm{g}} = \frac{C_{\mathrm{g}}}{\beta V_{\mathrm{dd}}}. \tag{2.52}$$

Here, C_{g} is the average load capacitance per gate and β a constant. The total delay of the circuit is assumed to linearly depend on the gate delay

[3] It is assumed that the circuit is running as fast as possible, i.e., the gate delay directly influences the delay of the circuit. If not, the quadratic dependency remains valid.

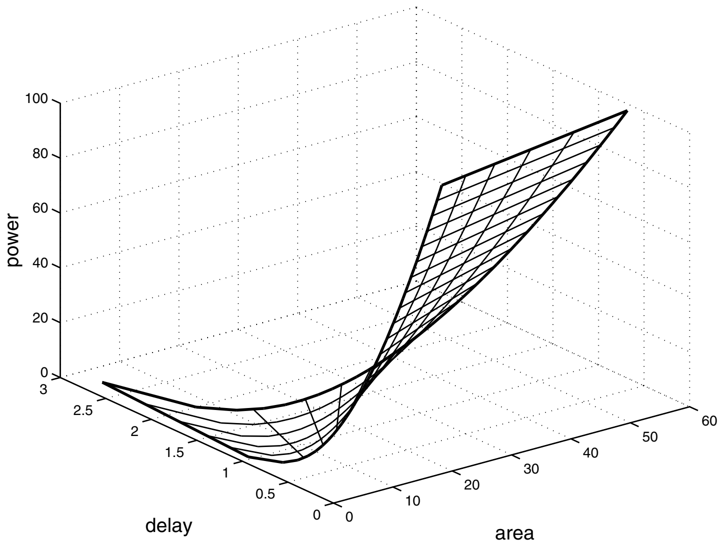


Figure 2.4. Constant voltage area, delay, power trade-off, for various values of γ.

$$T = c_T T_{\mathrm{g}}, \tag{2.53}$$

where c_T is again some constant.

Also it is assumed that the total capacitance is linearly dependent on the product of the average gate capacitance and the area

$$C_{\mathrm{av}} = \mu_{\mathrm{s}} C_{\mathrm{g}} A, \tag{2.54}$$

with μ_s a constant (different from μ_c).

Combining and solving equations 2.48, 2.49, 2.52, 2.53 and 2.54 gives the delay, area and power as function of the supply voltage or delay

$$T = \frac{c_T C_{\mathrm{g}}}{\beta V_{\mathrm{dd}}},\ A = \frac{\gamma}{T^r},\ P = \alpha \mu_{\mathrm{s}} \gamma \left(\frac{c_T}{\beta}\right)^2 \frac{{C_{\mathrm{g}}}^3}{T^{r+3}}. \tag{2.55}$$

In figure 2.5, this solution is plotted for various values of γ. Again, for the sake of clarity, all other constants are set to unity.

2.6 Summary

In this chapter, we have presented a short overview of techniques to estimate the chip area, delay and energy consumption of digital circuits. These estimates may serve as cost figures in the hardware–software partitioning tool presented in the next chapter. Crucial for the exploration of the design space of an algorithm (or set of algorithms) to be implemented is that the designer can quickly iterate within the design flow presented in figure 1.3, making changes to, for example, the original specification of the algorithm. To this end, a means to quickly get cost estimates is essential.

Fortunately, in present-day synthesis tools, area and timing prediction are commonly available. These predictions should be used whenever possible. Otherwise one would end up characterizing and modeling the internals of the synthesis tool. This meticulous job should then be repeated every time a new version of the tool is released, and whenever one switches to a different synthesis tool. Unfortunately, predicting the energy consumption of a circuit is harder than predicting the area

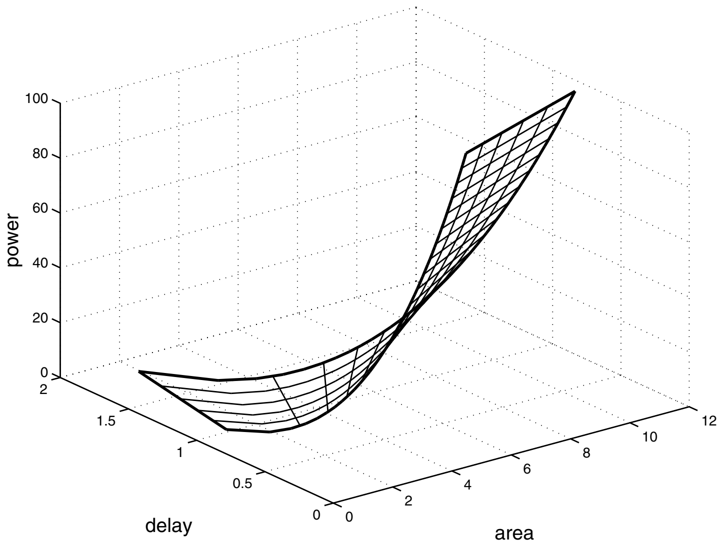


Figure 2.5. Scaled voltage area, delay, power trade-off, for various values of γ.

and timing, as it strongly depends on the particular characteristics of the processed data. As a consequence, synthesis tools still lack this feature and therefore, one of the methods discussed in section 2.4 should be applied.

Another issue discussed in this chapter is the possible trade-off between area, time and energy consumption. These trade-offs can be used to quickly find a range of costs estimates, given the cost figures of only a few realizations of a circuit (ultimately only one). Moreover, they give insight in what can be realistically expected within reasonable time.

As a final remark, we note that it would be interesting to investigate whether or not the entropy-based area estimation techniques described in section 2.2.1.2 can be extended to accurately estimate the energy consumption of any given function.

3 Architecting with uncertainties

3.1 Introduction

As argued in section 1.1.3, we need to decide which parts of an algorithm should be implemented in hardware and which in software. These decisions are governed by competing desires and needs:

- The desire to keep the development costs low; these costs include among others the design time and the production costs
- The need to meet the required performance, otherwise the resulting chip might become useless or worthless
- At least in case of battery-operated devices[1], the goal to keep the energy consumption low

These can be translated into the following opposing design goals:

- *Keep the chip small (the smaller, the better):* A chip with a larger area is more expensive to produce and furthermore has a lower yield.
- *Make the chip fast (the faster, the better):* Often, the performance of a chip is measured by the time it takes to execute an algorithm or by a derivative thereof, such as the data rate. If so, reducing the execution time increases the performance of the chip.
- *Consume little energy (the cooler, the better):* Obviously, to increase uninterrupted device operation times, the device should use as little energy as possible to accomplish its task.

A partitioning tool should come up with a feasible solution, while clarifying to what extent the design goals are met. This allows the designer to explore the design space and to make a well-founded decision.

The basic operation of the partitioning tool is to map an algorithm onto an architecture instance. From the set of all specified possible architectures to implement the algorithm, it should give the best instance with respect to given constraints. This approach was originally introduced in [33], and forms the basis of the work presented in the first part of this chapter.

In figure 3.1, the partitioning process is depicted. The start is an executable specification of the algorithm. Subsequently, this specification is first translated into a combined control data flow graph (CDFG), and then into a data flow graph (DFG). To implement the DFG, a set of architectures is specified, using the components specified in a library. For the DFG, the best architecture instance is selected, and simultaneously the algorithm is optimally divided into a hardware and a software part; this division is optimal with respect to one of the design constraints, which include the maximum allowed chip area, the maximum execution time and the maximum energy consumption.

[1] In case of devices connected to a power grid, high energy consumption might still be unacceptable because of undesirable side effects, such as reduced lifetime, expensive heat sinks, noisy fans, etc.

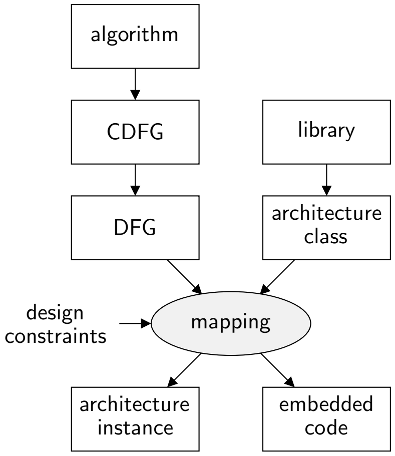


Figure 3.1. Design flow to map an algorithm to an architecture instance.

Once a solution has been found for the partitioning problem a single algorithm, the solution is extended to support multiple algorithms. Subsequently, a method is introduced to enable the use of inexact, i.e., fuzzy, numbers in the mathematical formulation, both for the single-algorithm case and for the multiple-algorithm case.

The second part of this chapter deals with the generation of synthesizable code from the original specification in C. The generated code is used to quickly come up with cost estimates to feed the optimization problem. The underlying thought is that without such a tool, the whole optimization procedure becomes only an exercise in linear programming, and no longer useful, as it cannot rely on reasonably accurate estimates (even though they are represented by fuzzy numbers).

3.2 Application model

In this book, we consider data-dominated algorithms. Therefore, we model the algorithms with a DFG. A DFG $\mathcal{G} = \{V, E\}$ is a directed graph where the nodes $v \in V$ represent processing and storage elements and the edges $e \in E$ represent data flowing to and from these elements.

The nodes (or vertices) V of the DFG $\mathcal{G}$ can be one of two types.

- *Processing elements $v \in V_b$:* Processing elements or processing blocks are vertices representing operations performed on the incoming data (incoming edges). The results of the operation are sent to storage elements (outgoing edges).
- *Storage elements $v \in V_s$:* Storage elements or (data) symbols are nodes representing pieces of stored data (variables or constants).

All nodes are either processing elements (blocks) or storage elements (symbols)

$$V = V_\mathrm{b} \cup V_\mathrm{s} \tag{3.1}$$

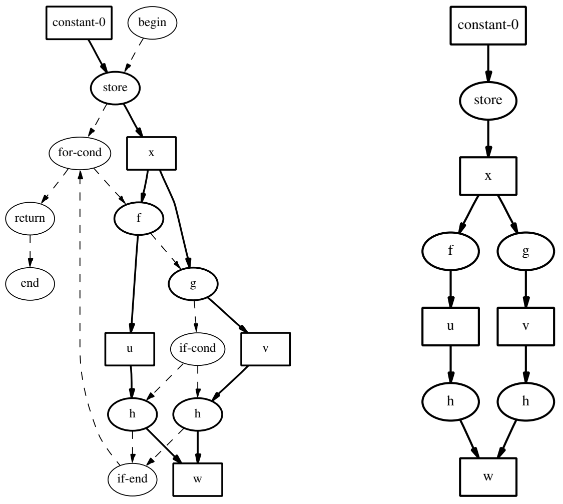
and no other types exist.

To construct the DFG from the C description of the algorithm, one needs to take a number of steps (see figure 3.1).

- *Create the combined CDFG:* The C-description of the algorithm is used to construct a graph representing both the control and the data flow of the algorithm.

```
/* dummy functions */
int f(int x) { return x; }
int g(int x) { return x; }
int h(int x) { return x; }

/* main algotithm */
void main() {
  /* control variables */
  int i;

  /* data variables */
  int u, v, w, x;

  /* input data */
  x = 1;

  /* static control */
  for (i=0; i<10; i++) {
        u = f(x);
        v = g(x);

        /* dynamic control */
        if (u > v)
          w = h(v);
        else
          w = h(u);
  }
}
```

(a) original C-code



(b) extracted CDFG (c) extracted DFG

Figure 3.2. Example of a combined control data flow graph (CDFG) and the extracted data flow graph (DFG).

- *Annotate the combined CDFG:* Running the algorithm, profiling information is generated and used to annotate the combined graph.
- *Extract the DFG:* The data flow is extracted from the combined CDFG. The control flow is discarded.
- *Annotate the DFG:* The sizes of the variables used in the algorithm and other profiling information are taken to annotate the DFG.

In the remainder of this section, each of these steps is discussed in more detail.

First, from the C-description of the algorithm, the combined CDFG is constructed. In the algorithm, both control flow and data flow statements are identified. The control flow represents the sequential behavior of the program. The data flow represents the flow of data and the operations performed on that data.

The C-code is parsed and the statements are analyzed. With every statement analyzed, the graph is extended: functions are mapped to processing elements and variables to storage elements. An example of a CDFG is depicted in figure 3.2(b) together with the original C-code fragment in figure 3.2(a).

The newly created CDFG does not contain profiling information yet. Therefore, the original C-description is compiled and run using a representative input data set. During the execution of the algorithm, a sequence is generated that contains the names and order of the functions called. This profiling information is used to annotate the CDFG with:

- The number of times a processing block (function) is executed
- The number of times a storage element (variable) is accessed
- The number of times a flow is activated

These numbers are used during the mapping to calculate the latency and the energy usage of the algorithm.

To extract the DFG from the CDFG, one simply removes the nodes and edges associated with control flow from the graph. This results in a graph representing the data flow of the algorithm. An example of an extracted DFG is depicted in figure 3.2(c).

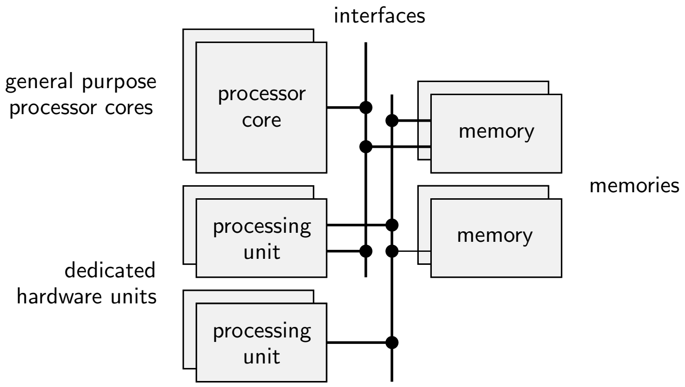


Figure 3.3. Example of an architecture class $\mathcal{C} = \{P, U, M, I\}$, defined by the available processor cores P, dedicated hardware units U, memories M and interfaces I.

At this stage the only information missing in the DFG is the sizes of the variables used. The sizes cannot be extracted automatically yet, therefore, they are manually specified in a list separate from the algorithm. This list is used to annotate the DFG with the sizes of the variables.

3.3 Architecture class

The algorithm should be mapped to an architecture. As the optimal architecture is not known yet, a class of architectures is specified, out of which the optimal architecture should be fabricated. An architecture class $\mathcal{C} = \{P, U, M, I\}$ is defined by its components.

- *Processor cores $p \in P$:* A processor core is capable of executing code, which implements various functions (processing elements). Additionally, it acts as the controller of the architecture. Only one processor core can be used in the final architecture instance.
- *Processing units $u \in U$:* A processing unit can be either a hardware unit or a software unit. A hardware unit is an additional specialized piece of hardware connected to the processor core implementing a specific function. A software unit is a code fragment to be run on the processor core which implements a certain functionality.
- *Memories $m \in M$:* A memory is capable of storing the variables and constants (symbols) used in the algorithm.
- *Interfaces $i \in I$:* An interface models the connections between the processor core, the additional hardware units and the memories. The same interface can be used to connect multiple units, thus modeling busses. Interfaces can be either hardware interfaces or mixed interfaces. A hardware interface connects a hardware processing unit with a memory. A mixed interface connects a software processing unit on a processor with a memory.

An example of an architecture class is depicted in figure 3.3.

The building blocks of an architecture class are used to construct the architecture instance (see figure 3.4) that best suits the algorithm to be implemented. The building blocks are divided into the subsets listed in table 3.1. This leads to the following relations

$$\forall_{v \in V}\ U_v = U_v^H \cup \bigcup_{p \in P} U_v^{S_p}, \tag{3.2}$$

$$\forall_{e \in E}\ I_e = I_e^H \cup \bigcup_{p \in P} I_e^{M_p}. \tag{3.3}$$

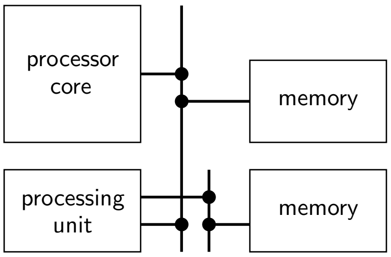


Figure 3.4. Example of an architecture instance.

Table 3.1. Sets of building blocks defining an architecture class

Symbol	Description
U_v	All available processing units for node (processing element) v
U_v^H	Available hardware processing units for node v
$U_v^{S_p}$	Available software processing units on processor p for node v
M_v	Available memories for node (storage element) v
I_e	All available interfaces for edge (flow) e
I_e^H	Available hardware interfaces for edge e
$I_e^{M_p}$	Available mixed interfaces for processor p for edge e

3.4 Hardware–software partitioning

To map the algorithm to an architecture (see figure 3.5), we formulate the mapping as a mixed integer linear programming problem (MILP) (see appendix B). The variables in this linear programming problem (see table 3.2) are binary valued variables which determine whether an implementation unit, memory or interface is used for a particular node (processing and storage elements) or edge (data flows). The constraints can be split up in two parts:

- *General selection constraints:* These constraints guarantee that a valid mapping is found. The mapping is not optimized in any way.
- *Cost constraints:* In addition to the general constraints, cost constraints are added in order to optimize the mapping. The cost constraints can be used as an optimization goal (maximize or minimize) or as boundary conditions.

These constraints are discussed in more detail in the next sections.

3.4.1 Selection constraints

The general selection constraints ensure that a valid solution is found for the mapping of the algorithm to an architecture.

3.4.1.1 Unit selection constraints

Every processing block $v \in V_b$ has to be implemented with either a hardware unit or a software unit. Only one implementation unit is selected for each block

$$\forall_{v \in V_b} \sum_{u \in U_v} s_u^v = 1. \tag{3.4}$$

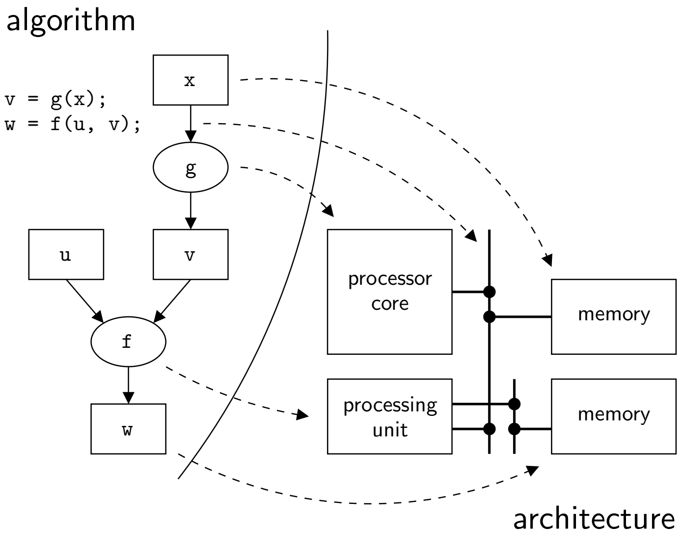


Figure 3.5. Example of mapping an algorithm on an architecture. During this mapping the optimal architecture is selected from the class of architectures.

Table 3.2. Binary valued decision variables (true = 1, false = 0)

Symbol	Description
s_u	True if unit u is used
s_m	True if memory m is used
s_i	True is interface i is used
s_u^v	True if unit u is used for block v
s_m^v	True if memory m is used for symbol v
s_i^e	True if interface i is used for flow e
s_H^v	True if a hardware processing element or memory is used for node v
$s_{S_p}^v$	True if a software processing element with processor p is used for node v
s_H^e	True if a hardware interface is used for edge e
$s_{HS_p}^e$	True if a hardware $\rightarrow$ software interface with processor p is used for edge e
$s_{SH_p}^e$	True if a software $\rightarrow$ hardware interface with processor p is used for edge e
s_p	True if processor p is used

3.4.1.2 Memory selection constraints

Every symbol $v \in V_s$ has to be assigned to a memory. Only one memory is selected for every symbol

$$\forall_{v \in V_s} \sum_{m \in M_v} s_m^v = 1. \tag{3.5}$$

Furthermore, a memory has a maximum size. Therefore, the combined size of the symbols assigned to a memory should not exceed the maximum size S_m of the memory

$$\forall_{m \in M} \sum_{v \in V_s} S_v s_m^v \leq S_m. \tag{3.6}$$

3.4.1.3 Interface selection constraints

All edges $e \in E$ in the DFG have to be assigned to an interface. Interfaces represent connections between processing blocks and storage elements. Therefore, the type of an interface is dependent on the type of the source and target of the flow (a hardware unit or a software unit). With the convenience variables

$$\forall_{v \in V} \quad s_H^v = \sum_{u \in U_v^H} s_u^v, \tag{3.7}$$

$$\forall_{v \in V} \quad s_{S_p}^v = \sum_{u \in U_v^{S_p}} s_u^v, \tag{3.8}$$

the proper type is selected using

$$\forall_{e \in E} \quad s_H^e = s_H^{v_s(e)} \cdot s_H^{v_t(e)}, \tag{3.9}$$

$$\forall_{e \in E}\ \forall_{p \in P} \quad s_{HS_p}^e = s_H^{v_s(e)} \cdot s_{S_p}^{v_t(e)}, \tag{3.10}$$

$$\forall_{e \in E}\ \forall_{p \in P} \quad s_{SH_p}^e = s_{S_p}^{v_s(e)} \cdot s_H^{v_t(e)}, \tag{3.11}$$

where $v_s(e)$ and $v_t(e)$ denote the source and target node of edge e, respectively. The $\cdot$ operator denotes a Boolean “and”. It can be translated into integer programming constraints as described in Appendix B.3.

Obviously, only one type of interface is allowed per edge

$$\forall_{e \in E} \quad s_H^e + \sum_{p \in P} \left(s_{SH_p}^e + s_{HS_p}^e \right) = 1. \tag{3.12}$$

The equal sign ensures that exactly one interface type is selected per edge.

The only thing left is to select an interface once the proper type is known

$$\forall_{e \in E} \quad \sum_{i \in I_e^H} s_i^e = s_H^e, \tag{3.13}$$

$$\forall_{e \in E}\ \forall_{p \in P} \quad \sum_{i \in I_e^{M_p}} s_i^e = s_{SH_p}^e + s_{HS_p}^e. \tag{3.14}$$

Since only one type is selected, only one interface is selected per edge with these constraints.

3.4.1.4 Processor selection constraints

For every processing block implemented in software, the appropriate processor is selected

$$\forall_{v \in V_b}\ \forall_{p \in P} \quad s_p \geq \sum_{u \in U_v^{S_p}} s_u^v. \tag{3.15}$$

This constraint guarantees that the appropriate processor is selected if a processing block is implemented in software on that particular processor.

To ensure that exactly one processor is selected for all blocks, the following constraint is added:

$$\sum_{p \in P} s_p = 1. \tag{3.16}$$

Constraints for the selection of a processor for the interfaces are not necessary, because the interfaces are selected such that they match the source and target implementations.

3.4.2 Cost constraints

In addition to the general constraints, specific cost constraints can be added to express costs, such as memory size, code size, etc.. If necessary, maximum or minimum values can be specified to limit these costs.

The main cost objectives implemented are area, latency, energy and the global clock period. Either one can be selected for direct minimization or maximization. Also the execution time can be given as an optimization goal, although not directly, as explained in section 3.4.2.5.

For convenience, we introduce the selection variables listed in table 3.3. They determine whether a unit, memory or interface is used anywhere

$$\forall_{u \in U} \quad s_u = \bigvee_{v \in V_b} s_u^v, \tag{3.17}$$

$$\forall_{m \in M} \quad s_m = \bigvee_{v \in V_s} s_m^v, \tag{3.18}$$

$$\forall_{i \in I} \quad s_i = \bigvee_{e \in E} s_i^e, \tag{3.19}$$

where the $\bigvee$ operator refers to a Boolean inclusive "or". A Boolean "or" can be translated into integer programming constraints as described in Appendix B.3.

Table 3.3. Convenience variables (true = 1, false = 0)

Symbol	Description
s_u	True if unit u is selected
s_m	True if memory m is selected
s_i	True if interface i is selected

3.4.2.1 Area

The area cost is the amount of chip area that is necessary for the processor core, the additional hardware units, the memories, plus the connecting interfaces. For every processor, unit, memory and interface, area estimates are given (see table 3.4).

Table 3.4. Area cost estimates

Symbol	Description
A_u	Area for processing unit u
A_m^o	Minimum required area for memory m
A_m^d	Additional area per byte for memory m
A_i	Area for interface i
A_p	Area needed for processor core p

The area costs for the processor core, the hardware units, the memories and interfaces are given by

$$A_P = \sum_{p \in P} A_p s_p, \tag{3.20}$$

$$A_U = \sum_{u \in U_v^H} A_u s_u, \tag{3.21}$$

$$A_M = \sum_{m \in M} A_m^o s_m + \sum_{v \in V_s} S_v \sum_{m \in M_v} A_m^d s_m^v, \tag{3.22}$$

$$A_I = \sum_{i \in I_e^H} A_i s_i + \sum_{p \in P} \sum_{i \in I_e^{M_p}} A_i s_i, \tag{3.23}$$

respectively. From equation 3.22 one can see that the area for the memories is split into two respective parts: independent and dependent of the size of the symbols S_v stored in the memories.

Because we use the variables s_u and s_i in equations 3.21 and 3.23, these equations express the area cost for shared resources. If the resources are not shared, they have to be duplicated for every instance. Then, the aforementioned equations should be modified to use s_u^v and s_i^e instead and iterate over all nodes and edges.

The total area cost is calculated using

$$A_T = A_P + A_U + A_M + A_I, \tag{3.24}$$

where A_T is the total chip area needed.

3.4.2.2 Latency

The total latency is expressed by the number of cycles necessary to execute the complete algorithm. Hence, for every unit, memory and interface, latency estimates are provided (see table 3.5).

Table 3.5. Latency cost estimates (latency is expressed by the number of cycles necessary to complete an operation)

Symbol	Description
L_u	Latency of processing unit u
L_m^A	Latency for accessing data in memory m
L_m^T	Latency for transferring data to or from memory memory m
L_i	Latency per byte of data transferred via interface i

The latencies per unit, memory and interface are given by

$$L_U = \sum_{v \in V_b} F_v \sum_{u \in U_v} L_u s_u^v, \tag{3.25}$$

$$L_M = \sum_{v \in V_s} \sum_{m \in M_v} \left(F_v L_m^A + \sum_{e \in e(v)} F_e S_e L_m^T \right) s_m^v, \tag{3.26}$$

$$L_I = \sum_{e \in E} F_e S_e \sum_{i \in I_e} L_i s_i^e, \tag{3.27}$$

respectively, where $e(v)$ denotes all incoming and outgoing edges of node v. The total latency is calculated using

$$L_T = L_U + L_M + L_I, \tag{3.28}$$

where L_T is the total number of cycles necessary to complete the algorithm.

3.4.2.3 Energy

The energy cost is the total amount of energy used during execution of the algorithm on the selected hardware. It includes energy used by the processor core to run the processing blocks implemented in software and the energy consumed by the hardware units, memories and interfaces. For every processor, unit, memory and interface, energy estimates are given (see table 3.6).

Table 3.6. Energy cost estimates

Symbol	Description
E_u	Energy usage of unit u
E_m^A	Energy usage for accessing data in memory m
E_m^T	Energy usage for transferring data to or from memory m
E_i	Energy usage per byte of data transferred via interface i
E_p	Energy usage per cycle for processor p

The energy usage per unit, memory and interface is given by

$$E_U = \sum_{v \in V_b} F_v \left(\sum_{u \in U_v^H} E_u s_u^v + \sum_{p \in P} E_p \sum_{u \in U_v^{S_p}} L_u s_u^v \right), \tag{3.29}$$

$$E_M = \sum_{v \in V_s} \sum_{m \in M_v} \left(F_v E_m^A + \sum_{e \in e(v)} F_e S_e E_m^T \right) s_m^v, \tag{3.30}$$

$$E_I = \sum_{e \in E} F_e S_e \left(\sum_{i \in I_e^H} E_i s_i^e + \sum_{p \in P} \sum_{i \in I_e^{M_p}} (E_i + E_p L_i) s_i^e \right), \tag{3.31}$$

respectively. The total energy cost is calculated using

$$E_T = E_U + E_M + E_I, \tag{3.32}$$

where E_T is the total energy used to run the algorithm on the selected architecture.

3.4.2.4 Clock period

Another important optimization objective is the global clock period. It is assumed that in the final architecture, all blocks are governed by a single clock. For every processor, hardware unit, memory and interface, the minimum clock period is specified (see table 3.7).

The global clock period C is lower bounded by

$$\forall_{u \in U^H} \quad C \geq C_u s_u, \tag{3.33}$$

$$\forall_{m \in M} \quad C \geq C_m s_m, \tag{3.34}$$

$$\forall_{i \in I^H} \quad C \geq C_i s_i, \tag{3.35}$$

$$\forall_{p \in P} \quad C \geq C_p s_p, \tag{3.36}$$

where $U^H = \bigcup_{v \in V_b} U_v^H$ and $I^H = \bigcup_{e \in E} I_e^H$ denote the available hardware units and hardware interfaces, respectively. Software units and mixed interfaces do not impose any additional constraints, because these execute with the same clock period as the processor core they are executing on or connected to.

Table 3.7. Minimum clock period estimates

Symbol	Description
C_u	Minimum clock period for unit u
C_m	Minimum clock period for memory m
C_i	Minimum clock period for interface i
C_p	Minimum clock period for processor p

3.4.2.5 Execution time

As the total execution time T is the product of the total latency and the overall clock period $T = L_T C$, i.e., the product of two variables, it cannot be expressed directly in a linear form necessary for MILP formulation. Therefore, the following procedure is applied to circumvent this problem.

An upper bound for the global clock period is obtained using

$$C \leq \sum_{x \in X} C_x s_x \tag{3.37}$$

and

$$\sum_{x \in X} s_x = 1, \tag{3.38}$$

where X is the set of clock estimates, C_x the estimated clock period and s_x a binary valued variable indicating whether clock estimate x is selected or not. The latter constraint is introduced to ensure that only one clock estimate is selected.

Once an estimate is available, the maximum execution time can be translated into a maximum number of cycles (latency). Thus, the maximum execution time can be guaranteed to be less than the desired maximum execution time $T_{\max}$ by means of

$$L_T \leq \sum_{x \in X} \frac{T_{\max}}{C_x} s_x. \tag{3.39}$$

These linear equations suit the MILP formulation. Obviously, this formulation does not allow the execution time to be an optimization objective.

To enable execution time optimization, we introduce the local optimization variables λ_x to replace the selection variables s_x in equation 3.39

$$L_T \leq \sum_{x \in X} \frac{T_{\max}}{C_x} \lambda_x. \tag{3.40}$$

Now, by minimizing the global optimization variable λ

$$\lambda = \sum_{x \in X} \lambda_x, \tag{3.41}$$

we minimize the execution time T provided that $\lambda_x = 0$ if $s_x = 0$. Therefore, we add

$$\forall_{x \in X} \;\; \lambda_x \leq s_x, \tag{3.42}$$

which ensures a correct optimization.

3.4.3 Mixed integer linear programming formulation

The selection and cost constraints can be reordered and summarized in matrix form as

$$A_s x_s \leq b_s \tag{3.43}$$

and

$$A_c x_c \leq b_c. \tag{3.44}$$

With these definitions we can define several MILP problems, each of which aims at minimizing a different objective under different constraints

$$\begin{aligned}
&\text{minimize} \\
&\qquad A_T, E_T \text{ or } \lambda, \\
&\text{subject to} \\
&\qquad \begin{bmatrix} A_s & 0 \\ 0 & A_c \end{bmatrix} \begin{bmatrix} x_s \\ x_c \end{bmatrix} \leq \begin{bmatrix} b_s \\ b_c \end{bmatrix}, \\
&\text{bounded to} \\
&\qquad A_T \leq A_{\max}, \\
&\qquad E_T \leq E_{\max}, \\
&\qquad L_T \leq L_{\max}, \\
&\qquad C \leq C_{\max}, \\
&\qquad T \leq T_{\max}.
\end{aligned} \tag{3.45}$$

In equation 3.45, $A_{\max}$, $L_{\max}$, $E_{\max}$, $C_{\max}$ and $T_{\max}$ are the maximum values for the area, latency, energy, clock period, and execution time, respectively. These bounds can be updated in sequential optimization runs, which each change the optimization objective and lower the maximum allowed costs.

3.5 Extension to multiple algorithms

In the approach presented so far, an optimal architecture instance was selected for a single algorithm. This approach does not suffice for solving more general problems.

- *How to implement multiple algorithms using the same hardware:* Often, a device has to perform different tasks. These tasks should preferably be implemented on the same hardware to save chip area. Therefore, the partitioning tool should be able to find an optimal mapping for a set of algorithms to be implemented on the same hardware.
- *How to select the best algorithm:* In case multiple algorithms are available to implement a system, there may not yet be an algorithm of choice. Hence, the partitioning tool should be able to select the best algorithm (in terms of cost) out of many.
  Obviously, the tool does not take into account the performance of the algorithms. It is still up to the designer to evaluate and compare the relative performance of the algorithms.

To address these problems, we have to modify the partitioning tool in several ways. In the next sections, the above two problems are first addressed individually and finally a combined solution is presented.

3.5.1 Multiple algorithm support

The extension of the partitioning tool to support a set of algorithms to be implemented is relatively straightforward. All it requires is replacing the original single DFG by a combined DFG by compounding the nodes and edges of the DFGs of the individual algorithms in the set.

No other modifications are necessary, since in the original MILP formulation, resource sharing was already enabled. The resulting architecture instance will be optimal for the combination of algorithms and not necessarily optimal for a single algorithm.

Table 3.8. Additional sets and variables for the extended MILP formulation

(a) Sets

Symbol	Description
G	Set of algorithm groups (sets)
A	Complete set of algorithms
A_g	Set of algorithms in group g
V_a	Set of nodes representing processing elements or memories in algorithm a
V_b^a	Set of nodes representing processing elements in algorithm a
V_s^a	Set of nodes representing data symbols in algorithm a
E_a	Set of edges representing data flows in algorithms a

(b) Variables

Symbol	Description
s_a	True if algorithm a is selected
s_g	True if algorithm group g is selected

3.5.2 Algorithm selection support

In order to support algorithm selection, we modify the original MILP formulation in two ways. First, the same extension is applied as presented in the previous section to support sets of algorithms. Second, an algorithm selection variable is introduced and the selection constraints given by 3.4, 3.5 and 3.12 are modified such that they only enable the selection of blocks when the algorithm is selected (algorithm selection variable is true). To ensure that exactly one algorithm is selected, we need one extra constraint.

3.5.3 Combined solution

The modifications presented in the previous sections are combined to address the more general case of selecting the best set of algorithms to be implemented on the same hardware. The additional sets and variables necessary are listed in table 3.8(a) and 3.8(b), respectively.

First, the DFG $\mathcal{G}$ is replaced by a combination of the DFGs of all algorithms

$$\mathcal{G} = \left\{ \bigcup_{a \in A} V_a, \bigcup_{a \in A} E_a \right\}. \tag{3.46}$$

Second, the selection constraints are modified to support the selection of algorithms

$$\forall_{a \in A} \ \forall_{v \in V_b^a} \quad \sum_{u \in U_v} s_u^v = s_a, \tag{3.47}$$

$$\forall_{a \in A} \ \forall_{v \in V_s^a} \quad \sum_{m \in M_v} s_m^v = s_a, \tag{3.48}$$

$$\forall_{a \in A} \ \forall_{e \in E_a} \quad s_H^e + \sum_{p \in P} \left(s_{SH_p}^e + s_{HS_p}^e \right) = s_a. \tag{3.49}$$

Third, if a group is selected, all algorithms in that group should be selected as well

$$\forall_{g \in G} \quad \sum_{a \in A_g} s_a = |A_g| \, s_g, \tag{3.50}$$

and finally only one group of algorithms should be selected

$$\sum_{g \in G} s_g = 1. \tag{3.51}$$

Together, these modifications ensure that the group of algorithms with the lowest costs is selected. At the same time they determine the optimum architecture instance for the group as a whole.

As the area, energy, latency and clock period costs are all computed solely using selection variables, only calculation and optimization of the the execution time have to be modified to support multiple algorithms. Thereto, the λ_x variables in equations 3.40 and 3.41 are replaced by λ_x^a, which yields

$$L_T \leq \sum_{a \in A} \sum_{x \in X} \frac{T_{\max}}{C_x} \lambda_x^a \tag{3.52}$$

and

$$\lambda = \sum_{a \in A} \sum_{x \in X} \lambda_x^a. \tag{3.53}$$

To ensure that λ_x^a is annulled, when an algorithm is not selected, λ_x^a is limited, not only by s_x, but now also by the algorithm selection variable s_a

$$\forall_{a \in A} \ \forall_{x \in X} \quad \lambda_x^a \leq s_a, \tag{3.54}$$

$$\forall_{a \in A} \ \forall_{x \in X} \quad \lambda_x^a \leq s_x. \tag{3.55}$$

Minimizing λ reduces the combined execution time of a group of algorithms to a minimum.

3.6 Dealing with uncertainty

In the MILP formulation of the partitioning problem presented in previous sections, exact cost estimates are assumed. These numbers might be available if IP-blocks are reused, but for many units and memories, the exact numbers will not be available and one has to use estimates for the area, latency and energy usage per instance. Inherent to the design process is that the higher level one starts the design process, the more inaccurate the initial estimates will be.

A drawback of the MILP formulation of the partitioning problem is that the optimal solution will be found on a boundary of the solution space. As a consequence, a small error in an estimate might result in an infeasible solution that crosses the boundary and thus violates a constraint.

Hence, it would be desirable if the cost estimates could be entered as inexact or non-crisp numbers, i.e., fuzzy numbers. If we use triangular fuzzy numbers, the resulting fuzzy MILP can be solved using Zimmermann's [87] fuzzy programming method with the normalization process described in [38].

A triangular fuzzy number is an imprecise number with a triangular possibility distribution as depicted in figure 3.6. A triangular number $\tilde{c}$ is specified by its most possible value c_m and its least possible values c_l and c_u

$$\tilde{c} = \{c_l, c_m, c_u\}, \tag{3.56}$$

where c_l and c_u are the lower bound and the upper bound of $\tilde{c}$, respectively and thus $c_l \leq c_m \leq c_u$.

First, the single-objective MILP with fuzzy (triangular) coefficients is translated into a multi-objective MILP with crisp coefficients, which is subsequently translated into a crisp single-objective MILP. Finally, a solution is determined using a standard solver for a single-objective MILP with crisp coefficients. Solving a single fuzzy MILP is equivalent to solving seven crisp MILPs of approximately the same size. For a detailed description, see Appendix C.

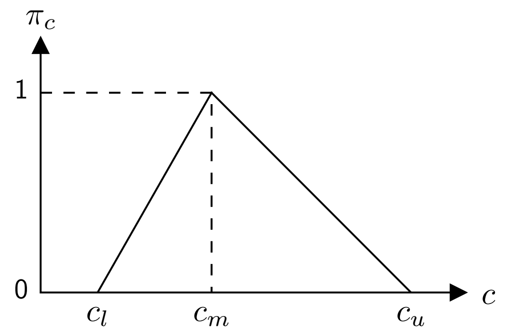


Figure 3.6. Fuzzy (imprecise) number $\tilde{c} = \{c_l, c_m, c_u\}$ with triangular possibility distribution π_c.

3.7 C to SystemC conversion

As discussed earlier in section 1.1, the design of an ASIC usually starts with a functional specification of the algorithm to be implemented in a high-level programming language like C, C^{++} or Matlab, because of the high level of abstraction and the high simulation speeds. However, these languages are typically purely sequential and lack constructs to express timing.

In order to use this specification to design an ASIC, we have top map it to a hardware description language (HDL). Current tools to map a specification in C to an HDL such as VHDL or Verilog either support only a small subset of the programming language or restrict a designer in other ways, e.g. by targeting a predefined architecture. These difficulties arise because of the previously mentioned differences in representation between a programming language like C and an HDL: sequential versus concurrent (see section 1.1.2).

With the growing maturity of SystemC [42] as modeling language, resulting in the approval of the IEEE 1666-2005 SystemC Standard [29] by the Standards Association of the IEEE in December 2005, followed by its approval by the American National Standards Institute in March 2006, combined with the availability of SystemC synthesis tools (for instance the CoCentric SystemC Compiler of Synopsys), another approach has become attractive, namely to create a SystemC model of the design from the original specification in C or C^{++}. This SystemC model can subsequently be refined to a final synthesizable description. The approach still suffers from the conversion difficulties mentioned earlier. However, since SystemC is based on C^{++}, the majority of the code can directly be incorporated in the SystemC model. This greatly simplifies the conversion process.

3.7.1 SystemC language

SystemC was first introduced to the general public in September 1999 with the release of SystemC v1.0. It consisted of a C^{++} class library providing the necessary constructs for hardware modeling, such as modules, ports, processes and events and a simulation kernel. In addition, it included basic data types such as bits, bit vectors, arbitrary precision integers and fixed-point numbers. At that time, the functionality of SystemC roughly matched the functionality offered by VHDL or Verilog.

With the subsequent release of SystemC v2.0 in October 2001 the core language was extended with the abstract notion of communication channels and interfaces. Furthermore, a master–slave library and a verification library were added. At this stage, the functionality of SystemC clearly superseded VHDL and Verilog, offering gradual refinement from a high abstract level towards synthesizable code within a single language.

On December 12, 2005, the IEEE approved the IEEE 1666-2005 standard for SystemC, which was based on the SystemC v2.1 release. It was followed by the approval of the American National Standards Institute, 28 March 2006. At the time of this writing, SystemC is supported by various

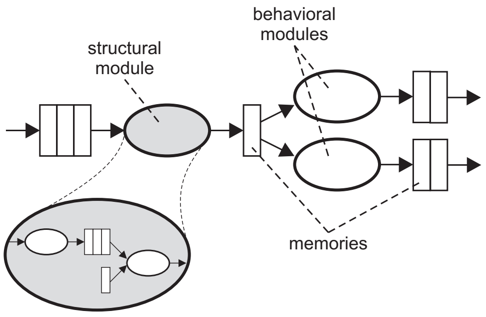


Figure 3.7. Example of the targeted SystemC model, consisting of structural and behavioral modules communicating via dedicated memories.

EDA (electronic design automation) companies for simulation and synthesis purposes, including Synopsys and Cadence.

3.7.2 Conversion approach

The major difference with the original executable specification is that the system is now modeled as concurrently executing processes. In SystemC, these processes correspond to SystemC modules. Each module consists of one or more concurrently executing threads. Of course these modules have to interface with other modules. Therefore, ports are added to communicate data between modules. To break the dependency between reading and writing of data between two modules, dedicated memories are inserted between the modules, acting as buffers. In this way, a module can generate and output its results without having to wait for the next module to be ready to accept input. An example of this model is depicted in figure 3.7. Each module has either a structural or a behavioral description. A structural module is composed of several nested modules.

In essence we fit the sequential C code into a data flow model. However, since generic C does not necessarily match a data flow model, we impose a number of restrictions on the C code.

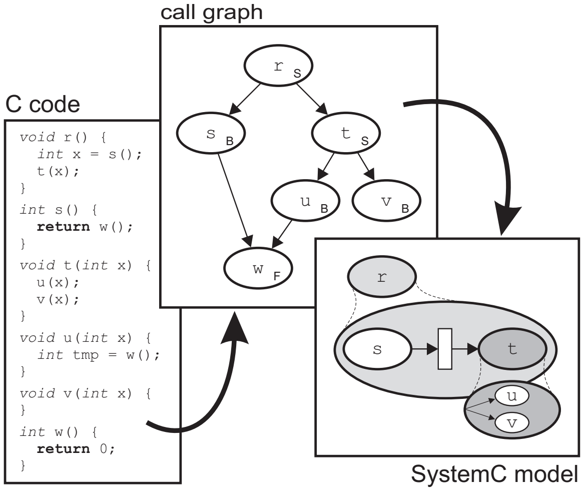
- *Only static memory allocation is allowed:* Dynamic memory allocation (via calls to `malloc()` and `free()`) requires a memory allocation unit and a shared memory [75], which is undesirable.
- *Pointers may only be used as references:* For the same reason, pointer arithmetic is disallowed. Pointers may be used as references to variables, however.
- *Only local variables are allowed:* Global variables require a shared memory. Hence, only variables with a local scope are allowed.
- *Arrays passed via function arguments must have predefined sizes:* In order to reserve storage space within a module, it must be known explicitly what the size of an array passed to a function via its arguments is.

It must be noted that these restrictions are necessary to enable the conversion step. When we target synthesizable code, additional restrictions arise, imposed by the synthesis tools used (see also section 3.7.5.4).

To construct the SystemC model, we need a module hierarchy. For this, we adhere to the hierarchy of the C code. Good programming practice is to divide the global function to be implemented into several sub-functions, which in turn may again be divided, and so on. Hence, we exploit this common practice to decompose the overall algorithm hierarchically. The hierarchy forms a so-called call graph, a directed graph where the nodes represent the functions, and the edges the function calls (see figure 3.8).

A function (node in the call graph) has to be mapped to one of three alternatives:

- A structural description for the corresponding SystemC module



```
void r() {
  int x = s();
  t(x);
}
int s() {
  return w();
}
void t(int x) {
  u(x);
  v(x);
}
void u(int x) {
  int tmp = w();
}
void v(int x) {
}
int w() {
  return 0;
}
```

Figure 3.8. Example of mapping a call graph (S = structural module, B = behavioral module, F = supporting function), with the original C code and its corresponding SystemC model.

- A behavioral description for the corresponding SystemC module, or
- A supporting function of a behavioral module.

Furthermore, two mapping rules have to be obeyed:

- Only structural modules can incorporate other modules (either behavioral or structural modules)
- Only behavioral modules can use supporting functions.

An example of a possible mapping and the corresponding C code and SystemC model are shown in figure 3.8.

Implicitly, the mapping rules result in an important constraint for functions, which are converted into either structural or behavioral modules: recursive function calls, either direct or indirect ones, are not allowed. This requirement ensures that nodes in the call graph corresponding to structural and behavioral modules are not part of any cycle.

The requirements and restrictions to the C code presented so far are easily met: just follow a limited set of simple programming rules. The designer does not need to learn any extensions or annotate the C code in any other way.

The selection of a mapping together with the one-to-one correspondence between functions and modules give the designer full control over the granularity of the final SystemC model. Selecting a mapping is not done automatically, but left to the designer. To our opinion the mapping is a design decision that the designer has to take. Furthermore, it is very difficult to make a sensible decision automatically without prior knowledge of the target.

3.7.3 Behavioral conversion

A behavioral module consists of input and output ports for control and data signals, and methods specifying at least the functional behavior of the module. In the generated modules, three methods are defined: `input()`, `output()` and `exec()`. The first two manage the reading and writing of data from and to the input and output ports, respectively. The third contains the main functionality of the module.

The function's body specifies its behavior and can be incorporated directly into the SystemC module, into the `exec()` method. To enable the tracing[2] of variables, all variable declarations are stripped from the function's body and the variables are declared as data members of the SystemC module. Any pointers are detected and replaced by references.

The remaining problem in constructing a behavioral module involves translating the input and output arguments of the function to their port equivalents. In the C language, no explicit mechanisms exist to uniquely identify whether an argument is an input or an output of a function. However, implicitly some information is available.

- Variables passed by value to the function cannot be modified by the called function and thus these arguments are always inputs.
- Constant arguments (whether variables are passed by value or by reference) are inputs by definition.
- Arguments passed by reference (through pointers or array constructs) can serve both as input and output.
- A non-void return value of the function serves as an output.

Summarizing, the only ambiguity arises when arguments are passed by reference. Several options exist to remove this ambiguity.

- Interpret these arguments as a combined input and output.
- Annotate the original code (for instance via a dedicated `#pragma` directive[3]).
- Parse and analyze the body of the function to see whether arguments are read or written to.
- Make sure that the programmer declares all inputs passed by reference as constant.

The first option introduces unnecessary overhead in many cases. The second is unattractive, because it is error prone. It might introduce a discrepancy between the executable specification, which does not use the annotated information, and the generated SystemC model, which does. The third is the most general, but is difficult to implement. Finally, the fourth does not cause overhead, because combined input and output ports are no longer possible. It is easy to implement and guarantees equivalent C and SystemC models. Therefore, the fourth option is chosen to remove the ambiguity. If the designer erroneously does not declare an input passed by reference as constant, this error is detected when the generated SystemC model is executed.

3.7.4 Structural conversion

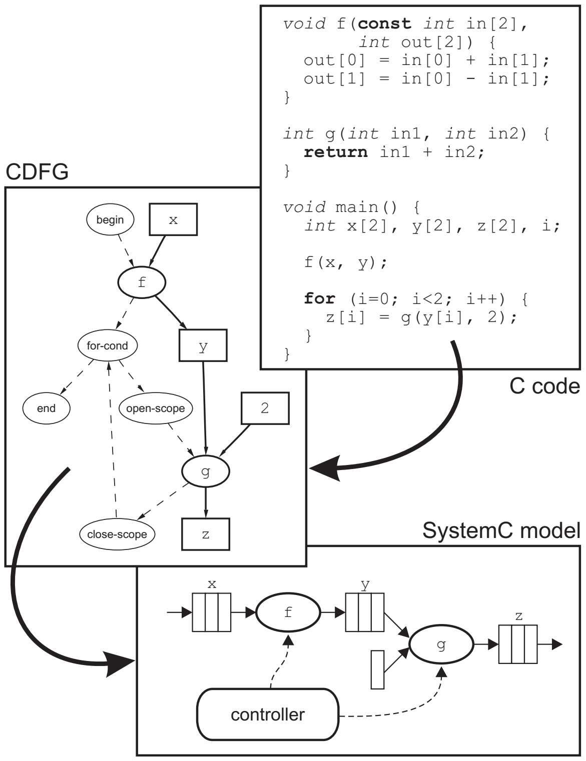
As depicted in figure 3.9, a two-step approach is used to convert a C function into a structural SystemC module. First, a combined CDFG is derived from the C source. Second, the CDFG is used to construct a structural module of the function. The CDFG is a directed graph where the nodes represent either variables or operations. The edges represent either the data or control flow.

To derive the CDFG from a C function, we use the SUIF system [1, 84]. The SUIF tools are used to parse the source code of the function and to generate the SUIF intermediate representation (IR) of the program, a tree. Subsequently, a pass is written that traverses the IR tree and generates the CDFGs of the functions encountered.

Next, the DFG is extracted from the CDFG. The nodes are translated into modules and memories, the edges to signal connections between them. The modules correspond to the functions called by the function and the memories to its variables and arguments.

[2] SystemC supports the tracing of a data member of a module, i.e., it logs all value changes of that data member.

[3] The `#pragma` directive is a preprocessing construct like `#include`, which causes implementation-dependent behavior if the token sequence following the directive is recognized. Unrecognized pragmas are ignored.



```
void f(const int in[2],
       int out[2]) {
  out[0] = in[0] + in[1];
  out[1] = in[0] - in[1];
}

int g(int in1, int in2) {
  return in1 + in2;
}

void main() {
  int x[2], y[2], z[2], i;

  f(x, y);

  for (i=0; i<2; i++) {
    z[i] = g(y[i], 2);
  }
}
```

Figure 3.9. Two-step structural conversion approach: C into a control data flow graph (CDFG) and the CDFG into a SystemC module.

Each module has a controller, which orchestrates the execution of the underlying modules. The information stored in the CDFG allows for basic scheduling optimizations [45], but these have not been implemented yet. For now, we use profiling information to construct a valid schedule for the controller of the module.

3.7.5 Conversion alternatives

The designer is offered a set of features and alternatives to influence the conversion process. For example, he can choose to alter the timing scheme or to target synthesizable SystemC code. The conversion alternatives at the designer's discretion are discussed in the next sections.

3.7.5.1 Timing scheme

The designer has the possibility to select one of two alternative timing schemes.

- *Self-timed:* In this scheme, modules are allowed to execute as soon as they have gathered enough data at their data input ports. The modules do not have explicit control ports to trigger the execution of the module.
- *Scheduled:* Modules are started explicitly by a controller unit. They have explicit control ports to trigger their execution.

An advantage of the first scheme is that the central controller can be removed from the structural modules. However, if this timing scheme is applied, the SystemC module is not guaranteed to have the same behavior as its corresponding C function. For instance, if a function is originally called twice using the same inputs, the absence of new data at the input ports of the self-timed module will cause the second function call to be missed. The second scheme requires explicit control, but guarantees the same behavior for the SystemC module and its equivalent C function.

3.7.5.2 Pipelining

In normal operation, the generated behavioral modules have a single thread of execution. The thread calls the `input()`, `exec()` and `output()` methods sequentially.

To increase the throughput of the modules, pipelined modules are supported. These modules have three concurrently executing threads. Each thread calls one of the `input()`, `exec()`, or `output()` methods. As a consequence, the methods can be called concurrently. Of course the data passed between the thread has to be buffered with additional registers.

3.7.5.3 Limited-precision data types

An important step in the refinement of C code to a hardware description is the conversion of standard data types like integers and doubles into limited-precision data types. To this end, SystemC supports various limited-precision data types, both integer types and fixed-point types. For each type the number of bits must be chosen, and for the fixed-point types also the position of the fractional point.

Optionally, the conversion tool can convert all integer and floating-point variables into their SystemC equivalents. The exact formats of the numbers must be supplied by the designer. The formats can be derived using methods based on the analysis or simulation of the data flow model or they can be derived of the original executable specification [36, 79, 83].

Another possibility is to use an external software tool to convert either the original C code or the generated SystemC code such that is uses limited-precision data types only. Several packages are available for this, such as FRIDGE [34] or the commercial package CoCentric Fixed-Point Designer from Synopsys [80].

3.7.5.4 Synthesis-oriented conversion

So far, we have not considered generating synthesizable SystemC code specifically. In this section, we concentrate on a particular synthesis tool: CoCentric SystemC Compiler from Synopsys. Therefore, the considerations regarding synthesizable code presented in this section apply to this synthesis tool. Similar considerations may arise for tools from other vendors.

The Synopsys synthesis tool requires synchronous modules. Therefore, clocked processes are applied and a clock input port is added to the modules, introducing time into the model. Consequently, we have to solve a number of timing issues [81]:

- To prevent zero-delay between consecutive port accesses
- To prevent zero-delay loops.

These issues only affect behavioral modules.

The first issue is readily solved by adding a `wait()` statement[4] after each port access, causing the module to suspend until the next clock cycle after each read or write from or to a port. As ports

[4] The SystemC `wait()` statement causes the module to suspend process execution until an event occurs on one of the signals the process is sensitive to. In a clocked process, one such signal is the module's clock.

are only accessed in the `input()` and `output()` methods, the main functionality in the `exec()` method is left unmodified.

To solve the second issue, the main functionality of the module (the `exec()` method) has to be modified. In every loop statement, a `wait()` statement is inserted. (For now, we only support `for` and `while` loop constructs.)

3.8 Summary

In this chapter, we discussed a mathematical programming solution to the hardware–software partitioning problem. First, a formulation was introduced for mapping a single algorithm to an optimal architecture. Subsequently, the formulation was extended to support the mapping and selection of multiple algorithms.

To deal with the uncertainty inherent to high-level design, triangular numbers were introduced to describe the cost estimates, leading to a fuzzy optimization problem. The optimization method used optimizes the most probable outcome, while minimizing the chances of finding a worse solution and maximizing the chances of a better one.

Apart from the solution for the partitioning problem, we described a procedure to quickly generate a synthesizable description of the original specification in order to get the initial cost estimates. The description language used is SystemC, as it has a large syntactic resemblance to the C language, used for the initial specification of the algorithms. Existing approaches to translate C into VHDL or Verilog did not provide a satisfactory solution.

The methods introduced in this chapter have all been implemented. To test and verify the ideas we will apply these design tools in chapter 5, to implement an OFDM transceiver. But first, to get a better understanding for this kind of transceiver, the next chapter discusses multi-carrier communications in general, and OFDM in particular.

4

Multi-carrier communications

4.1 Introduction

In a multi-path channel, data transmission suffers from frequency-selective fading. At high symbol rates, this results in severe inter-symbol interference. An approach to overcome this problem is to divide the available bandwidth into many independent narrow sub-bands and to assign each sub-band to a so-called subchannel [9]. As a result the symbols are not transmitted serially over a single channel, but in parallel over the subchannels. This lowers the symbol rate per subchannel, effectively reducing the inter-symbol interference while maintaining the overall symbol rate. Additional measures, such as the use of a cyclic prefix, effectively annul the remaining inter-symbol interference.

Orthogonal frequency-division multiplexing (OFDM), also known as multi-carrier modulation (MCM) or discrete multi-tone (DMT), is a popular transmission scheme, in which such a technique is used. It enables reliable transmission over frequency-selective fading channels. It is adopted in several international standards such as digital audio broadcasting (DAB) [23], digital video broadcasting (DVB-T) [66], asymmetric digital subscriber lines (ADSL) [10] and more recently in the IEEE 802.11a and 802.11g standards [28]. Furthermore, it has been proposed for the universal mobile telecommunications system (UMTS) [25] standard.

In the first part of this chapter, we will introduce multi-carrier communications and in particular OFDM, starting with an explanation of the effects encountered in high data rate communications. Next, multi-carrier modulation is explained, which mitigates these effects, and finally we derive classical OFDM.

Before optimizing the implementation of a system with respect to area, latency and energy costs, one should first investigate the possibilities to reduce these costs beforehand. This means modifying the functionality of the system itself, of course within its specifications.

In multi-carrier systems, such as OFDM, it is possible to assign different energy levels to each subchannel. This property can be exploited to minimize the total energy necessary to realize a desired average bit error rate. In this way, the transmission power can be reduced while still meeting the desired performance.

To reduce the costs even further, we use fixed-point or integer arithmetic instead of floating-point calculations for the baseband processing. Of course, we aim to use as few bits as possible in order to keep the area, latency and energy usage as low as possible.

When implementing OFDM systems on integer digital signal processors (DSPs) or on fixed-point dedicated hardware, one has to decide how many bits to assign for the integer and fractional part of a fixed-point number. The main design constraint governing this decision is the performance desired of the resulting quantized OFDM system. Usually the relationship between the number of integer or fractional bits used and the resulting performance is found using Monte Carlo simulations. This would take a prohibitive amount of time in our approach to architecting. Therefore, in the second part of this chapter, simple analytical expressions are derived for the performance of uniformly quantized and soft-limited OFDM systems, so that the design space can be quickly explored, and the minimum number of integer and fractional bits needed to achieve the desired performance can be determined.

4.2 Multi-path channels

When a radio wave is transmitted from one antenna to another, multiple propagation paths may exist. A line-of-sight path may or may not be present. Other propagation paths are caused by various propagation effects.

- *Reflection:* Reflection occurs when an electromagnetic wave strikes an object which is very large in comparison with the wavelength of the propagating wave.
- *Scattering:* Scattering occurs when many small objects (in comparison with the wavelength of the propagating wave) obstruct the propagation path of the radio wave.
- *Diffraction:* Diffraction occurs when the radio wave hits a surface with sharp edges. The surface irregularities cause the wave to be deflected in all directions, and can even cause the wave to bend around the obstructing object.

As the wavelengths of the radio waves get smaller for higher frequencies, objects with small dimensions which did not obstruct the signal for lower frequencies start to interfere with the radio wave.

4.2.1 Channel impulse response

Each propagation path has its own characteristic propagation delay, amplitude attenuation and phase rotation. Consequently, the propagation effects of a multi-path channel can be modeled by its complex low-pass equivalent impulse response $h(\tau)$

$$h(\tau) = \sum_{i=0}^{L-1} \alpha_i \mathrm{e}^{j\theta_i} \delta(\tau - \tau_i). \tag{4.1}$$

Here, α_i, θ_i and τ_i are the amplitude attenuation, phase rotation and relative time delay of the ith propagation path, respectively, and L is the number of propagation paths. Without loss of generality the L paths are assumed to be ordered according to their path lengths (and thus according to their delay times).

As a physical channel must have a causal response, all delay times are larger than or equal to zero. Furthermore, because the absolute arrival times of the waves are not relevant, the delay of the shortest path is set to zero ($\tau_0 = 0$).

4.2.2 Delay spread

The root mean square (RMS) delay spread τ_{RMS} of a multi-path channel is defined as the second central moment of the channel's normalized delay power spectrum $\phi_c(\tau)$ [31]

$$\tau_{\mathrm{RMS}} = \sqrt{\int_{-\infty}^{\infty} (\tau - \tau_m)^2 \phi_c(\tau) \mathrm{d}\tau}, \tag{4.2}$$

where τ_m is the average excess delay time defined by

$$\tau_m = \int_{-\infty}^{\infty} \tau \cdot \phi_c(\tau) \mathrm{d}\tau. \tag{4.3}$$

The channel's normalized delay power spectrum is given by

$$\phi_c(\tau) = \frac{1}{A} \sum_{i=0}^{L-1} \alpha_i^2 \delta(\tau - \tau_i), \tag{4.4}$$

where α_i^2 is the power attenuation of the ith propagation path and A is given by

$$A = \sum_{i=0}^{L-1} \alpha_i^2, \tag{4.5}$$

which is the average power response or power gain of the channel.

The multi-path delay spread of a channel, T_m, is defined as the maximum path delay of the delay power spectrum. If the channel impulse response is known, it is equal to the longest path delay $T_\mathrm{m} = \tau_{L-1}$. If the impulse response is not fully known, the multi-path delay spread is often taken as

$$T_\mathrm{m} \approx \gamma \cdot \tau_\mathrm{RMS}, \tag{4.6}$$

where γ is a constant depending on the channel model assumed (usually between 3 and 5).

4.2.3 Inter-symbol interference

As data rates increase, symbol periods decrease (the symbol period is inversely related to the data rate). At very high data rates or in channels with a large delay spread, the symbol period becomes smaller than the delay spread. As a result, delayed versions of the transmitted symbols (or echoes) start to interfere with subsequent symbols. This effect is known as inter-symbol interference and is illustrated in figure 4.1.

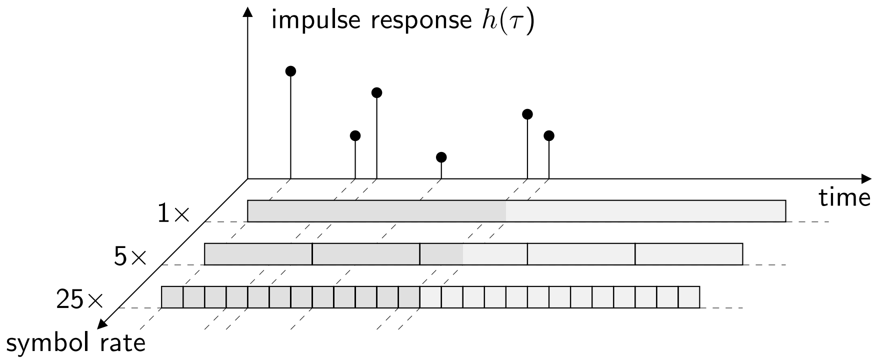


Figure 4.1. Multiple echoes cause inter-symbol interference.

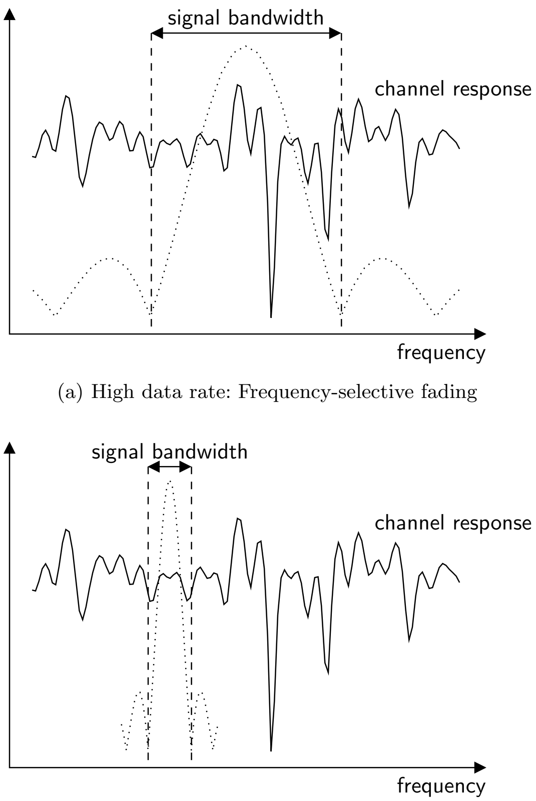
Even within a single symbol interval, the echoes can distort the signal. As each echo has its own amplitude attenuation and phase rotation, the echoes can either behave constructively or destructively.

Of course, in the presence of multiple propagation paths, there is always inter-symbol interference between two consecutive symbols. However, for long symbol periods, this interference is only a small fraction of the symbol period and the effect is usually negligible. In contrast, for short symbol periods, the interference may span several symbols and affects the whole symbol; it cannot be neglected anymore.

4.2.4 Frequency-selective fading

In the frequency domain, the inter-symbol interference can be viewed as frequency-selective fading, that is, the frequency response of the channel changes significantly within the bandwidth of the transmitted signal. This in contrast with flat fading where the frequency response of the channel is approximately constant within the bandwidth of the transmitted signal.

Where the symbol period decreases with higher data rates, the symbol bandwidth increases. Thus at low symbol rates, where the signal has a narrow bandwidth, the signal only experiences flat fading, but as the bandwidth increases, the signal starts to suffer from frequency-selective fading. This effect is shown in figure 4.2.



(a) High data rate: Frequency-selective fading

(b) Low data rate: Flat fading

Figure 4.2. At high data rates the transmitted signal suffers from frequency-selective fading.

4.3 Principles of multi-carrier modulation

Sequential equalizers are often used to mitigate the inter-symbol interference. However, many taps are needed when the interference spans many symbols at high data rates. In a time-variant environment, the equalizer becomes even more complex, because many tap coefficients have to be estimated quickly.

Another method to prevent inter-symbol interference is to enlarge the symbol period until the echoes of a single symbol do no longer span multiple symbols, i.e., the symbol period must be much larger than the delay spread of the channel. In the frequency domain, this can be seen as decreasing the bandwidth of the transmitted symbols until they only suffer from flat fading and no longer from frequency-selective fading. Further, interference reduction is achieved by using a guard interval between two consecutive symbols. A guard interval is a short protective period inserted between two symbols to prevent that echoes from the first symbol corrupt the second.

Expansion of the symbol period effectively lowers the symbol rate. To preserve the data rate, multiple symbols are transmitted simultaneously at different frequencies, known as sub-carriers. This can be achieved by applying a form of frequency-division multiplexing (FDM). In its simplest form FDM utilizes filter banks to separate the individual bands.

Summarizing, instead of transmitting symbols sequentially at a high rate, they are are transmitted in parallel at a lower rate. This way, the high data rate is preserved but the signal does not suffer from inter-symbol interference (or frequency-selective fading).

4.3.1 Orthogonal FDM

FDM methods that rely on filter banks use the available bandwidth inefficiently. To remedy this, we should space the sub-carriers as close as possible. The sub-carriers should be mutually orthogonal to retrieve the information mapped onto them.

In general, the message signal $m_i(t)$ transmitted on the ith sub-carrier consists of the data symbols z_i^k mapped on pulses with shape $g(t)$. It is given by

$$m_i(t) = \sum_{k=-\infty}^{\infty} z_i^k g(t - kT), \tag{4.7}$$

with T the symbol period and k the symbol index. Applying FDM gives the complex envelope of a multi-carrier signal $x(t)$

$$x(t) = \sum_{i=0}^{N-1} c_i(t), \tag{4.8}$$

where N is the number of sub-carriers and $c_i(t)$ the ith message mapped on the sub-carrier with frequency f_i

$$c_i(t) = m_i(t)\mathrm{e}^{j2\pi f_i t}. \tag{4.9}$$

The orthogonality constraint requires that the cross correlation of two modulated sub-carriers is zero

$$\int_{-\infty}^{\infty} c_u(t)\overline{c_v(t)}\mathrm{d}t = 0,\ \forall\ u \neq v, \tag{4.10}$$

where $\overline{c_v(t)}$ designates the complex conjugate of $c_v(t)$, or using equation 4.9

$$\begin{aligned} \int_{-\infty}^{\infty} c_u(t)\overline{c_v(t)}\mathrm{d}t &= \int_{-\infty}^{\infty} m_u(t)\mathrm{e}^{j2\pi f_u t} \cdot \overline{m_v(t)}\mathrm{e}^{-j2\pi f_v t}\mathrm{d}t \\ &= \int_{-\infty}^{\infty} m_u(t)\overline{m_v(t)}\mathrm{e}^{j2\pi (f_u - f_v)t}\mathrm{d}t = 0. \end{aligned} \tag{4.11}$$

Now consider the special case where $g(t)$ is a rectangular pulse

$$g(t) = \Pi\left(\frac{t}{T}\right), \tag{4.12}$$

with

$$\Pi(t) = \begin{cases} 1 \text{ if } 0 \leq t \leq 1, \\ 0 \text{ elsewhere.} \end{cases} \tag{4.13}$$

Then the resulting integral in equation 4.11 can be split as follows:

$$\int_{-\infty}^{\infty} c_u(t)\overline{c_v(t)}\mathrm{d}t = \sum_{k=-\infty}^{\infty} z_u^k \overline{z_v^k} \int_0^T \mathrm{e}^{j2\pi(f_u-f_v)t}\mathrm{d}t. \tag{4.14}$$

Thus in this special case, orthogonality is guaranteed when

$$\int_0^T \mathrm{e}^{j2\pi(f_u-f_v)t}\mathrm{d}t = 0, \tag{4.15}$$

which leads to

$$\frac{1}{j2\pi(f_u-f_v)}\mathrm{e}^{j2\pi(f_u-f_v)t}\Big|_0^T = 0, \tag{4.16}$$

and finally gives

$$\mathrm{e}^{j2\pi(f_u-f_v)T} - 1 = 0,\ f_u - f_v \neq 0. \tag{4.17}$$

It follows that the frequency separation f_Δ which guarantees orthogonality should be an integer multiple of the symbol rate $\frac{1}{T}$

$$f_\Delta = f_u - f_v = \frac{k}{T},\ k = \pm 1, \pm 2, \ldots. \tag{4.18}$$

Combining equations 4.8 and 4.18 gives the complex envelope of an OFDM signal with a minimum bandwidth

$$x(t) = \sum_{i=0}^{N-1} m_i(t)\mathrm{e}^{j2\pi\frac{it}{T}}. \tag{4.19}$$

Here, the frequency of the ith sub-carrier is recognized as $f_i = \frac{i}{T}$, which leads to a frequency separation of $f_\Delta = \frac{1}{T}$. Finally an OFDM signal transmitted on carrier frequency f_c is represented by

$$\begin{aligned} s(t) &= \mathrm{Re}\left\{x(t)\mathrm{e}^{j2\pi f_c t}\right\} \\ &= \mathrm{Re}\left\{\mathrm{e}^{j2\pi f_c t}\sum_{k=-\infty}^{\infty}\sum_{i=0}^{N-1} z_i^k \Pi\left(\frac{t-kT}{T}\right)\mathrm{e}^{j2\pi\frac{it}{T}}\right\}. \end{aligned} \tag{4.20}$$

An OFDM signal can be represented equivalently in the time domain as well as in the frequency domain. In the time domain, the signal can be viewed as the sum of several sines and cosines. Their periods are a multiple of the symbol period.

In the frequency domain, an OFDM signal can be interpreted as the sum of several sinc functions (frequency-domain representation of a rectangular pulse shape), which are spaced $f_\Delta = \frac{1}{T}$ apart (see figure 4.3). This particular separation distance ensures that the maximum of one sinc occurs exactly where all others are zero.

4.3.2 Guard interval

To combat the remaining interference between two consecutive symbols, we introduce a guard interval. We choose a cyclic prefix as guard interval to ease the channel estimation and equalization. A cyclic prefix is constructed by cyclically extending a symbol, i.e., by copying the last part of a symbol and placing it in front of the symbol (see figure 4.4).

To find a representation of a cyclically extended OFDM signal, we enlarge the symbol period

$$T = T_\mathrm{s} + T_\mathrm{g}, \tag{4.21}$$

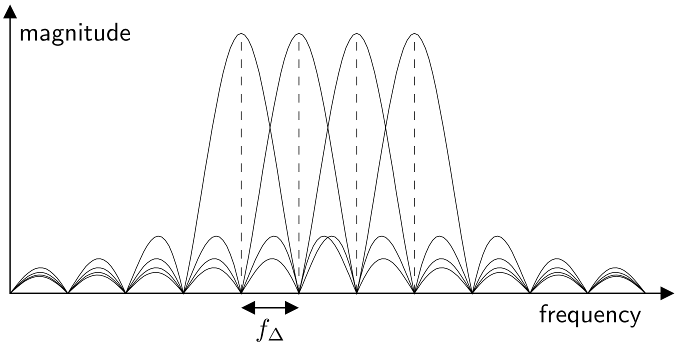


Figure 4.3. Overlapping power spectra of four sub-carriers. At the peak of a sub-carrier, the spectra of all other carriers are zero, illustrating their orthogonality.

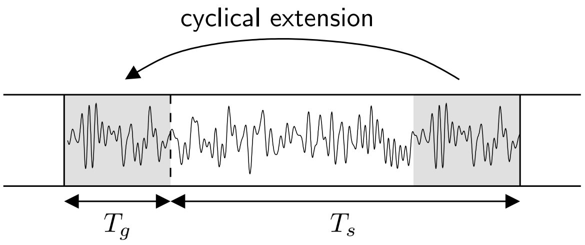


Figure 4.4. Construction of a cyclic prefix. The cyclical extension of the symbol acts as a guard interval against inter-symbol interference while preserving the orthogonality of the sub-carriers over the original symbol period T_s.

where T_s is the original symbol period and T_g is the duration of the cyclic prefix. Now, the complex envelope of a cyclically extended OFDM signal is given by

$$x(t) = \sum_{k=-\infty}^{\infty} \sum_{i=0}^{N-1} z_i^k \Pi\left(\frac{t-kT}{T}\right) \mathrm{e}^{j2\pi\frac{i(t-T_\mathrm{g})}{T_\mathrm{s}}}. \tag{4.22}$$

This leads to

$$s(t) = \mathrm{Re}\left\{\mathrm{e}^{j2\pi f_c t} \sum_{k=-\infty}^{\infty} \sum_{i=0}^{N-1} z_i^k \Pi\left(\frac{t-kT}{T}\right) \mathrm{e}^{j2\pi\frac{i(t-T_\mathrm{g})}{T_\mathrm{s}}}\right\}. \tag{4.23}$$

Here, the ith sub-carrier frequency is given by $f_i = \frac{i}{T_\mathrm{s}}$. The frequency separation is given by

$$f_\Delta = \frac{1}{T_\mathrm{s}}. \tag{4.24}$$

To completely eliminate all inter-symbol interference, one should choose a duration of the cyclic prefix that is longer than the multi-path delay spread T_m of the channel.

4.3.3 OFDM demodulation

The complex envelope of the received signal is represented by

$$r(t) = h(t) * x(t), \tag{4.25}$$

where the $*$ operator denotes the convolution with the channel impulse response.

Assuming ideal demodulation, i.e., a perfectly synchronized receiver, the demodulated data symbol $\tilde{z}_{i'}^{k}$ in the kth symbol and on the i' sub-carrier is given by

$$\begin{aligned}\tilde{z}_{i'}^{k} &= \frac{1}{T}\int_{kT+T_\mathrm{g}}^{(k+1)T} r(t)\mathrm{e}^{-j2\pi\frac{i'(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t \\ &= \frac{1}{T}\int_{kT+T_\mathrm{g}}^{(k+1)T} r_k(t)\mathrm{e}^{-j2\pi\frac{i'(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t,\end{aligned} \tag{4.26}$$

where

$$r_k(t) = h(t) * x_k(t) \tag{4.27}$$

is the complex envelope representation of the received signal in the interval $t \in [kT + T_\mathrm{g}, (k+1)T\rangle$. Here, $x_k(t)$ is given by

$$x_k(t) = \Pi\left(\frac{t-kT}{T}\right)\sum_{i=0}^{N-1} z_i^k \mathrm{e}^{j2\pi\frac{i(t-T_\mathrm{g})}{T_\mathrm{s}}}, \tag{4.28}$$

which is the complex envelope of the cyclically extended OFDM symbol in the time interval $t \in [kT, (k+1)T\rangle$. Equation 4.27 is only valid if the guard interval duration T_g is longer than the multi-path delay spread T_m of the channel.

Substitution of equation 4.27 in equation 4.26 gives

$$\begin{aligned}\tilde{z}_{i'}^{k} &= \frac{1}{T}\int_{kT+T_\mathrm{g}}^{(k+1)T} h(t) * x_k(t)\mathrm{e}^{-j2\pi\frac{i'(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t \\ &= \frac{1}{T}\int_{kT+T_\mathrm{g}}^{(k+1)T}\left[\int_{-\infty}^{\infty} h(\tau)x_k(t-\tau)\mathrm{d}\tau\right]\mathrm{e}^{-j2\pi\frac{i'(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t.\end{aligned} \tag{4.29}$$

Subsequent substitution of equation 4.28 leads to

$$\begin{aligned}\tilde{z}_{i'}^{k} &= \frac{1}{T}\int_{kT+T_\mathrm{g}}^{(k+1)T}\left[\int_{-\infty}^{\infty} h(\tau)\sum_{i=0}^{N-1} z_i^k \mathrm{e}^{j2\pi\frac{i(t-\tau-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}\tau\right]\mathrm{e}^{-j2\pi\frac{i'(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t \\ &= \frac{1}{T}\sum_{i=0}^{N-1} z_i^k \int_{kT+T_\mathrm{g}}^{(k+1)T}\left[\int_{-\infty}^{\infty} h(\tau)\mathrm{e}^{j2\pi\frac{i(t-\tau-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}\tau\right]\mathrm{e}^{-j2\pi\frac{i'(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t.\end{aligned} \tag{4.30}$$

Moving the term $\mathrm{e}^{j2\pi\frac{i(t-T_\mathrm{g})}{T_\mathrm{s}}}$ out of the inner integral gives

$$\begin{aligned}\tilde{z}_{i'}^{k} &= \frac{1}{T}\sum_{i=0}^{N-1} z_i^k \int_{kT+T_\mathrm{g}}^{(k+1)T}\left[\int_{-\infty}^{\infty} h(\tau)\mathrm{e}^{-j2\pi\frac{i\tau}{T_\mathrm{s}}}\mathrm{d}\tau\right]\mathrm{e}^{j2\pi\frac{(i-i')(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t \\ &= \frac{1}{T}\sum_{i=0}^{N-1} z_i^k \int_{-\infty}^{\infty} h(\tau)\mathrm{e}^{-j2\pi\frac{i\tau}{T_\mathrm{s}}}\mathrm{d}\tau \int_{kT+T_\mathrm{g}}^{(k+1)T}\mathrm{e}^{j2\pi\frac{(i-i')(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t \\ &= \frac{1}{T}\sum_{i=0}^{N-1} z_i^k H_i \int_{kT+T_\mathrm{g}}^{(k+1)T}\mathrm{e}^{j2\pi\frac{(i-i')(t-T_\mathrm{g})}{T_\mathrm{s}}}\mathrm{d}t,\end{aligned} \tag{4.31}$$

with

$$H_i = H\left(\frac{i}{T_\mathrm{s}}\right), \tag{4.32}$$

where

$$H(f) = \int_{-\infty}^{\infty} h(\tau)\mathrm{e}^{-j2\pi f\tau}\mathrm{d}\tau, \tag{4.33}$$

which is the frequency response of the channel (the Fourier-transformed channel impulse response).

Thus we arrive at

$$\tilde{z}_{i'}^{k} = z_i^k H_i, \tag{4.34}$$

where it is clearly seen that in an ideal receiver, the multi-path channel causes an amplitude attenuation and a phase rotation given by the frequency response of the channel. This effect can easily be compensated for, with a one-tap equalizer.

4.4 Optimal energy assignment

Multi-carrier systems such as OFDM allow the assignment of different energy levels to each sub-carrier. This enables pre-distortion of the transmitted sub-symbols in order to improve the performance. In [32], an optimization scheme is presented to maximize the overall bit rate in case the total transmitter power is limited. This is achieved by an optimal division of the available power amongst the sub-carriers, similar to the water-pouring solution known from information theory. In this chapter however, a different approach is taken. Given a power budget, the probability of a bit error is minimized, instead that the overall bit rate is maximized.

4.4.1 Channel model

Since the inter-symbol interference is effectively canceled, the individual sub-carriers do not suffer from frequency-selective fading. The remaining flat fading is modeled with a power attenuation factor C_i for every sub-carrier i. This factor is a piecewise constant approximation of the channel transfer function $H(f)$ (see equation 4.33) and is given by

$$C_i = |H(f_i)|^2, \tag{4.35}$$

where f_i is the central frequency of the ith sub-carrier.

In addition to frequency-selective fading, the channel also suffers from additive white Gaussian noise (AWGN) which limits the transmission capacity. The noise power N_0 is assumed constant and equal for all sub-carriers.

4.4.2 System model

On every sub-carrier i an average bit energy E_{b_i} is used to transmit the data symbols (see figure 4.5). The probability of a bit error on the ith sub-carrier is a function of the average bit energy, the power attenuation factor C_i, and the average noise power N_0

$$P_{e_i} = F\left(E_{b_i}, C_i, N_0\right). \tag{4.36}$$

In general, F is a strictly decreasing function with respect to E_{b_i}. The precise form depends on the applied symbol mapping scheme.

For the remaining part of this chapter, we assume a uniform modulation scheme for each sub-carrier. This assumption is made only for analytical convenience and does not limit the validity of the model.

The total energy transmitted over the channel E_b is the sum of the energies transmitted over all sub-carriers, i.e.,

$$E_b = \sum_{0}^{N-1} E_{b_i}, \tag{4.37}$$

where N is the number of sub-carriers. The average bit error rate is given by the mean of the bit error probabilities per sub-carrier

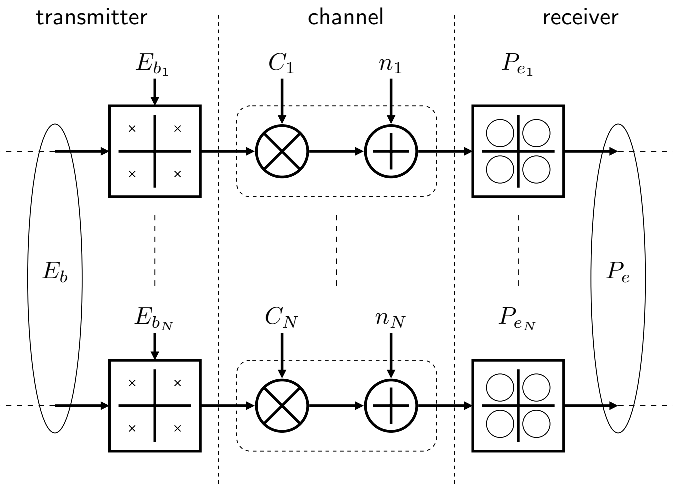


Figure 4.5. Model of a multi-carrier communication system.

$$P_e = \frac{1}{N} \sum_{0}^{N-1} P_{e_i}. \tag{4.38}$$

The total number of bits B transmitted depends on the number of bits per symbol b_i which, still assuming a uniform symbol mapping scheme for all sub-carriers, is given by

$$B = \sum_{0}^{N-1} b_i = N \cdot b, \tag{4.39}$$

where b is the number of bits per sub-carrier.

4.4.3 Optimization problem

With the definitions above, the following optimization problem can be formulated:

Find the average bit energies $E_{b_i} \forall\ i = 1 \ldots N$, with N the number of subchannels, such that the total energy E_b is minimized

$$\min \left[E_b = \sum_{1}^{N} E_{b_i} \right],$$

while the average bit error probability P_e does not exceed the desired error rate R_e

$$P_e = \frac{1}{N} \sum_{1}^{N} P_{e_i} \leq R_e.$$

Solving this optimization problem gives the minimum amount of energy needed to transmit B bits given a desired bit error rate R_e.

Since the individual error probability functions per subchannel P_{e_i} are strictly decreasing functions of the average bit energy per subchannel E_{b_i}, the overall average bit error probability P_e

is also a strictly decreasing function. An optimum is reached when the partial derivatives of the average bit error probability with respect to the average bit energy per subchannel are equal, that is

$$\frac{\partial P_e}{\partial E_{b_i}} = \frac{\partial P_e}{\partial E_{b_j}}. \tag{4.40}$$

The solution to equation 4.40 gives the relative energy distribution. To find the absolute energy per subchannel, we have to satisfy the additional boundary condition, i.e.,

$$P_e = \frac{1}{N}\sum_{1}^{N} P_{e_i} = R_e, \tag{4.41}$$

where R_e is the desired average bit error rate. Solving equations 4.40 and 4.41 simultaneously gives the optimal energy distribution to achieve the desired bit error rate at the lowest total energy cost.

4.4.3.1 QPSK optimization

In case of QPSK (quadrature phase shift keying) symbol modulation, we are able to derive a closed formula for the relative energy distribution. The error probability per subchannel P_{e_i} for QPSK symbol modulation is given by [30]

$$P_{e_i} = Q\left(\sqrt{2\frac{C_i E_{b_i}}{N_0}}\right). \tag{4.42}$$

Q is defined, as usual, by

$$Q(z) = \frac{1}{\sqrt{2\pi}}\int_z^{\infty} \mathrm{e}^{-\frac{1}{2}\lambda^2}\mathrm{d}\lambda. \tag{4.43}$$

The partial derivative with respect to the average energy per bit then boils down to

$$\frac{\partial P_e}{\partial E_{b_i}} = -\frac{1}{2}\sqrt{\frac{a_i}{\pi E_{b_i}}}\ \mathrm{e}^{-a_i E_{b_i}}, \tag{4.44}$$

with $a_i = \frac{C_i}{N_0}$. Substitution of equation 4.44 in equation 4.40 gives

$$\sqrt{\frac{a_i}{E_{b_i}}}\ \mathrm{e}^{-a_i E_{b_i}} = \sqrt{\frac{a_j}{E_{b_j}}}\ \mathrm{e}^{-a_j E_{b_j}}. \tag{4.45}$$

Squaring and rearrangement lead to

$$a_i E_{b_j}\ \mathrm{e}^{2a_j E_{b_j}} = a_j E_{b_i}\ \mathrm{e}^{2a_i E_{b_i}}. \tag{4.46}$$

Finally solving equation 4.46 for E_{b_j}, substituting $a_i = \frac{C_i}{N_0}$ and assuming a unity reference attenuation factor $C_i = 1$, gives the optimum normalized energy

$$\frac{\widehat{E}_{b_j}}{N_0} = \frac{1}{2C_j} W\left(2C_j^2 \frac{E_{b_i}}{N_0}\ \mathrm{e}^{2\frac{E_{b_i}}{N_0}}\right). \tag{4.47}$$

$W(x)$ is Lambert's W function, which obeys

$$W(x)\,\mathrm{e}^{W(x)} = x. \tag{4.48}$$

Equation 4.47 gives the relative energy distribution over the subchannels; the absolute values can be found by an iterative approximation of equation 4.41, using for example Newton–Raphson.

4.4.3.2 QAM-16 optimization

In case of QAM-16 (16-ary quadrature amplitude modulation) symbol modulation, we derive a numerical procedure to calculate the optimal energy distribution. The bit error probability per subchannel P_{e_i} for QAM-16 symbol modulation is given by

$$P_{e_i} = \frac{3}{4} Q\left(\sqrt{\frac{1}{5}\frac{C_i E_{b_i}}{N_0}}\right) + \frac{1}{4} Q\left(\sqrt{\frac{9}{5}\frac{C_i E_{b_i}}{N_0}}\right). \tag{4.49}$$

Setting the partial derivatives equal and solving the resulting equations, however, does not yield a simple analytical solution. Therefore, the solution is calculated numerically using the following procedure.

```
1. Assign $E_{b_i} \forall\, i = 1 \dots N$ such that $P_{e_i} = R_e$
2. Calculate $\left|\frac{\partial P_e}{\partial E_{b_i}}\right|_{E_{b_i}} \quad \forall\, i = 1 \dots N$
3. While $\left|\frac{\partial P_e}{\partial E_{b_i}}\right|_{E_{b_i}} \neq \left|\frac{\partial P_e}{\partial E_{b_j}}\right|_{E_{b_j}} \quad \forall\, i = 1 \dots N,\, j = 1 \dots N$
    (a) Select the subchannels with the largest and smallest partial derivatives $\left|\frac{\partial P_e}{\partial E_{b_i}}\right|_{E_{b_i}}$
    (b) Increase the energy for the subchannel with the largest partial derivative
    (c) Find the new $E_{b_i}$ for the subchannel with the smallest partial derivative while satisfying $P_e = R_e$
    (d) Calculate the new partial derivatives
```

The procedure above can be applied for any modulation scheme with a bit error probability P_{e_i} per subchannel of the form

$$P_{e_i} = \sum_{1}^{K} \alpha_k Q\left(\sqrt{\gamma_k \frac{C_i E_{b_i}}{N_0}}\right), \tag{4.50}$$

where K is an arbitrary integer and α_k and γ_k are modulation-dependent constants. The same procedure can also be applied for non-uniform symbol modulation schemes. If the bit error probability per subchannel P_{e_i} is a strictly decreasing function of E_{b_i}, then the procedure converges.

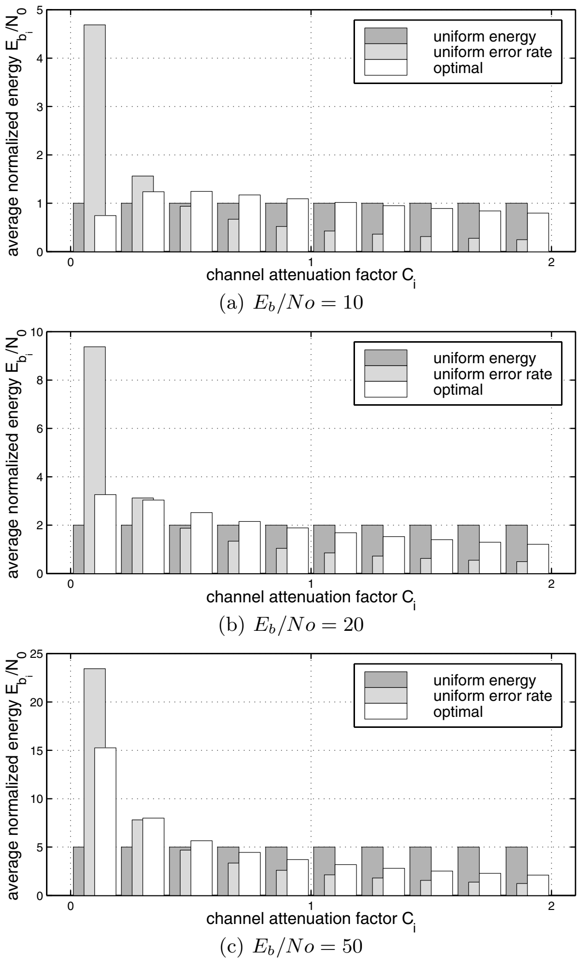
4.4.4 Comparison

To validate and demonstrate the effectiveness of the proposed optimal energy assignment scheme, we compare the method with two other widely used energy assignment schemes. The first is the traditional approach to assign all subchannels the same energy per bit (a uniform energy). The second scheme assigns the energies in such a way that all subchannels have the same error rate (a uniform error rate). In all cases, QPSK is used for the symbol modulation.

Obviously, the uniform energy assignment scheme does not take into account the channel conditions when assigning energy to each subchannel, whereas the uniform error rate assignment scheme does. However, the latter only locally optimizes the energy for each subchannel. The proposed optimal method globally optimizes the energy assigned to each subchannel, making maximum use of the channel characteristics and the available energy.

4.4.4.1 Energy distribution

Figure 4.6 shows the energy distributions for increasing energy levels for each of the assignment methods. From the graphs, it is observed that the optimal scheme assigns relatively less energy to the (poor) subchannels with low attenuation factors than the uniform error rate scheme and at the same time, it assigns more energy to the (good) subchannels with higher attenuation factors. The higher error rate for poor subchannels is compensated for by a lower error rate for good subchannels, resulting in a decrease of the average error rate.



(a) $E_b/No = 10$

(b) $E_b/No = 20$

(c) $E_b/No = 50$

Figure 4.6. Comparison of different energy distribution schemes for various values of the total normalized energy E_b/N_0.

For low energy levels, it is observed that the optimal scheme resembles the uniform energy scheme. For higher levels, it behaves more like a uniform error rate assignment scheme.

4.4.4.2 Performance

The three assignment schemes are compared by simulation, using a Ricean fading channel model [61]. For different numbers of subchannels, a large number of channel responses is generated representing different channel conditions. This is achieved by changing the Ricean K-factor, which represents the severity of multi-path effects in the channel. The lower the K-factor, the more the multi-path effects dominate. In case the Ricean K-factor equals zero, the Ricean fading channel model turns into a Rayleigh fading channel model.

The performance of the schemes is compared by plotting the average bit error rate P_e against the average normalized energy per subchannel E_{b_i}/N_0 (see figure 4.7, left-hand graphs, solid curves). Next to the average, the performance with the best and that with the worst channel conditions are shown (dashed curves). Additionally the energy gain G of the optimal scheme with respect to that of the other two schemes is plotted (figure 4.7, right-hand graphs).

In all cases, the simulations show that the scheme proposed in this paper performs best. The improvement is most significant with respect to the uniform energy distribution. The gain is moderate with respect to the uniform error rate distribution scheme. The most energy is gained (compare figure 4.7, right-hand graphs) when the channel conditions are worst, i.e., when the channel suffers from deep fades ($K \approx 0$).

4.5 Quantization level

To derive an analytical expression for the performance of quantized OFDM systems, we investigate the effects of quantization on a, apart from that, ideal modulator–demodulator pair. Performance degradation due to the limited precision of the arithmetic is not taken into account, but numerous references on this topic and strategies to reduce computational noise can be found in literature, e.g. [51, 83].

4.5.1 System model

The quantized OFDM system model under consideration is depicted in figure 4.8. It consists of a sub-symbol encoder and an inverse discrete Fourier transform (IDFT) connected to a forward discrete Fourier transform (DFT) and a sub-symbol decoder via a uniform quantizer Q. In this model, the extension of the OFDM symbol with a cyclic prefix can be neglected, as ideal time synchronization is assumed.

Channel effects and distortions, such as multi-path interference and propagation loss, are not taken into account; we only focus on the effects of quantization on the data transmission capabilities. For the same reason, we assume ideal time and frequency synchronization in the receiver. Hence, the expressions found in section 4.5.2 give a lower bound for the bit error probability of a quantized OFDM system.

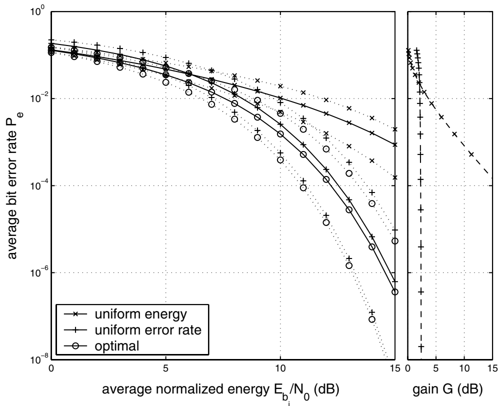
A fixed-point number consists of a number of bits, split into an integer part and a fractional part separated by the radix point (see figure 4.9). When uniform quantization is applied, the level distance d of the quantizer is determined by the number of fractional bits n

$$d = 2^{-n}. \tag{4.51}$$

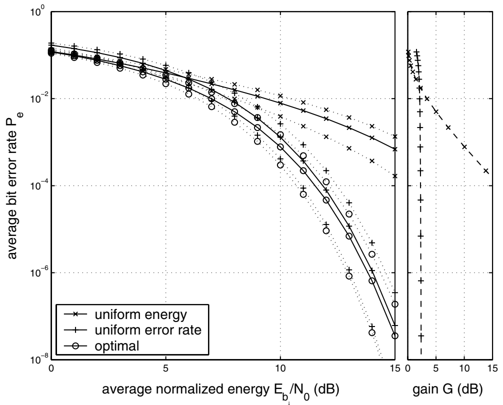
The quantization process distorts the modulated signal, causing errors in the demodulated data.

4.5.1.1 Quantization noise before transformation

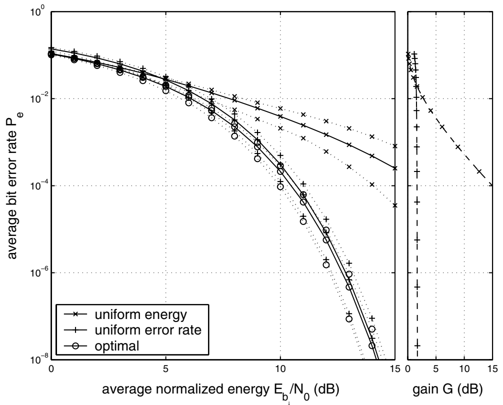
We approximate the effect of quantization of the OFDM signal by adding quantization noise (see figure 4.10). The noise is modeled as a complex uniform random variable n_Q



(a) $K = 0$ (Rayleigh fading)



(b) $K = 3$ dB



(c) $K = 6$ dB

Figure 4.7. Performance comparison of distribution schemes over a Ricean fading channel (K - Ricean K-factor, number of subchannels $N = 128$, number of simulations $n = 100$).

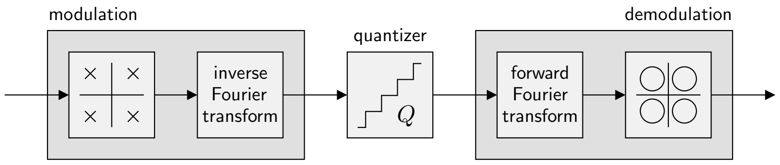


Figure 4.8. Model of a quantized OFDM system.

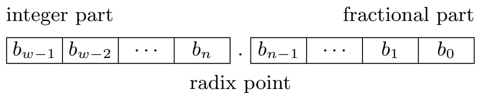


Figure 4.9. Fixed-point number with n fractional bits (b_i is the ith bit of a binary number with in total w digits).

$$n_Q = i_{n_Q} + jq_{n_Q}, \tag{4.52}$$

where i_{n_Q} and q_{n_Q} are the in-phase and quadrature components of the noise, respectively. The probability density function (PDF) $f_{n_Q}(i_{n_Q}, q_{n_Q})$ of n_Q is given by

$$f_{n_Q}(i_{n_Q}, q_{n_Q}) = \begin{cases} \left(\frac{1}{d}\right)^2 & \forall (i_{n_Q}, q_{n_Q}) : |i_{n_Q}| \leq \frac{d}{2} \text{ and } |q_{n_Q}| \leq \frac{d}{2}, \\ 0 & \text{elsewhere,} \end{cases} \tag{4.53}$$

where n is the number of fractional bits and d the level distance defined in equation 4.51.

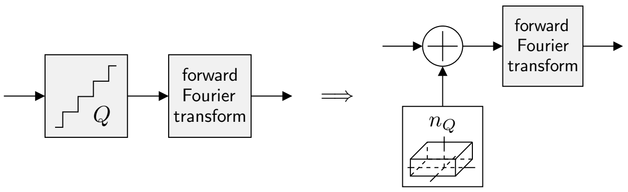


Figure 4.10. The quantization of the signal is modeled as the addition of quantization noise n_Q.

The model discussed so far does not enable us to easily deduce an expression for the performance of a quantized OFDM system. Therefore, we try to develop the model further to come to a more suitable framework.

4.5.1.2 Quantization noise after transformation

A quantized OFDM signal with K sub-carriers is demodulated using a K-point DFT. Because a Fourier transform is a linear operation, the addition of quantization noise can be moved after the DFT as long as the same transformation is applied to the noise as well.

The forward K-point DFT is given by

$$\widehat{x}(k) = \frac{1}{\sqrt{K}} \sum_{n=0}^{K-1} x(n) \mathrm{e}^{-\frac{j2\pi kn}{K}} \text{ for } k = 1 \ldots K. \tag{4.54}$$

Applying this transformation to the noise can be viewed as taking the weighted sum of K phase-rotated samples of n_Q. Therefore, we introduce two new random variables, n'_Q and ϕ, where n'_Q is equal to n_Q except for the random phase rotation ϕ

$$n'_Q = n_Q e^{\phi}. \tag{4.55}$$

Now, we substitute for the transformed quantization noise a weighted sum of random variable n'_Q as depicted in figure 4.11.

Summarizing, we have replaced the original uniform quantization noise n_Q before the DFT with a weighted sum of K samples of n'_Q after the DFT. Consequently, if K is large, the central limit theorem applies if

$$\mathrm{VAR}\big[n'_Q\big] > B_1 > 0 \tag{4.56}$$

and

$$\mathrm{E}\Big[\big|n'_Q - \mathrm{E}\big[n'_Q\big]\big|^3\Big] < B_2, \tag{4.57}$$

where B_1 and B_2 are positive numbers [54], and where E[·] and VAR[·] denote the expected value and variance of a random variable, respectively. Then, the resulting sum can be approximated by a Gaussian random variable (see figure 4.11). Hence, we introduce a new random variable N_Q with a Gaussian probability density function.

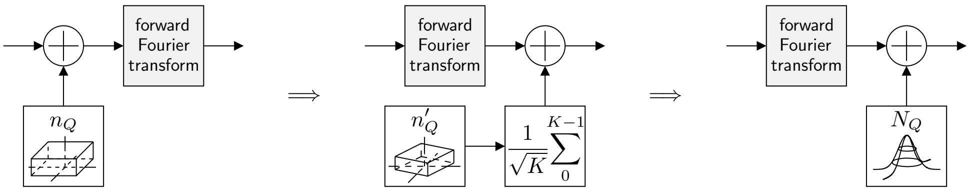

forward Fourier transform
n_Q
$\Longrightarrow$
forward Fourier transform
n'_Q
$\frac{1}{\sqrt{K}}\sum_{0}^{K-1}$
$\Longrightarrow$
forward Fourier transform
N_Q

Figure 4.11. The addition of uniform quantization noise n_Q before the Fourier transform can be modeled as the addition of Gaussian noise N_Q after the Fourier transform.

We now have arrived at a model with AWGN after transformation. In contrast to the model with uniform noise before demodulation, the new model allows us to easily derive an expression for the performance of the quantized OFDM system.

4.5.2 Performance of quantized OFDM systems

The noise after transformation is now modeled as a complex Gaussian random variable. Therefore, the performance of the quantized OFDM system can be calculated easily, because the consecutive inverse and forward DFTs are each other's inverse and both can thus be removed from the OFDM system model (see figure 4.12). Hence, the noise can be seen as AWGN directly added to the sub-symbols. Therefore, the noise performance of the sub-symbol modulation scheme determines the overall performance of the quantized OFDM system.

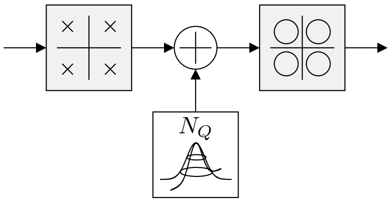


Figure 4.12. Collapsed model of a quantized OFDM system.

4.5.2.1 Analytical derivation

Usually, an analytical expression for the performance of the applied sub-symbol encoding scheme in the presence of AWGN is known. In order to apply this expression, we must find the properties of the Gaussian noise. Therefore, we have to derive the key parameters of the Gaussian variable N_Q, namely its mean μ_{N_Q} and its standard deviation σ_{N_Q}. To that end, we first calculate the mean and variance of the original uniform variable n_Q. Obviously, the mean of n_Q is zero

$$\mathrm{E}[n_Q] = 0, \tag{4.58}$$

and its variance is given by

$$\mathrm{VAR}[n_Q] = \mathrm{E}\left[{n_Q}^2\right] - \mathrm{E}[n_Q]^2 = \frac{1}{6}4^{-n}. \tag{4.59}$$

Because the mean of n_Q is zero and because a phase rotation does not influence the variance of a random variable with zero mean, the mean and variance of n'_Q are equal to the mean and variance of n_Q, thus

$$\mathrm{E}\left[n'_Q\right] = \mathrm{E}[n_Q] = 0 \tag{4.60}$$

and

$$\mathrm{VAR}\left[n'_Q\right] = \mathrm{VAR}[n_Q] = \frac{1}{6}4^{-n}. \tag{4.61}$$

Obviously, the variance of n'_Q is bounded and positive for $n \neq -\infty$, and the first condition (see equation 4.56) to apply the central limit theorem is met. Now, we only have to check the second condition (see equation 4.57), and to that end, we calculate the third central moment of n'_Q

$$\mathrm{E}\left[\left|n'_Q - \mathrm{E}\left[n'_Q\right]\right|^3\right] = \mathrm{E}\left[\left|n'_Q\right|^3\right] = \mathrm{E}\left[\left|n_Q\right|^3\right]. \tag{4.62}$$

An upper bound is given by

$$\mathrm{E}\left[\left|n_Q\right|^3\right] \leq \int_{-\frac{d}{2}}^{\frac{d}{2}} \int_{-\frac{d}{2}}^{\frac{d}{2}} \frac{1}{d^2} \left(\frac{d^2}{2}\right)^{\frac{3}{2}} \mathrm{d}x\mathrm{d}y = \frac{1}{2\sqrt{2}} 8^{-n}, \tag{4.63}$$

which is again bounded for $n \neq -\infty$. Thus, the mean and variance of N_Q can be calculated using the central limit theorem

$$\mu_{N_Q} = K \cdot \mathrm{E}\left[n'_Q\right] = 0. \tag{4.64}$$

Due to the weighting factor $\frac{1}{\sqrt{K}}$ in the sum, the variance of N_Q equals the variance of n_Q

$$\mathrm{VAR}[N_Q] = K \cdot \mathrm{VAR}\left[\frac{1}{\sqrt{K}} n_Q\right] = \frac{1}{6}4^{-n}, \tag{4.65}$$

which gives the standard deviation σ_{N_Q} of N_Q

$$\sigma_{N_Q} = \sqrt{\mathrm{VAR}[N_Q]} = \sqrt{\frac{1}{6}4^{-n}}. \tag{4.66}$$

As the one-dimensional noise power is given by $\frac{1}{2}{\sigma_{N_Q}}^2$, the bit error probability P_e for QPSK sub-symbol modulation can be calculated using

$$P_e = Q\left(\sqrt{\frac{E_b}{\frac{1}{2}{\sigma_{N_Q}}^2}}\right) = Q\left(\sqrt{12 E_b 4^n}\right), \tag{4.67}$$

where E_b is the average energy per bit. Likewise, for QAM-16 (16-symbol quadrature amplitude modulation), the probability of a bit error is

$$\begin{aligned} P_e &= \frac{3}{4}Q\left(\sqrt{\frac{2}{5}\frac{E_b}{\frac{1}{2}\sigma_{N_Q}{}^2}}\right) + \frac{1}{2}Q\left(\sqrt{\frac{18}{5}\frac{E_b}{\frac{1}{2}\sigma_{N_Q}{}^2}}\right) - \frac{1}{4}Q\left(\sqrt{10\frac{E_b}{\frac{1}{2}\sigma_{N_Q}{}^2}}\right) \\ &= \frac{3}{4}Q\left(\sqrt{\frac{24}{5}E_b 4^n}\right) + \frac{1}{2}Q\left(\sqrt{\frac{216}{5}E_b 4^n}\right) - \frac{1}{4}Q\left(\sqrt{120 E_b 4^n}\right). \end{aligned} \tag{4.68}$$

The Q-function is defined, as usual, by

$$Q(z) = \frac{1}{\sqrt{2\pi}}\int_z^\infty \mathrm{e}^{-\frac{1}{2}\lambda^2}\mathrm{d}\lambda. \tag{4.69}$$

With equations 4.67 and 4.68, we have deduced analytical expressions for the performance of quantized OFDM systems applying QPSK and QAM-16 sub-symbol modulation schemes, respectively, using the model depicted in figure 4.12. The same model and the same performance derivations can be applied for other sub-symbol encoding schemes as well.

4.5.3 Verification

In figure 4.13, the bit error rates (dotted lines) of several quantized OFDM systems are plotted against the number of fractional bits with a unity average energy per bit ($E_b = 1$) for QPSK and QAM-16 sub-symbol modulation and different numbers of sub-carriers. These curves were found using Monte Carlo simulations. Also plotted are the derived bit error probabilities (solid lines).

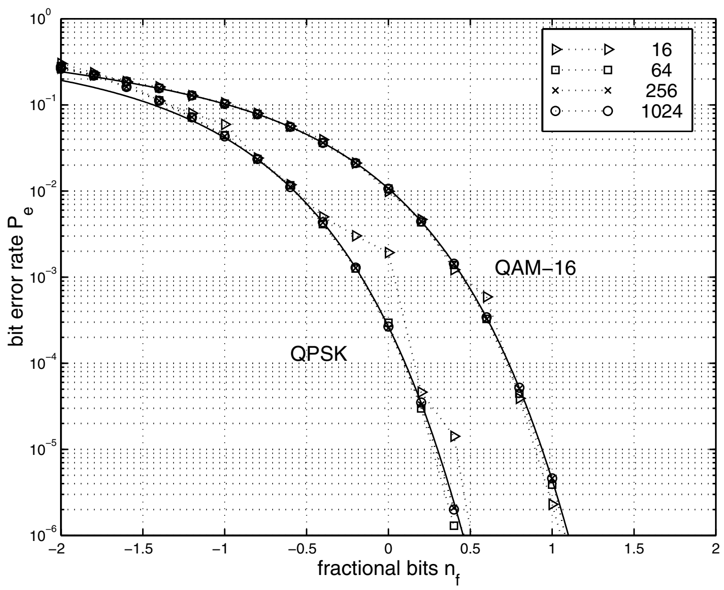


Figure 4.13. Performance of quantized OFDM systems ($E_b = 1$).

The predicted bit error probabilities match the simulation results well, suggesting that our assumptions and derivations are valid. Only for strongly quantized systems (region where $n < -1$)

the derived curves diverge from the simulation results. Additional simulations have shown that for severely quantized systems, the noise can no longer be modeled as a uniform random variable. In fact, the quantization noise for heavily quantized OFDM systems has distribution that is Gaussian instead of the assumed uniform one. Thus, the derived expressions for the performance no longer apply. However, the regions of interest for most systems lie well away from these extremes.

4.6 Clipping level

To derive an analytical expression for the performance of soft-limited OFDM systems, we apply the same procedure as that presented in the previous section, but now for soft-clipped OFDM systems. A number of authors have also investigated the performance of clipped OFDM (and other non-linear distortions) [6, 7, 50], but they take a different approach, as they limit the envelope of the complex OFDM signal, whereas here both the real and imaginary signal parts are limited in amplitude.

4.6.1 System model

The clipped OFDM system model under consideration is depicted in figure 4.14. The difference with the model presented in section 4.5.1 is that the quantizer is replaced by a soft limiter L.

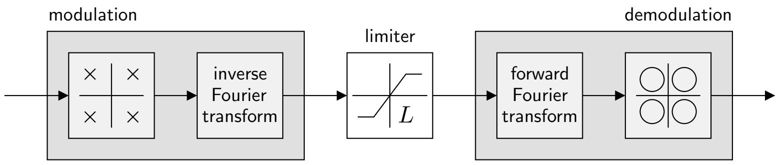


Figure 4.14. Model of a soft-limited OFDM system.

Again channel effects and distortions, such as multi-path interference and propagation loss, are not taken into account, in order to focus only on the effects of limiting the signal on the data transmission capabilities.

If a signal is represented by a fixed-point number, its maximum amplitude is determined by the number of bits used for the representation of the integer part (see figure 4.9). The clipping level c is given by

$$c = 2^{(n_i - 1)}, \tag{4.70}$$

where n_i is the number of integer bits used.

4.6.2 Noise modeling

In general, one cannot model the limiting of a signal straightforwardly by using an equivalent of quantization noise. Nevertheless, as an OFDM signal with random data sub-symbols has a complex Gaussian distribution if the number of sub-carriers is large (central limit theorem), we assume that the limiting can be modeled as a combination of attenuating the signal and adding clipping noise (see figure 4.15).

This assumption is loosely based on Bussgang's theorem [69] (a special case of Price's theorem [60]), which states that applying a Gaussian input to a non-linearity, such as a soft limiter, results in an attenuated version of the input signal plus an additional noise term.

Following the same reasoning as in section 4.5.1.2, we transform the clipping noise into Gaussian noise. Because the Fourier transform is a linear operation, we can easily move the attenuation factor α_c after the DFT. The resulting model is shown in figure 4.16.

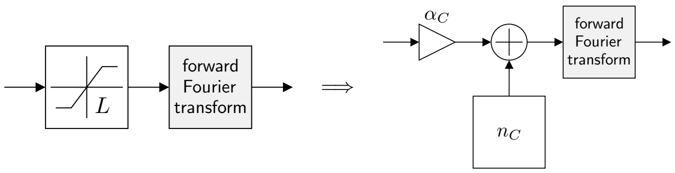


Figure 4.15. The limiting of the signal is modeled as a combination of multiplying the signal with an attenuation factor α_c and adding clipping noise n_C.

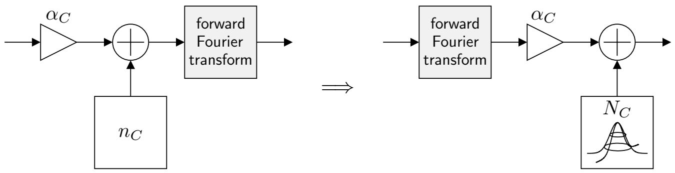


Figure 4.16. The multiplication with an attenuation factor and the addition of clipping noise n_C before the Fourier transform can be replaced by the same multiplication and the addition of the transformed clipping noise N_C after the Fourier transform.

4.6.3 Performance of soft-limited OFDM systems

The effects of limiting the OFDM signal have been modeled as the multiplication with an attenuation factor α_C and the addition of a complex Gaussian noise term N_C. The resulting model is depicted in figure 4.17, where the inverse and forward DFTs are removed from the system model.

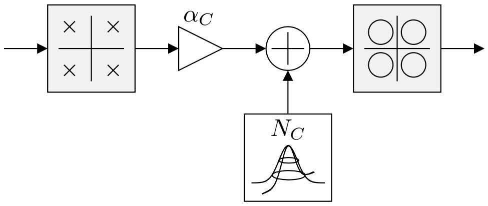


Figure 4.17. Collapsed model of a soft-limited OFDM system.

4.6.3.1 Analytical derivation

To find an analytical expression for the performance of a soft-limited OFDM system, we must find the properties of the Gaussian noise. To derive the mean μ_{N_C} and its standard deviation σ_{N_C} we first calculate the mean and variance of the clipping noise n_C.

The PDF of an OFDM signal $f_s(i_s, q_s)$ is Gaussian and therefore given by

$$f_s(i_s, q_s) = \frac{1}{\sqrt{2\pi}\sigma_s} \mathrm{e}^{-\frac{{i_s}^2 + {q_s}^2}{2{\sigma_s}^2}}, \tag{4.71}$$

where $2{\sigma_s}^2$ is the energy of the signal and i_s and q_s its in-phase and quadrature components, respectively. For simplicity we assume symmetrical clipping, i.e., the OFDM signal is limited to values between $-c$ and c, where c is the clipping level. As a consequence, the mean of the clipping noise n_C equals zero

$$\mathrm{E}[n_C] = 0. \tag{4.72}$$

The variance can be calculated using

$$\begin{aligned}\mathrm{VAR}[n_C] =& \mathrm{E}\left[{n_C}^2\right] - \mathrm{E}[n_C]^2\\ =& \frac{4}{\sqrt{2\pi}\sigma_s}\int_c^\infty\int_c^\infty \left[(x-c)^2+(y-c)^2\right]\mathrm{e}^{-\frac{x^2+y^2}{2{\sigma_s}^2}}\mathrm{d}x\mathrm{d}y\,+\\ & \frac{4}{\sqrt{2\pi}\sigma_s}\int_{-c}^{c}\int_c^\infty (x-c)^2\mathrm{e}^{-\frac{x^2+y^2}{2{\sigma_s}^2}}\mathrm{d}x\mathrm{d}y\\ =&4\left({\sigma_s}^2+c^2\right)Q\!\left(\frac{c}{\sigma_s}\right)-2\sqrt{\frac{2}{\pi}}\sigma_s c\mathrm{e}-\frac{c^2}{2{\sigma_s}^2}.\end{aligned} \tag{4.73}$$

Here, the first integral calculates the noise power if both the in-phase and quadrature components are clipped. The second term calculates the power when either the in-phase or the quadrature component is limited.

Now, the mean μ_{N_C} and standard deviation σ_{N_C} are easily determined

$$\mu_{N_C} = \mathrm{E}[n_C] = 0 \tag{4.74}$$

and

$$\sigma_{N_C} = \sqrt{\mathrm{VAR}[n_C]} = \sqrt{4\left({\sigma_s}^2+c^2\right)Q\!\left(\frac{c}{\sigma_s}\right)-2\sqrt{\frac{2}{\pi}}\sigma_s c\mathrm{e}^{-\frac{c^2}{2{\sigma_s}^2}}}. \tag{4.75}$$

The only thing left is to find an expression for the attenuation factor α_C. If the noise power $\mathrm{VAR}[n_C]$ is small with respect to the signal power $\mathrm{VAR}[s]$, this factor can be approximated by the square root of the power ratio of the limited signal and the original signal

$$\alpha_C \approx \sqrt{\frac{\mathrm{VAR}[s_C]}{\mathrm{VAR}[s]}}. \tag{4.76}$$

Here, $\mathrm{VAR}[s]$ is the variance of the OFDM signal, which is given by

$$\mathrm{VAR}[s] = 2{\sigma_s}^2. \tag{4.77}$$

The variance of the clipped signal $\mathrm{VAR}[s_C]$ can be calculated with

$$\begin{aligned}\mathrm{VAR}[s_C] =& \frac{1}{\sqrt{2\pi}\sigma_s}\int_{-c}^{c}\int_{-c}^{c}\left(x^2+y^2\right)\mathrm{e}^{-\frac{x^2+y^2}{2{\sigma_s}^2}}\mathrm{d}x\mathrm{d}y+\\ & \frac{4}{\sqrt{2\pi}\sigma_s}\int_c^\infty\int_c^\infty 2c^2\mathrm{e}^{-\frac{x^2+y^2}{2{\sigma_s}^2}}\mathrm{d}x\mathrm{d}y+\\ & \frac{4}{\sqrt{2\pi}\sigma_s}\int_c^\infty\int_{-c}^{c}\left(x^2+c^2\right)\mathrm{e}^{-\frac{x^2+y^2}{2{\sigma_s}^2}}\mathrm{d}x\mathrm{d}y\\ =&4c^2Q\!\left(\frac{c}{\sigma_s}\right)+2{\sigma_s}^2\left[1-2Q\!\left(\frac{c}{\sigma_s}\right)\right]-2\sqrt{\frac{2}{\pi}}\sigma_s c\mathrm{e}^{-\frac{c^2}{2{\sigma_s}^2}}.\end{aligned} \tag{4.78}$$

The first integral represents the power in the undistorted parts of the signal, the second the power if both the in-phase and quadrature components are limited to c, and the third the power if only one of the components is clipped.

Substituting equations 4.77 and 4.78 into equation 4.76 gives

$$\alpha_C \approx \sqrt{1+2\left(\frac{c^2}{{\sigma_s}^2}-1\right)Q\!\left(\frac{c}{\sigma_s}\right)+\sqrt{\frac{2}{\pi}}\frac{c}{\sigma_s}\mathrm{e}^{-\frac{c^2}{2{\sigma_s}^2}}}. \tag{4.79}$$

With the found expressions for σ_{N_C} and α_C, we can calculate the error probability for P_e limited OFDM systems with QPSK sub-symbol modulation as follows

$$P_e = Q\left(\sqrt{\frac{{\alpha_c}^2 E_b}{\frac{1}{2}{\sigma_{N_C}}^2}}\right). \tag{4.80}$$

Here, $\frac{1}{2}{\sigma_{N_C}}^2$ is the one-dimensional noise power. Similar expressions can be found for QAM-16 sub-symbol modulation and other modulation schemes.

4.6.4 Verification

In figure 4.18, the bit error rates (dotted lines) of several soft-limited OFDM systems are plotted against the number of integer bits n_i with a unity average energy per bit ($E_b = 1$) for QPSK and QAM-16 sub-symbol modulation and different numbers of sub-carriers. These curves were found using Monte Carlo simulations. Also plotted are the derived bit error probabilities (solid lines).

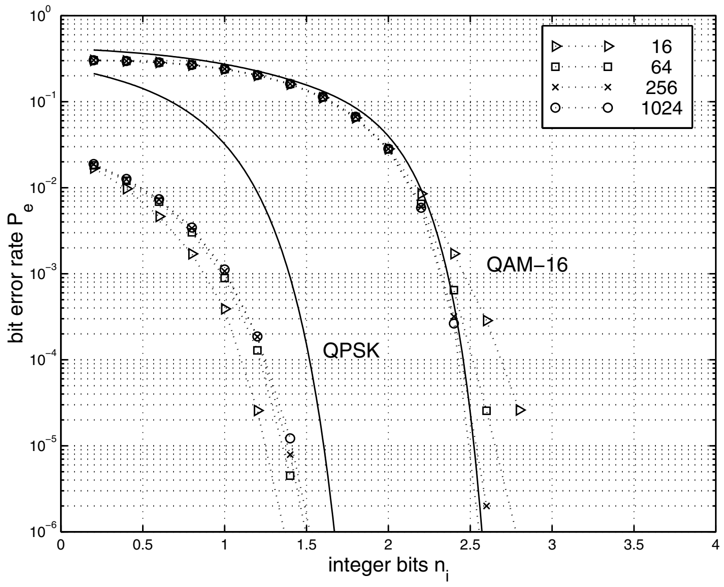


Figure 4.18. Performance of soft-limited OFDM systems ($E_b = 1$).

Clearly, the derived expressions match the simulation results very well for OFDM systems with QAM-16 sub-symbol modulation. However, the predicted error rates strongly deviate from the simulated error rates for QPSK sub-symbol modulation, especially for low numbers of integer bits. Obviously, the assumption that soft limiting can be modeled as an attenuation factor and the addition of noise is invalid for extreme clipping levels, because at these levels the clipping signal almost equals the original signal.

4.7 Summary

This chapter started with an introduction of the problems encountered in a multi-path channel, which is common to high frequency and high symbol rate radio transmission. Multi-carrier communication was introduced as a means to overcome the frequency-selective fading caused by the multi-path. Subsequently, OFDM was introduced as a particular multi-carrier communication scheme.

As this book focuses on power optimizations, among others, we investigated a method to use the energy assigned to the sub-carriers more efficiently. Furthermore, reckoning with a fixed-point implementation of the OFDM transceiver, we investigated the effects of such an implementation on the performance of the transceiver. The results can be used to determine the minimum number of bits necessary to implement an OFDM transceiver given a desired bit error rate.

Now that we have a better understanding of multi-carrier communications in general, and OFDM in particular, the next chapter will describe the implementation of an OFDM transceiver. Its design will serve as a test case, in order to test and verify the design automation tools developed in chapter 3.

5

Application

5.1 Introduction

To verify the completeness of the design flow and the correct operation of the automation tools, we need a representative application to serve as a test case. As already mentioned in section 1.3, the baseband processing for an OFDM transceiver was selected. The transceiver is used in an experimental setup to supply a mobile user with augmented reality, as envisioned in the Ubiquitous Communications program (see appendix A).

Because the baseband modulator and demodulator are not fully specified, we first have to determine the key communication parameters, such as the number of sub-carriers, and the sub-symbol modulation and multiple access schemes. Each time, the consequences of a particular decision have to be evaluated, in terms of the cost and performance of the resulting transceiver.

Subsequently, several alternatives for the baseband modulation and demodulation are described in C. This code serves as a starting point for the design flow presented in section 1.1.3, applying the hardware–software partitioning and code generation tools described in chapter 3. Finally, the results are presented and discussed.

5.2 Transceiver specification

As already mentioned in appendix A, a number of design choices have already been made for the UbiCom scenario. These include the use of OFDM on a 17 GHz radio channel. To fully specify a mobile transceiver, additional choices have to be made. These include the key parameters for the OFDM transceiver, such as the number of sub-carriers and the frequency separation, and the multiple-access and duplexing schemes.

To make a well-founded decision, we have to characterize the radio channel accurately and specify the user requirements. Then, the key communication parameters for the transceiver can be deduced. Throughout this process, the effects on the power consumption and other performance measures should be taken into account.

5.2.1 Frequency band

As described in section A.3, for the UbiCom project initially a target carrier frequency was chosen of roughly 17 GHz. In the frequency range from 16 to 18 GHz, only two bands are available, according to international regulations: the 17.1–17.2 GHz band, and the 17.2–17.3 GHz band.[1]

[1] Near 17 GHz, no ISM bands (bands internationally reserved for non-commercial use for industrial, scientific and medical purposes) are available. The nearest ISM bands are the 5725.0–5875.0 MHz band and the 24.0–24.25 GHz band.

In the Netherlands, these bands are freely available without license for radio local area networks (RLANs) applying short range devices (SRDs), according to the Dutch national frequency register [48]. In Europe, the bands are recommended to be allocated to HIPERLANs (high-performance RLANs) [24], starting from the year 2008 [22].

For UbiCom the first band was chosen, which means that the center carrier frequency is $f_c = 17.15\,\text{GHz}$ and the total available bandwidth $B = 100\,\text{MHz}$.

5.2.1.1 Radio channel measurements

To characterize the 17 GHz radio channel an extensive measurement campaign was conducted by A. Bohdanowicz [11, 12]. Wideband channel measurements were conducted in both indoor and outdoor environments.

- *Office building:* Indoor measurements were performed in an office building at several locations, a.o. in office rooms, corridors and in a large room (canteen). Both line-of-sight (LOS) and obstructed, non-line-of-sight (OBS) cases were studied.
- *Parking area:* The outdoor measurements were performed at two parking areas: one without large obstacles and one with trees surrounding it. In both cases, line-of-sight was preserved.

The main results of the RMS delay spread measurements are shown in figures 5.1, 5.2, and 5.3 (courtesy of A. Bohdanowicz).

The typical ranges of the delay spread values were calculated for different confidence intervals. These, together with the mean, are listed in table 5.1.

Table 5.1. Measured values for the root mean square (RMS) delay spread τ_{RMS} (in ns) for various indoor and outdoor, line-of-sight (LOS) or obstructed (OBS) environments

Environment	Place	1%	5%	95%	99%	Mean
Indoor (LOS)	CorridorLB	1.97	3.55	28.57	36.42	14.28
	Corridor16	6.29	8.20	25.88	27.61	15.28
	Canteen	2.45	4.85	29.14	40.81	13.91
	OfficeRoomA	2.25	3.17	7.67	7.77	5.42
	OfficeRoomB	3.82	5.34	9.99	12.58	7.73
	Overall	2.54	3.90	25.30	32.69	11.32
Indoor (OBS)	Corridors16	3.82	4.35	10.47	11.54	7.56
	Rooms16	3.82	4.59	11.07	11.75	7.54
	Overall	3.82	4.54	10.67	11.75	7.55
Outdoor (LOS)	Parking A	12.74	14.41	49.14	55.37	29.63
	Parking B	1.90	4.48	29.39	31.59	20.92
	Overall	3.71	5.27	44.32	52.85	25.28

5.2.2 Gross bit rate

The refresh rate of the displayed graphics and the required quality depend heavily on the distance of the user from the displayed virtual object. As nearby objects show more detail than those far away, usually the closer the user is to the virtual object, the more detail he will require.

Usually the requirements become more stringent when a user is close to a virtual object, as a more detailed image or graphic is required, to be acceptable to the user.

A detailed study [52] has been conducted on the effects of various level of detail (LOD) rendering variants on the required link capacity, taking into account effects like the distance of the user from the virtual object, and the lifetime of the object. Overall, for a graphic quality acceptable to the user, a gross bit rate R_b is needed ranging from 1 kbit/s to 10 Mbit/s.

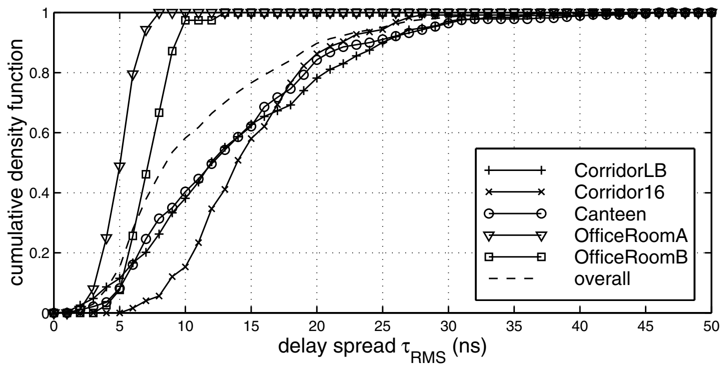


Figure 5.1. Cumulative density function of the root mean square (RMS) delay spread τ_{RMS} for various indoor line-of-sight (LOS) environments.

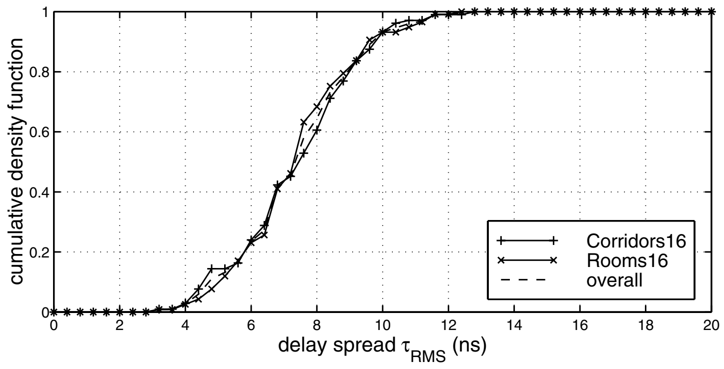


Figure 5.2. Cumulative density function of the root-mean-square (RMS) delay spread τ_{RMS} for various indoor obstructed, non-line-of-sight (OBS) environments.

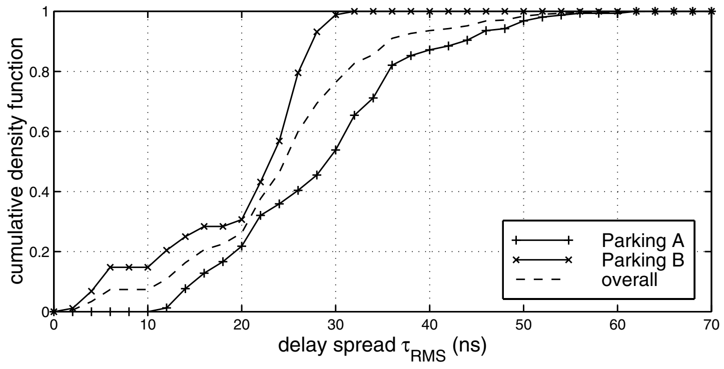


Figure 5.3. Cumulative density function of the root-mean-square (RMS) delay spread τ_{RMS} for various outdoor, line-of-sight (LOS) environments.

5.2.3 Symbol format

Based on the user requirements and the channel characteristics, the format of an OFDM symbol can be specified. A proper symbol format ensures robustness against the multi-path distortions and temporal distortions of the channel.

5.2.3.1 Inter-symbol interference

To cancel all inter-symbol interference, the guard interval should be longer than the multi-path delay spread of the channel

$$T_{\mathrm{g}} \geq T_{\mathrm{m}}. \tag{5.1}$$

In order to prevent too large a reduction in capacity, one must choose the symbol period T_{s} with useful information to be at least a few times longer than the guard interval T_{g}, as the guard interval does not contain useful information. For a symbol efficiency of $\eta = \frac{T_{\mathrm{s}}}{T_{\mathrm{s}}+T_{\mathrm{g}}}$, the symbol period should be at least

$$T_{\mathrm{s}} \geq \frac{\eta}{1-\eta}\, T_{\mathrm{g}}. \tag{5.2}$$

From equation 4.24 we have the sub-carrier frequency separation

$$f_{\Delta} = \frac{1}{T_{\mathrm{s}}}. \tag{5.3}$$

Substitution of equations 4.6, 5.1, and 5.2 leads to

$$f_{\Delta} \leq \frac{1-\eta}{\gamma\eta}\frac{1}{\tau_{\mathrm{RMS}}}. \tag{5.4}$$

If the total bandwidth available for a user equals B_{u}, then the minimum number of sub-carriers N to meet the efficiency demand is given by

$$N = \frac{B_{\mathrm{u}}}{f_{\Delta}}. \tag{5.5}$$

Substitution of equation 5.4 gives a lower bound to the number of sub-carriers

$$N \geq \frac{\gamma\eta}{1-\eta} B_{\mathrm{u}} \cdot \tau_{\mathrm{RMS}}. \tag{5.6}$$

5.2.3.2 Coherence bandwidth

To determine whether the individual sub-carriers can be considered narrow-band, we calculate the coherence bandwidth B_{c} of the channel. The coherence bandwidth is approximated with the reciprocal of the multi-path delay spread of the channel [61]

$$B_{\mathrm{c}} \approx \frac{1}{T_{\mathrm{m}}}. \tag{5.7}$$

To prevent frequency-selective fading within a sub-carrier, one should choose the frequency separation smaller than the coherence bandwidth

$$f_{\Delta} \leq B_{\mathrm{c}}. \tag{5.8}$$

This again leads to a lower bound to the number of sub-carriers N

$$N \geq \frac{B}{B_{\mathrm{c}}}. \tag{5.9}$$

Substitution of equations 5.7 and 4.6 gives

$$N \geq \gamma B_{\mathrm{u}} \cdot \tau_{\mathrm{RMS}}. \tag{5.10}$$

It should be noted that for $\eta > \frac{1}{2}$, equation 5.6 specifies a more strict requirement.

5.2.3.3 Coherence time

The Doppler spread of a channel is defined by [61]

$$B_{\mathrm{d}} = \frac{v_{\mathrm{o}}}{\lambda_{\mathrm{c}}} = v_{\mathrm{o}} \frac{f_{\mathrm{c}}}{c}, \tag{5.11}$$

where v_{o} is the maximum speed of objects interfering with the radio beam, c is the speed of light, and $\lambda_{\mathrm{c}} = \frac{c}{f_{\mathrm{c}}}$ the wavelength of the carrier.

The reciprocal of B_{d} is a measure of the coherence time t_{c} of the channel. That is,

$$t_{\mathrm{c}} \approx \frac{1}{B_{\mathrm{d}}}. \tag{5.12}$$

The relative coherence time τ_{c} with respect to the symbol period is defined by

$$\tau_{\mathrm{c}} = \frac{t_{\mathrm{c}}}{T}. \tag{5.13}$$

It indicates how rapidly the channel conditions are changing and can be used to determine how often a channel estimate should be updated for proper channel equalization.

5.2.4 Sub-symbol encoding scheme

The sub-symbol encoding scheme determines the amount of bits transmitted per sub-carrier. The data bits can be encoded in the phase or the amplitude of the symbol, or in a combination of both. As a result, all variants of PSK (phase shift keying), ASK (amplitude shift keying) and QASK (quadrature amplitude shift keying), also known as QAM (quadrature amplitude modulation), are applicable.

The gross bit rate is the total number of bits transmitted on all sub-carriers per symbol period

$$R_b = \frac{1}{T} \sum_{i=0}^{N-1} \ell_i, \tag{5.14}$$

where ℓ_i is the number of bits transmitted on the ith sub-carrier.

With a uniform sub-symbol encoding scheme, i.e., when the same encoding scheme is used for all sub-carriers $\ell_i = \ell$, equation 5.14 simplifies to

$$R_b = \frac{N \cdot \ell}{T}, \tag{5.15}$$

where ℓ is the number of bits per sub-symbol. It is directly related to the number of possible symbols M of the encoding scheme via $M = 2^{\ell}$, or, equivalently, $\ell = \log_2 M$.

5.2.4.1 Channel estimation and equalization

As seen in section 4.3.3, the effects of the multi-path channel ultimately translate into multiplying the transmitted sub-symbols by the frequency response of the channel. As a consequence, if the frequency response of the channel is known, the equalization can be performed by multiplying by the inverse of the channel's frequency response.

To estimate the frequency response, one can transmit known sub-symbols. Calculating the phase rotation and amplitude attenuation of these symbols gives the frequency response of the channel.

For differential sub-symbol encoding schemes, it is not necessary to estimate and equalize the channel, as the data is encoded in the difference between sub-symbols; so coherent decoding is not necessary.

5.2.5 Time and frequency synchronization

Only a demodulator that has been synchronized to the modulator, both in time and in frequency, can correctly demodulate an OFDM symbol. Especially, frequency synchronization is crucial, as a slight offset already disrupts the orthogonality of the sub-carriers, as depicted in figure 5.4. Two effects occur because of the frequency offset: one is the reduction of signal strength of the desired sub-carrier, and the second is the introduction of inter-carrier interference from the other sub-carriers. A worse bit error rate is the result [55].

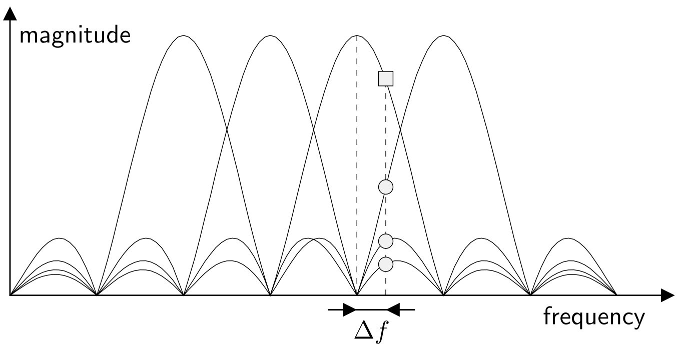


Figure 5.4. Loss of orthogonality due to a frequency offset Δf. Illustrated is the power spectrum of four sub-carriers, where a frequency offset causes a reduction in the signal strength of the desired sub-carrier (□) and introduces interference between the sub-carriers (○).

The impact of a small timing offset is less severe, as it results in a progressive phase rotation of the demodulated symbols. In coherent sub-symbol demodulation, this can be detected and compensated for by properly adjusting the equalization. Differential demodulation is robust against these phase rotations, although they can lead to a performance degradation.

Numerous synchronization schemes are known from literature, e.g. [8, 47, 74]. A comprehensive overview can be found in [77]. Two main approaches are identified:

- Demodulators that extract the information needed for synchronization before sub-carrier demodulation, either using a training sequence or using the specific structure of an OFDM symbol
- Demodulators that extract the information *after* the demodulation of the sub-carriers using special data symbols or symbol patterns

To keep the synchronization overhead to a minimum, we chose the synchronization method suggested by Witrisal [86]. The method utilizes a training symbol constructed by modulating the odd sub-carriers with a PN (pseudo-random noise) sequence, while setting the even sub-carriers to zero. This leads to a symbol with, apart from the sign, identical halves. This training symbol is used to simultaneously estimate the timing and frequency offsets.

5.2.6 Multiple access

In the UbiCom scenario, multiple users are envisioned, possibly interacting with each other. As a consequence, a base station has to serve more than one user, and it has to divide the available communication resources, i.e., it has to implement the multiple access of the radio channel.

In principle, OFDM supports all traditional multiple-access methods, such as TDMA (time-division multiple access) and FDMA (frequency-division multiple access). Another possibility is to assign different sub-carriers to different users, a technique known as OFDMA (orthogonal FDMA).

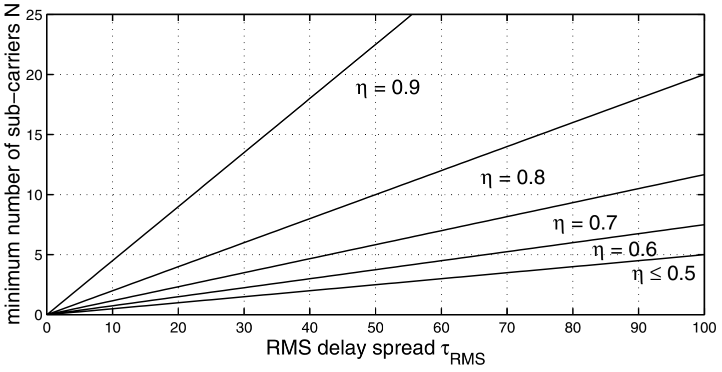


Figure 5.5. Minimum number of sub-carriers N as a function of the root mean square (RMS) delay spread τ_{RMS} (channel model constant $\gamma = 5$, bandwidth $B_{\mathrm{u}} = 10\,\mathrm{MHz}$).

The last scheme, however, requires the mobile device to demodulate all the sub-carriers in order to retrieve the data of only a few, thus wasting energy. Furthermore, the higher the number of sub-carriers, the more computing power is needed for the frequency multiplexing.

From equation 5.5 it is clear that the number of sub-carriers is restricted by reducing the bandwidth per user B_{u}. Increasing the carrier frequency separation f_Δ is not possible, since its maximum value is determined by the channel characteristics.

Whereas with TDMA, a user gets the whole available bandwidth assigned; with FDMA, he gets only a fraction of the total bandwidth. Hence, with FDMA, the number of sub-carriers is less than with TDMA. Furthermore, in a TDMA scheme, the user has to wait for his turn to transmit and receive, decreasing the responsiveness of the mobile device. Therefore, we chose FDMA. To prevent cross-talk between transmitter and receiver, time-division duplexing (TDD) is applied as a duplexing scheme.

In section A.3.2, the minimum number of users per base station N_{u} is given

$$N_{\mathrm{u}} \geq 10. \tag{5.16}$$

The total bandwidth is equally divided over the users. Thus,

$$B_{\mathrm{u}} = \frac{B}{N_{\mathrm{u}}} \leq 10\,\mathrm{MHz}, \tag{5.17}$$

gives an upper limit to the total bandwidth available to a user.

5.2.7 Summary

As both conditions formulated in equations 5.6 and 5.10 have to be fulfilled, the minimum number of sub-carriers is given by

$$N \geq \begin{cases} \gamma B_{\mathrm{u}} \cdot \tau_{\mathrm{RMS}} & \eta \leq \frac{1}{2}, \\ \frac{\eta}{1-\eta}\gamma B_{\mathrm{u}} \cdot \tau_{\mathrm{RMS}} & \text{otherwise.} \end{cases} \tag{5.18}$$

This relationship is depicted in figure 5.5.

Using $T_{\mathrm{s}} = \frac{N}{B}$ and $T = \frac{1}{\eta}T_{\mathrm{s}}$ gives the maximum gross bit rate

$$R_{\mathrm{b}} \leq \eta B_{\mathrm{u}} \ell. \tag{5.19}$$

Obviously, the number of carriers does not influence the gross bit rate.

In table 5.2, the main communication parameters discussed so far are calculated for different channel conditions, using the results from the radio channel measurement campaign. Overall, the

Table 5.2. OFDM parameters for line-of-sight (LOS) and obstructed (OBS) indoor and outdoor environments (channel model constant $\gamma = 5$, symbol efficiency $\eta = 0.8$, available bandwidth $B_u = 10$ MHz, maximum object velocity $v_o = 100$ km/h, carrier frequency $f_c = 17$ GHz, τ_{RMS} values taken from [11])

		Indoor (LOS)			Indoor (OBS)			Outdoor (LOS)		
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
τ_{RMS}	[ns]	2.54	11.32	32.69	3.82	7.55	11.75	3.71	25.28	52.85
B_c	[MHz]	78.74	17.66	6.12	52.36	26.49	17.02	53.91	7.91	3.78
T_s	[ns]	50.80	226.48	653.80	76.40	150.99	235.00	74.20	505.51	1057.00
f_Δ	[MHz]	19.69	4.42	1.53	13.09	6.62	4.26	13.48	1.98	0.95
N		1	2	7	1	2	2	1	5	11
τ_c		9917	2224	771	6594	3337	2144	6790	997	477

Table 5.3. Key parameters of the UbiCom OFDM transceiver

Number of sub-carriers	N	16
Frequency separation	f_Δ	0.50 MHz
User bandwidth	B_u	8 MHz
Symbol efficiency	η	0.8
Bits per carrier	ℓ	2
Gross bit rate	R_b	12.8 Mbit/s

maximum frequency separation is approximately 1 MHz and the minimum number of sub-carriers is 11. In all cases, the relative coherence time τ_c is larger than 400, indicating that the channel can be assumed quasi-static for at least 400 symbols.

At this point, we are able to specify an OFDM transceiver for the UbiCom project. This transceiver serves as a test case for the design trajectory, laid out in section 1.1.3. The key parameters of the transceiver setup are listed in table 5.3. As argued, the setup applies FDMA in combination with TDD to simultaneously accommodate a maximum of ten users. Its frequency spectrum is shown in figure 5.6. The transceiver applies QPSK, possibly differentially, as a sub-symbol modulation scheme.

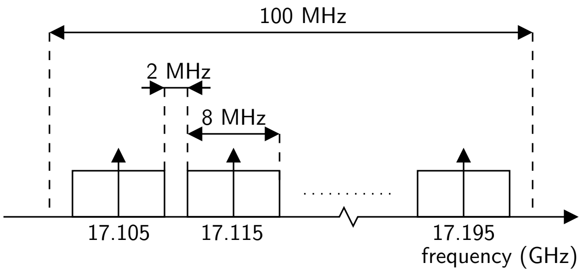
Given these parameters, the relative coherence time τ_c is approximately 250. Therefore, to estimate the channel and to synchronize the demodulator in time and frequency, we use a training sequence. The training sequence is transmitted before the start of a series of symbols. The total length of the series must be smaller than the relative coherence time. For instance, if we set the series length to 32 symbols, each series contains 1 kbit of information. It should be noted that the training overhead reduces the efficiency of the transmission scheme.

5.3 Implementation alternatives

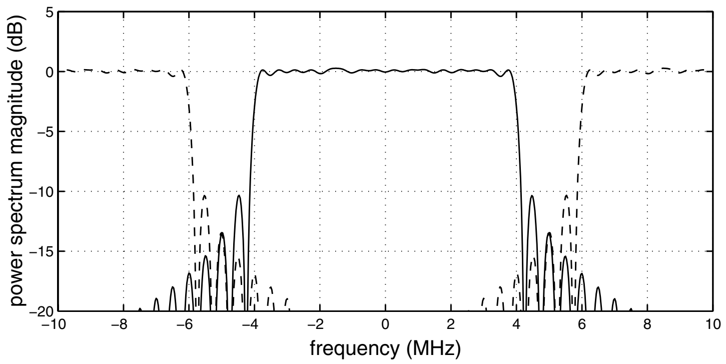
To implement the mobile transceiver as described in section 5.2.7, we have to decide how to partition the transceiver. Therefore, first an executable specification of the transceiver is written in C with several different implementations of the functions. Then, cost estimates are made for implementing these functions in hardware or in software. The costs for the hardware blocks are estimated, using synthesizable descriptions in SystemC generated from the original C code.

The goal of these steps is to come up with design alternatives and their associated costs. This information can be used to answer questions such as

- "How much power does it take to increase the data rate of the transceiver?" The data rate can be raised for example by changing to a higher-order sub-symbol modulation scheme. This probably also increments the costs, but it is unclear to what extent.
- "What is the cost difference between coherent and non-coherent (differential) sub-symbol modulation schemes?" Differential modulation increases the probability of bit errors, but it is unknown what the gains are in terms of energy, speed and area.



(a) Spectrum assignment



(b) Detailed spectrum for a single user

Figure 5.6. Spectrum assignment for the UbiCom OFDM transceiver.

- "What does pre-distortion cost?" Is it worthwhile to implement a pre-distortion scheme, as it enables the reduction of the transmission power while maintaining the same performance through a more effective use of the channel, as described in section 4.4?

The tools do not answer these questions directly as they cannot compare the performance and costs qualitatively. However, they provide quickly the necessary information about the costs of certain design choices.

5.3.1 Building blocks

A classic OFDM transceiver consists of a few basic building blocks. Each of these blocks can be implemented in numerous ways, each resulting in different costs (area, delay, power). Furthermore, the functionality of the block can be changed to arrive at a different transceiver setup.

5.3.1.1 Sub-symbol encoding/decoding

The sub-symbol encodings considered are all variants of QASK schemes. In particular, three schemes are investigated: 4-ary QASK, alternatively known as QPSK (quadrature phase shift keying), differential QPSK (DQPSK) and 16-ary QASK, alternatively known as QAM-16.

The bit error probability P_{e_i} of QPSK for the ith sub-carrier is given by

$$P_{e_i} = Q\left(\sqrt{2\frac{E_{b_i}}{N_0}}\right). \tag{5.20}$$

Differential encoding has a 3 dB penalty

$$P_{e_i} = 2Q\left(\sqrt{2\frac{E_{b_i}}{N_0}}\right), \tag{5.21}$$

and finally the bit error probability of QAM-16 is

$$P_{e_i} = \frac{3}{4}Q\left(\sqrt{\frac{1}{5}\frac{E_{b_i}}{N_0}}\right) + \frac{1}{4}Q\left(\sqrt{\frac{9}{5}\frac{E_{b_i}}{N_0}}\right). \tag{5.22}$$

Both QPSK and QAM-16 require coherent demodulation and hence some form of channel estimation and equalization. In differential QPSK, the information is encoded in the phase difference between two consecutive sub-symbols. As a consequence, coherent demodulation is not necessary, making channel estimation and equalization obsolete.

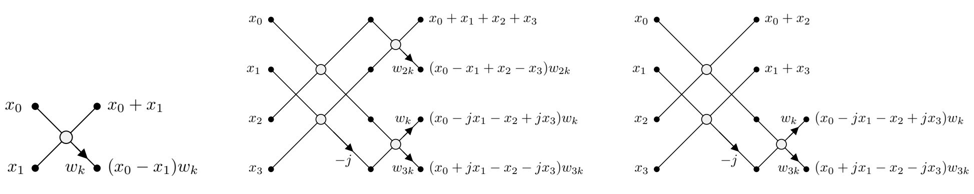
Differential encoding can be applied in frequency and in time. In the first case, symbols are differentially encoded between two adjacent sub-carriers. In the latter, consecutive symbols (in time) on the same sub-carrier are encoded differentially. Differential encoding is only possible if the channel responses of the adjacent or the consecutive sub-carriers is approximately the same.

5.3.1.2 Fast Fourier transform

Numerous efficient implementations of the discrete Fourier transform are known in literature [14, 18, 59, 63], the so-called fast Fourier transforms (FFTs). Three implementations are taken into consideration. The first two are different versions of the Cooley and Tukey radix-N FFT [18], and the third is a combination of them, the so-called split-radix FFT [20].

- *Radix-2 FFT:* The key observation in a radix-2 implementation is that multiplication by $e^{j\pi n}$ for $n \in \mathbb{N}$ is equal to multiplication by ± 1, which are trivial operations. The order of the radix-2 FFT should be a power of 2.
- *Radix-4:* Additionally, in the radix-4 implementation, it is observed that multiplication by $e^{j\frac{\pi}{2}n}$ for $n \in \mathbb{N}$ is equal to multiplication with ± 1 or $\pm j$, again trivial operations. A drawback of the radix-4 FFT is that the order of a radix-4 FFT should be a power of 4 instead of 2. However, the number of non-trivial multiplications is lower than that when using a radix-2 FFT.
- *Split-radix:* The split-radix FFT combines the properties of the radix-2 and radix-4 FFTs. This results in an implementation with an even lower number of non-trivial multiplications [76], while the order should be a power of 2. A drawback is the slightly less regular structure.

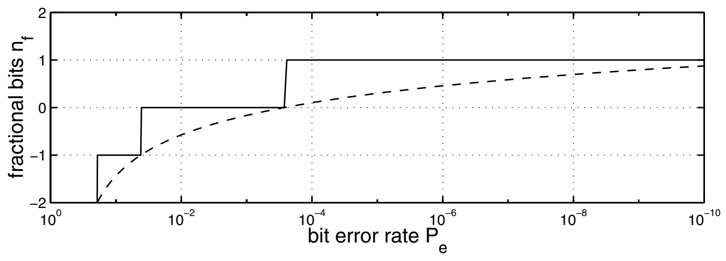
The radix-2, radix-4 and split-radix butterfly structures are depicted in figure 5.7.



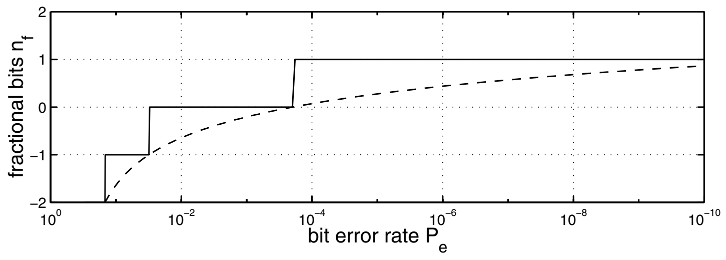
(a) radix-2 (b) radix-4 (c) split-radix

Figure 5.7. The butterfly structures for the radix-2, radix-4 and split-radix Fast Fourier Transforms (FFTs).

All three implementations can be performed in-place, i.e., the intermediate results can be stored in the same place as the original data. Consequently, the memory requirements do not increase while performing an FFT. Other FFTs like the Winograd FFT [85] do not have this property and require more memory space.



(a) QPSK – $E_b = 1$



(b) QAM-16 – $E_b = 2.5$

Figure 5.8. The number of fractional bits n_f necessary to achieve the desired bit error probability P_e for QPSK and QAM-16 sub-symbol modulation.

Bit reversal

A side effect of in-place calculation is that either the inputs or the outputs are bit reversed, i.e., they are not ordered sequentially, and thus they have to be sorted. In case of decimation in time (DIT), the inputs have to be re-ordered and in case of decimation in frequency (DIF), the outputs.

However, if a DIT inverse Fourier transform in the modulator is followed by a DIF forward transform in the demodulator, the re-ordering can be skipped, since the two re-ordering steps effectively cancel each other. It should be noted that this approach leads to a different channel assignment, which can be viewed as a scrambling step. Scrambling can be advantageous, as it spreads localized errors.

Word length

To determine the minimum word length for the FFT, we use the expressions derived in section 4.5, i.e., equations 4.67 and 4.68. Given a desired bit error probability, the number of fractional bits n_f can be calculated for QPSK and QAM-16 sub-symbol modulation schemes, as depicted in figures 5.8(a) and 5.8(b), respectively. Given a desired error probability of at least 10^{-6}, both QPSK and QAM-16 sub-symbol modulation require at least a single fractional bit ($n_\mathrm{f} = 1$).

To determine the number of integer bits necessary, we substitute equations 4.75 and 4.79 into 4.80. Given a desired bit error probability, the number of integer bits n_i required can be calculated for QPSK and QAM-16 sub-symbol modulation schemes, as depicted in figures 5.9(a) and 5.9(b), respectively. Given a desired error probability of at least 10^{-6}, QPSK sub-symbol modulation requires at least 2 integer bits ($n_i = 2$) and QAM-16 modulation at least 5 integer bits ($n_i = 5$).

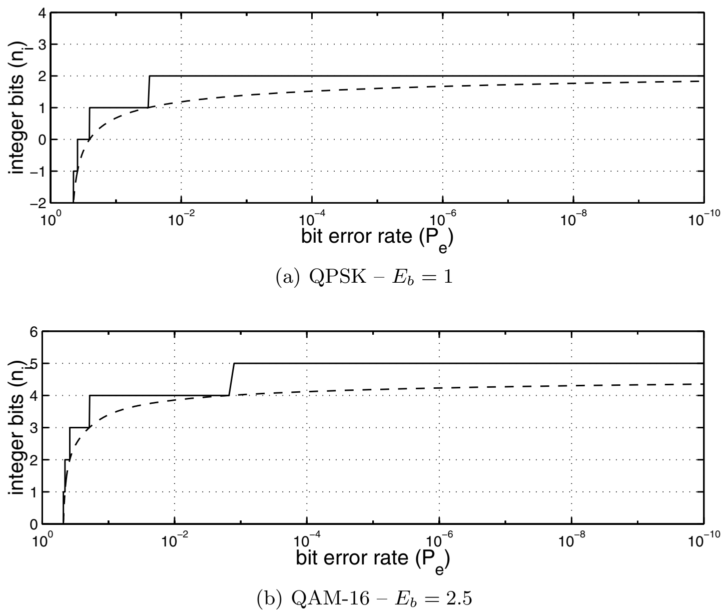


Figure 5.9. The number of integer bits n_i necessary to achieve the desired bit error probability P_e for QPSK and QAM-16 sub-symbol modulation.

5.3.1.3 Cyclic prefix extension / removal

An OFDM symbol can be extended straightforwardly with a cyclic prefix by repeating the last samples of the symbol. The cyclic prefix can be removed by discarding those samples if the correct symbol timing is known from the timing synchronization.

5.3.1.4 Pre-distortion / equalization

As discussed in section 4.3.3, the multi-path channel causes an amplitude attenuation and a phase rotation of each sub-symbol, given by the frequency response of the channel

$$\tilde{z}_{i'}^k = z_i^k H_i. \tag{5.23}$$

This effect can easily be compensated for, with a one-tap equalizer. A one-tap equalizer basically consists of a complex multiplication to multiply the raw sub-symbol with the inverse of the estimate of H_i.

The same approach can be used to pre-distort the sub-symbols in the modulator. A precondition is that an accurate estimate of the channel response is available. This condition can only be met if the channel can be considered quasi-static (which is the case, as argued in section 5.2.7).

5.3.2 Processor alternatives

To accommodate the software tasks, a processor core must be selected. Additionally, the core has to implement the global control for the modules implemented in hardware. A prerequisite for the selection of a suitable processor core is that a compiler, an assembler and a linker should be available in order to generate the object code for the functionality implemented in software (see figure 1.3).

We have implemented two versions of the MIPS processor described in [53], called the µMIPS. The µMIPS has a 32-bit RISC (reduced instruction set computer) architecture. It is selected because of its simplicity, and because it can easily be modified. The data path of the first version is depicted in figure 5.10. It supports all basic integer operations except division. A floating-point unit is not included. The second version of the μMIPS has the same architecture except for the multiplier, which has been omitted.

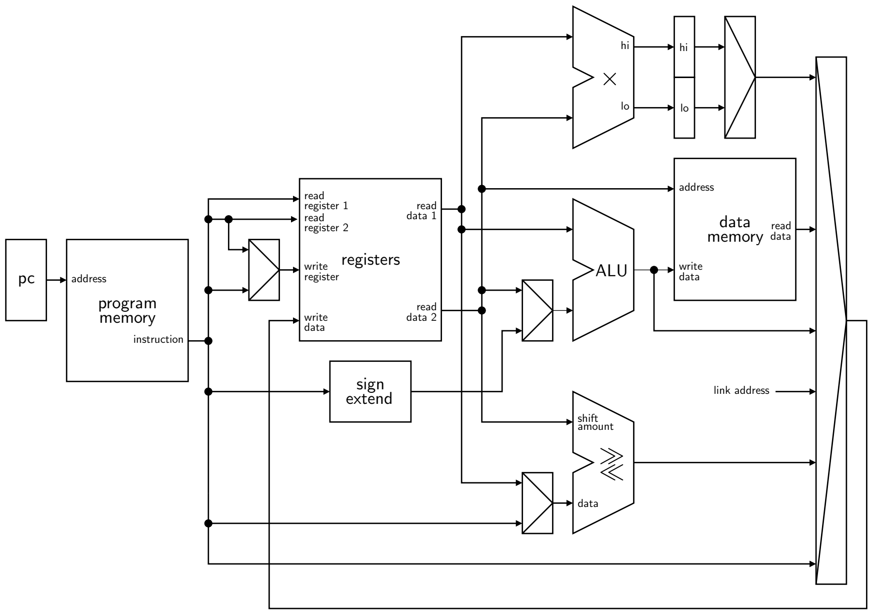


Figure 5.10. Simplified data path of the μMIPS architecture. For clarity, the control path and the program counter data path are excluded.

A compiler is constructed using the retargetable ANSI-C compiler LCC [26] originally developed at Princeton University. For the assembler and linker, the GNU binutils are used. Instructions not supported by the µMIPS architecture (e.g. an integer division) are emulated in software. The compiler can either emulate integer multiplication or use the integer multiplier, if available.

Both versions of the µMIPS are described in SystemC. The code is fully synthesizable. Each version of the µMIPS has been synthesized targeting different clock speeds. In addition, the SystemC code is used to construct a cycle-accurate simulator. The simulator can execute the compiled programs directly. It is used to estimate the cycle counts for the building blocks implemented in software.

5.3.3 Cost estimates

All design alternatives will be constructed using a limited set of building blocks. The cost figures for each block have to be calculated, using one (or more) of the methods presented in chapter 2.

To estimate the execution time and area of a block, a hardware description is generated. Subsequently, to estimate the clock period, the latency and the necessary area, we use a quick scheduling step. The total area estimate is composed of two factors:

- The combinatorial and non-combinatorial area, i.e., the area necessary to accommodate the combinatorial and sequential gates
- The net interconnect area, i.e., the area forecasted to be used by the additional interconnect

These area estimates are used to predict the total capacitive load of the circuit. In combination with an activity estimate obtained through simulation, these estimates are used to calculate an estimate of the energy consumption of the block.

Of course, as absolute figures, the estimates of the energy consumption are not accurate. However, they do give a good estimate of the relative energy costs of the building blocks. Therefore, the results of the partitioning on the energy consumption should not be interpreted in terms of absolute numbers, i.e., quantitatively. Instead, one should see the results qualitatively: one solution is better or worse than another in terms of energy consumption.

On the other hand, the area and delay estimates are much more accurate. We verified this by fully synthesizing a few selected blocks and comparing the results with the estimates. All tests show close agreement between the synthesis results and the estimates obtained after scheduling.

5.3.4 Implementation alternatives

With the building blocks presented in the previous section, we studied several alternatives to implement an OFDM modulator, demodulator or combined transceiver.

5.3.4.1 Stand-alone modulator

Figure 5.11 shows three alternatives to implement a stand-alone OFDM modulator. They apply QPSK, differential QPSK and QAM-16 sub-symbol modulation, respectively. Except for the differential QPSK sub-symbol modulation, pre-distortion of the sub-symbols can optionally be applied in order to improve the performance (expressed as the probability of a bit error) as discussed in section 4.4.

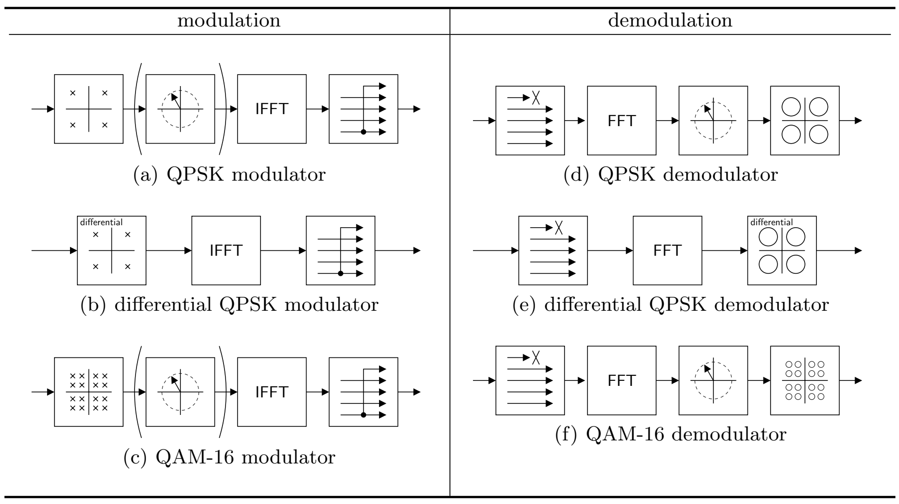


Figure 5.11. Stand-alone OFDM modulation and demodulation implementation alternatives.

5.3.4.2 Stand-alone demodulator

Figure 5.11 also shows three alternatives to implement a stand-alone OFDM demodulator. The first and third coherently decode QPSK-encoded or QAM-16-encoded sub-symbols and thus require an accurate channel estimation and equalization. The second decodes differentially encoded sub-symbols and consequently does not need coherent detection.

Synchronization aspects like coarse-grained and fine-grained frequency synchronization or time synchronization are not taken into account, nor is channel estimation. As the relative coherence time τ_c is fairly large for all environments (see table 5.2), the channel can be assumed quasi-static. Therefore, in the current setup, synchronization and channel estimation will be performed only at the start of a series of symbols. Thus, the impact of the synchronization and channel estimation algorithms will be small with respect to the symbol demodulation, and these are therefore neglected.

5.3.4.3 Transceiver

A transceiver combines an OFDM modulator and a demodulator in one device. Consequently, the device has to implement two different algorithms. Possibly, the device's resources can be shared between the algorithms.

A few straightforward implementations for the transceiver are constructed by combining the modulator and demodulator alternatives presented in the previous sections. Additionally, in figure 5.12, six alternatives are presented to implement an OFDM transceiver. The combined QPSK, DPQSK and QAM-16 transceivers make use of a modified FFT block, which is capable of performing both the forward and the inverse transform. Possibly, this resource sharing reduces the area cost. In contrast, the DIT-DIF alternatives use two separate FFT blocks. However, they do not re-order the data, as discussed in section 5.3.1.2, possibly leading to a faster implementation.

5.3.5 Partitioning results

We used the partitioning tool described in chapter 3 to find the best architecture for the OFDM transceiver. We partitioned the transceiver several times for several optimization goals, under different constraints. The results are used to span the three-dimensional design space (area, power and latency). To aid the designer in identifying the trade-offs, we first partitioned the OFDM modulator and demodulator separately. Subsequently, we applied the same procedure to partition the combined modulator–demodulator. The results are discussed in the next sections. Moreover, trends observed in the partitioning process are described.

5.3.5.1 Stand-alone modulator

The normalized partitioning results for the three implementation alternatives for the stand-alone modulator are plotted in figure 5.13. It is clear that the costs for a QPSK modulator, a differential QPSK modulator or a QAM-16 modulator do not differ significantly.

In figures 5.14 and 5.15, the results are plotted for QPSK and QAM-16 modulation with and without pre-distortion. It is apparent that the addition of a pre-distortion unit increases all costs. However, the possible improvement in performance caused by such a unit is not incorporated in the figure.

5.3.5.2 Stand-alone demodulator

Figure 5.16 shows the normalized partitioning results for the three implementation alternatives for the stand-alone demodulator. In contrast to the stand-alone modulator, the differential QPSK demodulator is obviously less expensive than the coherent QPSK demodulator, because it does not require an equalizer. The difference between QPSK and QAM-16 sub-symbol modulation schemes remains small.

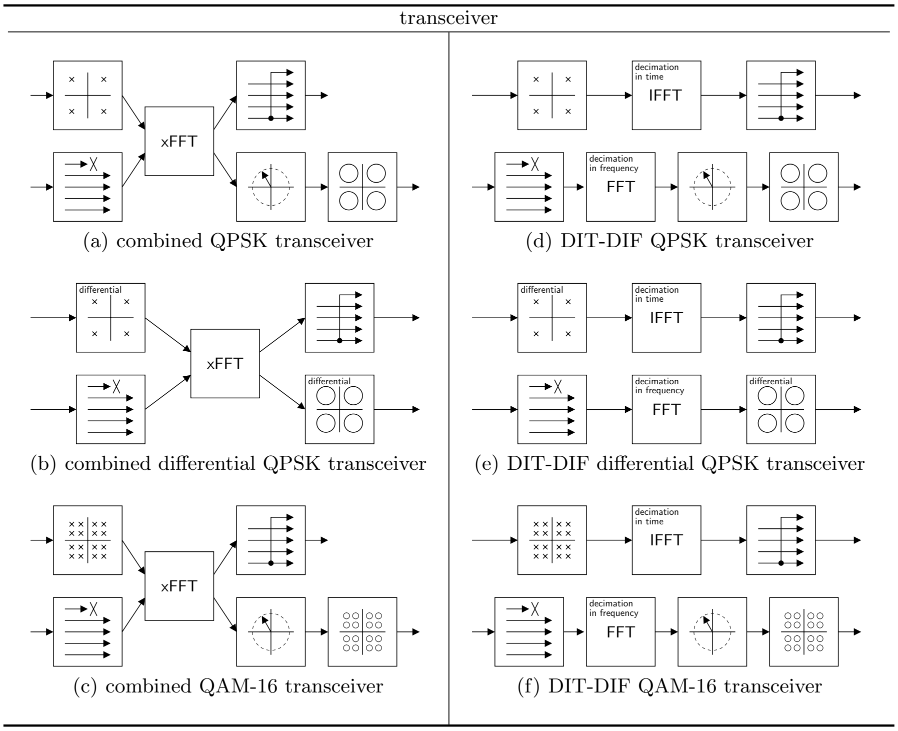


Figure 5.12. Transceiver (combined modulation and demodulation) implementation alternatives.

5.3.5.3 Transceiver

In figure 5.17, the normalized partitioning results for the alternatives for the combined transceiver are plotted. Still the differential QPSK transceiver is the least expensive. In figures 5.18, 5.19 and 5.20, the results of the combined transceivers are compared with their DIT-DIF (scrambled) equivalents for QPSK, differential QPSK and QAM-16 sub-symbol modulation respectively. The DIT-DIF transceivers show both an improvement in execution time and in energy consumption.

5.3.5.4 General observations

Examining the partitioning results in more detail, we can identify a few trends. In general, we have seen that when implementing a certain functionality, realizations with dedicated hardware is faster and consumes less energy than an implementation on a general-purpose processor. It must be noted that the processors used are probably a bit heavyweight for their purpose. Nevertheless, the difference in cost is quite large, and an implementation with a lighter processor will probably be still more expensive in terms of energy and execution time than an implementation with dedicated hardware.

Moreover, we observe the following:

- Area optimization favors software implementation. This observation is quite expected, as a processor is always required in a valid partitioning and as a consequence a trivial solution is to implement all functionality in software using the smallest processor.
- Area optimization favors a single large memory, whereas time and energy optimization favors dedicated distributed memories.

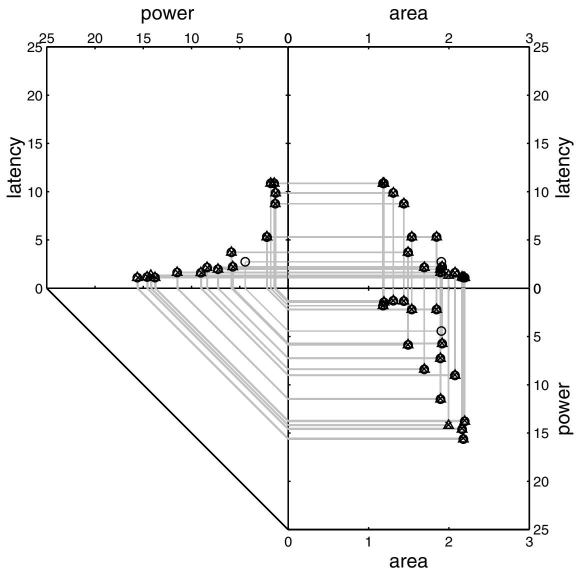


Figure 5.13. Normalized partitioning results for the stand-alone QPSK modulator (×), the stand-alone differential QPSK modulator (○), and the stand-alone QAM-16 modulator (△).

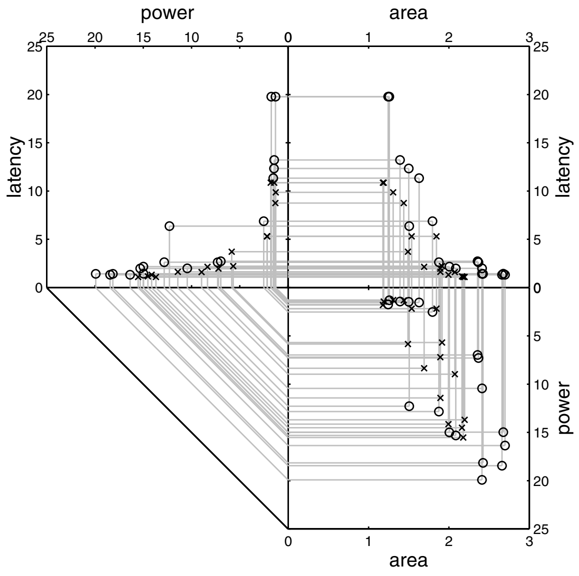


Figure 5.14. Normalized partitioning results for the stand-alone QPSK modulator with (○) and without (×) pre-distortion.

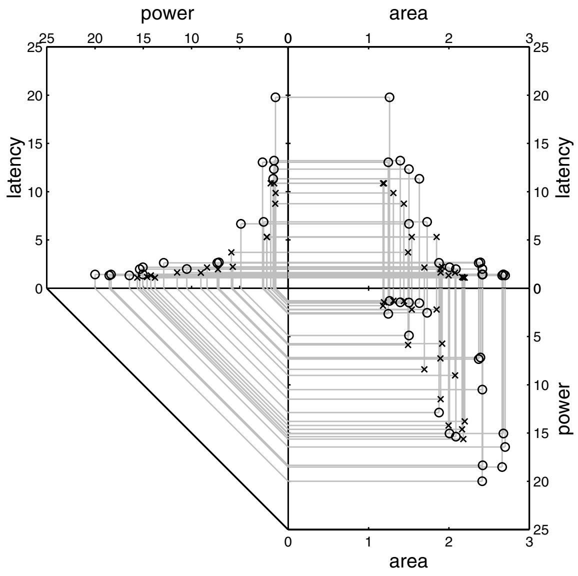


Figure 5.15. Normalized partitioning results for the stand-alone QAM-16 modulator with (○) and without (×) pre-distortion.

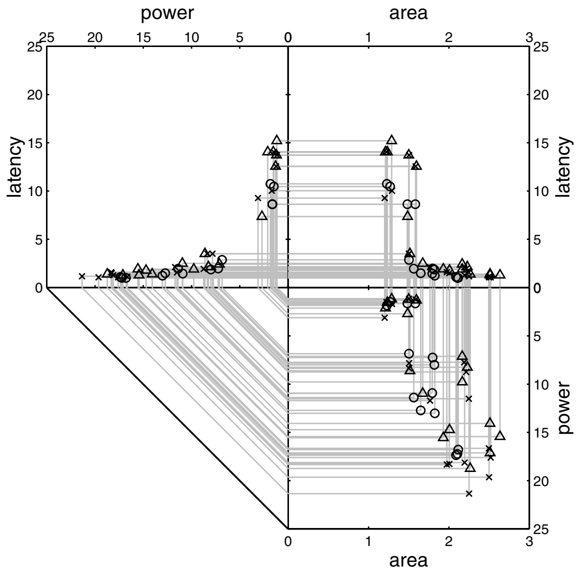


Figure 5.16. Normalized partitioning results for the stand-alone QPSK demodulator (×), the stand-alone differential QPSK demodulator (○), and the stand-alone QAM-16 demodulator (△).

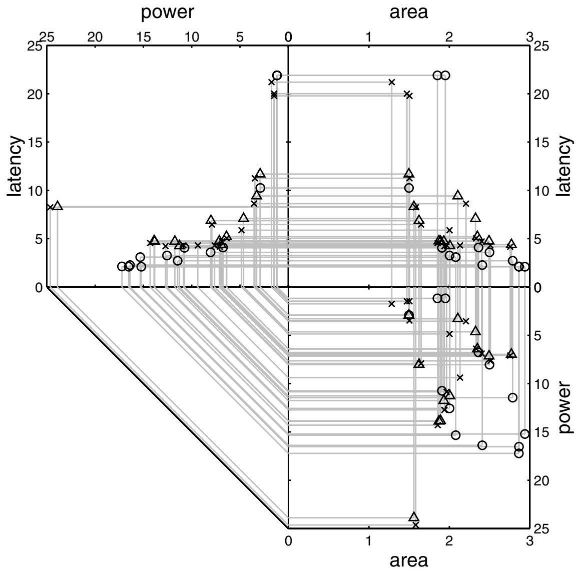


Figure 5.17. Normalized partitioning results for the combined QPSK transceiver (×), the combined differential QPSK transceiver (○), and the combined QAM-16 transceiver (△).

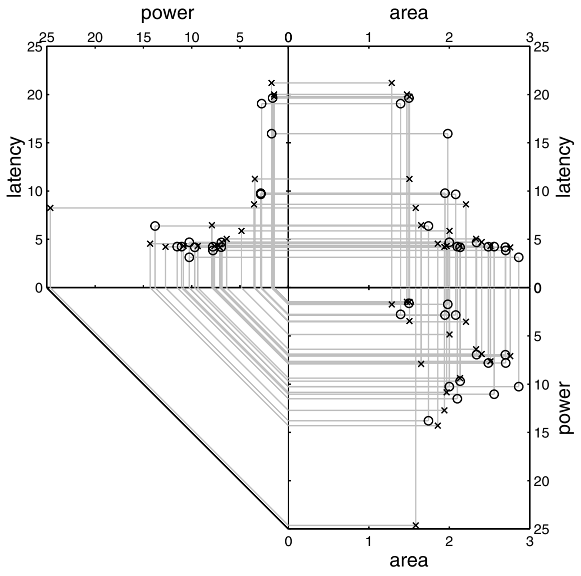


Figure 5.18. Normalized partitioning results for the combined QPSK transceiver, both the normal (×) and the scrambled (○) version.

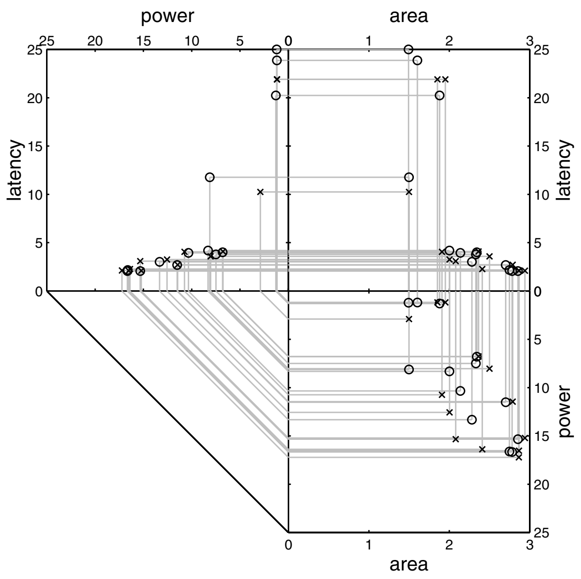


Figure 5.19. Normalized partitioning results for the combined differential QPSK transceiver, both the normal (×) and the scrambled (∘) version.

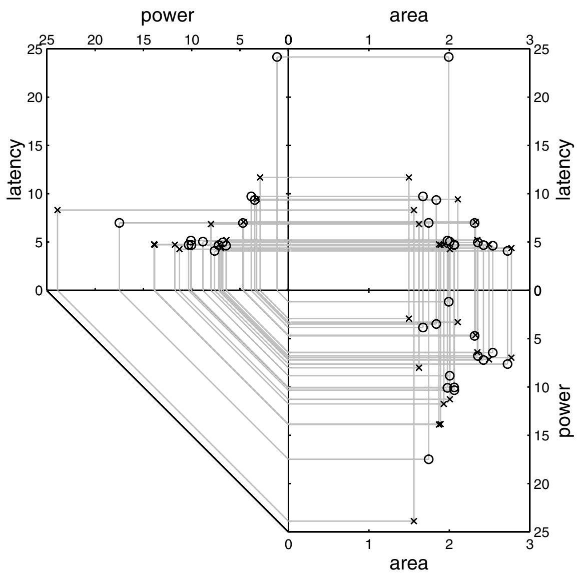


Figure 5.20. Normalized partitioning results for the combined QAM-16 transceiver, both the normal (×) and the scrambled (∘) version.

- A decimation in time radix-4 FFT in hardware gives the best results with respect to execution time and energy consumption.
- A decimation in time split-radix FFT is always selected in software, of course except when the decimation in frequency is explicitly required, but then still the split-radix is used.
- A combined FFT engine gives little overhead in terms of execution time and energy consumption. It is practically always selected, except when maximum speed or minimum energy is desired.
- To implement a combined modulator and demodulator, the FFT blocks without bit reordering give the best results in terms of execution time and and energy consumption.

It must be noted that these are trends observed for specific implementations. Other implementations and different optimization settings may yield other results.

5.4 Summary

To test and verify whether the design flow introduced in chapter 3 is complete, and whether the implemented tools work correctly, we implemented an OFDM transceiver. First, the interdependencies between the key communication parameters were described. Subsequently, these parameters could be determined quantitatively for the UbiCom scenario, given the properties of the multi-path channel in the 17 GHz band and the graphic quality required for augmented reality.

Using the newly found specification of the OFDM transceiver, we identified various alternative implementations. These alternatives were used to specify various algorithms, each implementing an OFDM modulator or demodulator in a different manner. Finally, the consequences for the area, execution time and energy consumption were evaluated using the design tools. In this process, we explored the design spaces of a single modulator, a single demodulator, and a combined transceiver.

In the UbiCom program, we can directly apply the results of the transceiver specification and the subsequent design space explorations. We can use their outcome to determine the best trade-off between cost and performance for the various implementations.

6 Conclusions

Today's designs are characterized by their ever increasing size and complexity. A key problem is that initially a designer has only insufficient or at least inaccurate information available to make proper decisions. Moreover, the overwhelming number of options and the pace with which they change, render it impossible for a human to make a sound decision about the complex designs. He simply cannot predict the consequences of a particular choice accurately. Therefore, tools to quickly identify the trade-offs are indispensable in making well-founded design decisions.

Another problem that a designer faces is that traditional hardware description languages no longer suffice to express the increasing complexity. Therefore, designers turn to general programming languages like C or C^{++} for the initial high-level specification of the algorithms to be implemented. This causes inconsistencies in the design flow (see figure 1.3), as these languages are not easily translated to hardware description languages like VHDL or Verilog.

This book describes several methods and tools to come to a consistent design flow, starting with a high-level specification in C. The resulting design flow is depicted in figure 6.1. An essential part in the flow is a new tool to solve the hardware–software partitioning problem. The tool finds an optimal architecture to implement one or more algorithms. The architecture is optimal with respect to one of the design constraints, which include the maximum allowed chip area, the maximum execution time and the maximum energy consumption. The partitioning problem is solved using mathematical programming, more precisely, using a mixed integer linear program. To deal with the uncertainties inherent to high-level design, the linear program uses fuzzy instead of crisp numbers to represent the coefficients. The linear program is constructed in such a way that it optimizes the most probable outcome, while minimizing the chances of finding a worse solution and maximizing the chances of finding a better one.

Apart from the optimal architecture, the partitioning tool gives a prediction of the total chip area, execution time and energy consumption. The quick operation allows us to spend more time on the exploration of the algorithms, and ascertains that crucial decisions are based on facts and not on mere assumptions or guesses. Ultimately, this prevents that we end up with a product that does not meet its specifications.

As the successful operation of the partitioning tool depends on the quick availability of accurate estimates for area, latency and energy consumption, we implemented an additional tool to convert the original C code into, preferably synthesizable, SystemC. This novel approach was taken because existing solutions that translate C code into an HDL were inadequate, at least for our purposes. The automated conversion from C to SystemC enables a closed design flow where tedious manual rewrites are no longer necessary. Furthermore, the automated conversion eliminates the chances of coding errors. After the generation of synthesizable SystemC code, a partial synthesis step quickly delivers reliable cost estimates. Partial synthesis was chosen to prevent the otherwise necessary modeling of the synthesis tools.

Of course, the proof of the pudding is in the eating, and therefore, the newly crafted tools were put to the test in the design of an OFDM transceiver as specified in the Ubiquitous Communications

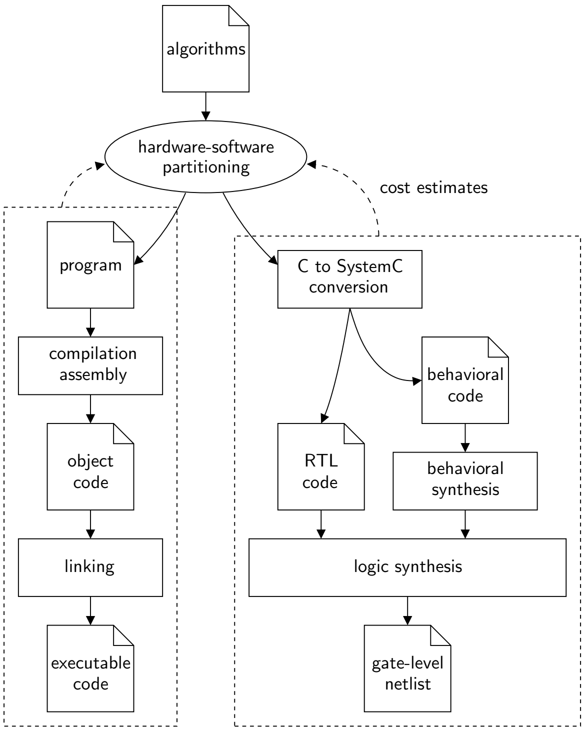


Figure 6.1. Final design flow, using the tools developed in chapter 3.

program. Using the new methods and tools, we were able to quickly design a power-efficient OFDM transceiver. However, before optimizing the implementation of the transceiver with respect to area, latency and energy costs, we investigated several techniques to reduce the energy consumption of the OFDM transceiver beforehand.

Multi-carrier systems such as OFDM allow the assignment of different energy levels to each sub-carrier. This enables pre-distortion of the transmitted sub-symbols in order to improve the performance. We developed an optimization scheme that either minimizes the probability of a bit error for a given energy budget, or that minimizes the total energy necessary to attain a desired bit error rate.

Furthermore, reckoning with a fixed-point implementation of the OFDM transceiver, we investigated the effects of such an implementation on the performance of the transceiver. The results are simple analytical expressions for the performance of uniformly quantized and soft-limited OFDM systems, so that the design space can be quickly explored, and the minimum number of integer and fractional bits needed to achieve the desired performance can be determined.

Looking forward, we foresee a further increase in the (ab)use of general programming languages as system specification languages. The ever growing complexity of the devices envisioned simply cannot be expressed with the current languages, as they lack the level of abstraction necessary to cope with the complexity. Of course, languages like C, C^{++} or even Matlab were never intended for this purpose, but just because of their popularity and widespread use, they set the standard.

Furthermore, we expect energy consumption to become an ever more important constraint, if not the most important constraint, for future designs. Design automation tools must take energy consumption into account. This will require that knowledge of the data to be processed is made available to the synthesis tools in some form or another.

Looking beyond single-embedded systems, a similar design methodology can be used to implement networks of embedded systems, such as wireless sensor networks [2]. In such networks, the individual nodes collect, process and exchange information, and a balance must be found between data processing and communication to increase the lifetime of the network as a whole. The data flow-oriented approach introduced in chapter 3 nicely fits this emerging field, and as a consequence similar optimization strategies can be used.

A
Ubiquitous Communications

The UbiCom program is a multidisciplinary research program at Delft University of Technology. It aims to develop wearable systems for mobile multimedia communications, with a focus on (i) real-time communication and processing of visual information for context-aware applications such as augmented reality, (ii) high bit rate communication at 17 GHz and (iii) architectural issues and performance optimization of heterogeneous communication and computation systems.

A.1 Applications

In a scenario such as that depicted in UbiCom, all kinds of applications become feasible. Apart from applications already envisioned for GPRS (general packet radio services) devices, also applications specifically exploiting the augmented-reality aspects become viable.

The possible applications can be divided into several categories:

- *Geographical information services:* A geographical information system (GIS) can provide the user with information related to his current location. This information can be represented graphically or textually to the user in the form of arrows, highlighted objects, text balloons and in many other ways. Exemplary applications include
  – route planners, and
  – guided tours (in cities, museums, etc.).
- *Remote information services:* All kinds of applications requiring remote expertise or information become possible. To name a few
  – up-to-date traffic information,
  – online interactive manuals, and
  – maintenance supervised by remote experts.
- *Entertainment:* Of course, games are challenging applications, as they usually demand a high level of responsiveness and accuracy. More than one user may play, in games like:
  – *3D quake*: play hide and seek and shoot in the “real” world, and
  – virtual mazes.
- *Military:* In military applications, soldiers at a battlefield are supplied with up to date maps, strategic overviews and targeting information.

A.2 Necessities and consequences

To enable the applications mentioned above, a number of prerequisites have to be fulfilled. The most important are

- *Wearable:* The terminal and display should be easy to wear, and ideally the user should not be hindered or restricted in any way by carrying and operating them. Consequently, these demands impose two important restrictions on the devices:
  - *Lightweight:* Of course the devices used should be lightweight, i.e., the total weight should be less than half a kilogram, but preferably the device should weigh the same as a cellular phone.[1]
  - *Wireless:* As cables and wires severely limit the freedom of movement of a user, they cannot be applied.

  As a consequence, power cables and large and heavy battery packs are not suitable. Therefore, the devices should not consume more than the limited amount of battery power available. Other alternative sources of energy, such as solar panels or fuel cells, are not feasible either because of their limited capacity or because of their weight.
- *Remote information retrieval:* In almost all applications envisioned, the user needs to access data that is not available beforehand, because the user needs up to date data, specific to his current position and current activity. As a consequence, the data required cannot be carried along on some storage medium, but it has to be remotely requested and fetched from a distant information source. This implies the use of some form of wireless communication.
- *High-quality graphics:* As the user's view of the real world is augmented with additional visual information, the graphics and images displayed have to be precisely aligned with the real world. Moreover, they have to be updated frequently to adjust for minute changes in the user's view, caused by movement of the user himself, his head, or even his eyes. Otherwise the user might become disoriented and dizzy, as the added visual information does not match the real world anymore.
- *Localization and determination of direction of view:* Especially for GIS and military applications, the user's position and direction of view should be known, to enhance his view with relevant information. Therefore, positional information about the user has to be communicated when information is requested.

A.3 Preliminary choices

In the UbiCom research program, a number of choices were already made at the start of the project. These include: the use of a high-speed data link at a carrier frequency of 17 GHz, the use of OFDM as a modulation scheme, and the type of infrastructure used. These choices are discussed in more detail in the next sections.

A.3.1 Carrier frequency and OFDM modulation

The choice for the 17 GHz band is primarily based on criteria for the analog front-end design. A carrier frequency was targeted that would allow for integrated front-ends. Integration at higher frequencies was not considered feasible within the life span of the project.

The choice for 17 GHz as carrier frequency also influences the choice for OFDM as modulation scheme. At this frequency, the wavelength of the radio wave is approximately 1.7 cm. Therefore, the channel is expected to exhibit severe frequency-selective fading at the data rates envisioned. To combat the resulting inter-symbol interference, OFDM was selected.

A.3.2 Infrastructure

The device carried around by the user is part of a larger infrastructure. A cellular network is envisioned like that depicted in figure A.1. It consists of four parts:

[1] Other issues like form factor and esthetics are equally important, but are neglected at this point, as they primarily impose limitations on the final construction of the device and not on the electronics within.

- *Mobile terminal:* The device carried by the user capable of both augmenting the user's view and communicating with the base stations to send positional information and to retrieve requested information.
- *Base stations:* Each base station serves a single cell. The wearable devices communicate with one base station at a time.
- *Data servers:* The data servers process the user's requests and subsequently retrieve the information required. For instance, the GIS databases required in some applications are implemented in the data servers.
- *Backbone:* The backbone interconnects the base stations and the data servers. It also connects to the internet if required.

A base station should be able to serve and accommodate at least ten users.

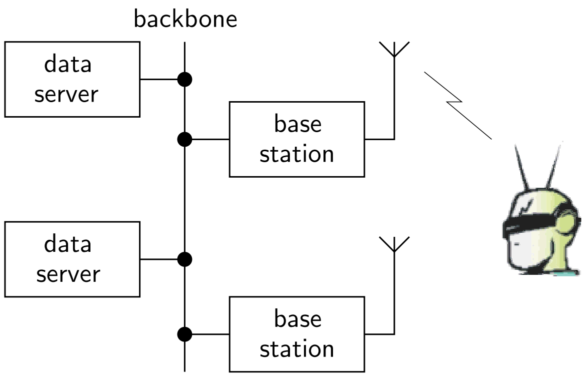


Figure A.1. Infrastructure in the UbiCom scenario.

B

Mixed integer programming

B.1 Linear programming

In general, a linear programming problem has the form

$$
\begin{array}{ll}
\text{maximize} & \\
& c_1x_1 + c_2x_2 + \ldots + c_nx_n, \\
\text{subject to} & \\
& a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \sim b_1, \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \sim b_2, \\
& \vdots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \sim b_m, \\
\text{bounded to} & \\
& l_1 \leq x_1 \leq u_1, \\
& l_2 \leq x_2 \leq u_2, \\
& \vdots \\
& l_n \leq x_n \leq u_n,
\end{array}
\tag{B.1}
$$

where $\sim$ can be any of $\leq$, $\geq$ or $=$. The upper and lower bounds u_i and l_i may be positive or negative infinity, or any real number. The known constants and unknown variables of this program are

objective function coefficients	$c_1, \ldots, c_n,$
constraint coefficients	$a_{11}, \ldots, a_{mn},$
constraint limits	$b_1, \ldots, b_m,$
lower bounds	$l_1, \ldots, l_n,$
upper bound	$u_1, \ldots, u_n,$
variables	$x_1, \ldots, x_n.$

Equation B.1 can be expressed in matrix form as follows:

$$\begin{aligned}
&\text{maximize} \\
&\qquad \begin{bmatrix} c_1 \; c_2 \ldots c_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \\
&\text{subject to} \\
&\qquad \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \sim \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \\
&\text{bounded to} \\
&\qquad \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{bmatrix} \le \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \le \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix},
\end{aligned} \tag{B.2}$$

which leads to

$$\begin{aligned}
&\text{maximize} && c^T x, \\
&\text{subject to} && Ax \sim b, \\
&\text{bounded to} && l \le x \le u.
\end{aligned} \tag{B.3}$$

In practice, general linear programming problems are solved very quickly by the Simplex algorithm. The algorithm visits the corner points of the search space (the region that satisfies $Ax \sim b$), where the optimal solution is known to exist. However, the Simplex algorithm is not guaranteed to reach a solution in polynomial time. The ellipsoid algorithm and Karmarkar's algorithm do reach a solution in polynomial time, however.

B.2 Mixed integer programming

In mixed integer programming, any of the variables $x_1, \ldots, x_n$ of a linear program can be restricted to integer or binary values. In general, integer programming is $\mathcal{NP}$-hard.

B.3 Boolean algebra

In chapter 3, Boolean algebra is used to set up the partitioning MILP. Therefore, the Boolean equations have to be substituted by one or more arithmetical (in)equality constraints.

B.3.1 Boolean inversion

A Boolean inverse

$$y = \overline{x} \tag{B.4}$$

is replaced by a single equality

$$y = 1 - x, \tag{B.5}$$

provided that x and y are binary-valued variables.

B.3.2 Boolean and

A Boolean product

$$y = \bigwedge_{i=1}^{n} x_i = x_1 \cdot x_2 \cdot \ldots \cdot x_n \tag{B.6}$$

is replaced by two inequalities

$$\begin{cases} \sum_{i=1}^{n} x_i \geq n \cdot y, \\ \sum_{i=1}^{n} x_i \leq y + (n-1), \end{cases} \tag{B.7}$$

provided that x_i and y are binary-valued variables.

B.3.3 Boolean or

A Boolean sum

$$y = \bigvee_{i=1}^{n} x_i = x_1 + x_2 + \ldots + x_n \tag{B.8}$$

is replaced by two inequalities

$$\begin{cases} \sum_{i=1}^{n} x_i \geq y, \\ \sum_{i=1}^{n} x_i \leq n \cdot y, \end{cases} \tag{B.9}$$

again provided that x_i and y are binary-valued variables.

C

Possibilistic linear programming

C.1 Introduction

In this appendix, we try to solve a possibilistic linear program (LP) given by

$$\begin{array}{ll}
\text{minimize} & \\
& \tilde{c}_1 x_1 + \tilde{c}_2 x_2 + \ldots + \tilde{c}_n x_n, \\
\text{subject to} & \\
& \tilde{a}_{11} x_1 + \tilde{a}_{12} x_2 + \ldots + \tilde{a}_{1n} x_n \sim \tilde{b}_1, \\
& \tilde{a}_{21} x_1 + \tilde{a}_{22} x_2 + \ldots + \tilde{a}_{2n} x_n \sim \tilde{b}_2, \\
& \vdots \\
& \tilde{a}_{m1} x_1 + \tilde{a}_{m2} x_2 + \ldots + \tilde{a}_{mn} x_n \sim \tilde{b}_m, \\
\text{bounded to} & \\
& l_1 \leq x_1 \leq u_1, \\
& l_2 \leq x_2 \leq u_2, \\
& \vdots \\
& l_n \leq x_n \leq u_n,
\end{array} \tag{C.1}$$

or in short

$$\begin{array}{ll}
\text{minimize} & \tilde{c}^T x, \\
\text{subject to} & \tilde{A} x \leq \tilde{b}, \\
\text{bounded to} & l \leq x \leq u,
\end{array} \tag{C.2}$$

where $\tilde{c}$, $\tilde{A}$ and $\tilde{b}$ are the imprecise coefficients for the objective, constraints and limits, respectively.

In case the imprecise coefficients are represented by triangular fuzzy numbers (see section 3.6), we can solve equation C.2 using Zimmermann's fuzzy programming method [87] with the normalization process described in [38].

C.2 Fuzzy objective coefficients

First, consider the case of imprecise objective coefficients $\tilde{c}$ and precise (or crisp) constraint and limit coefficients. As x is not fuzzy but crisp, minimizing $\tilde{c}x$ is equivalent to minimizing $\{c_l x, c_m x, c_u x\}$. Instead of simultaneously minimizing these three objectives, we will minimize $c_m x$, maximize $(c_m - c_l)x$, and minimize $(c_u - c_m)x$, i.e.,

$$\text{minimize } \tilde{c}x \Rightarrow \begin{cases} \text{minimize } z_1 = c_m x, \\ \text{maximize } z_2 = (c_m - c_l)x, \\ \text{minimize } z_3 = (c_u - c_m)x. \end{cases} \tag{C.3}$$

This means minimizing the most possible value of the imprecise objective, while maximizing the possibility of a lower cost and minimizing the risk of a higher cost, respectively.

To solve equation C.3, we use Zimmermann's fuzzy linear programming approach using the min operator, in combination with the linear membership functions of the three objective functions

$$\mu_{z_1} = \begin{cases} 1 & \text{if } z_1 < z_1^l, \\ \dfrac{z_1 - z_1^u}{z_1^l - z_1^u} & \text{if } z_1^l \le z_1 \le z_1^u, \\ 0 & \text{if } z_1 > z_1^u, \end{cases} \tag{C.4}$$

$$\mu_{z_2} = \begin{cases} 0 & \text{if } z_2 < z_2^l, \\ \dfrac{z_2 - z_2^l}{z_2^u - z_2^l} & \text{if } z_2^l \le z_2 \le z_2^u, \\ 1 & \text{if } z_2 > z_2^u, \end{cases} \tag{C.5}$$

and μ_{z_3} similar to μ_{z_1}. The lower and upper bounds z_i^l and z_i^u of an objective function z_i are given by

$$z_i^l = \min_{x \in X} \; z_i, \tag{C.6}$$

$$z_i^u = \max_{x \in X} \; z_i, \tag{C.7}$$

for all $i \in 1, 2, 3$, and where $X = \{x \mid Ax \le b, \; l \le x \le u\}$ is the solution space.

With Zimmermann's method, the membership functions z_i are maximized simultaneously. Thus we arrive at the single objective crisp LP given by

$$\begin{array}{ll} \text{maximize} & \\ & \lambda, \\ \text{subject to} & \\ & \mu_{z_i} \ge \lambda, \; \forall i \in (1, 2, 3), \\ & Ax \le b, \\ \text{bounded to} & \\ & l \le x \le u, \end{array} \tag{C.8}$$

which can be solved using a standard LP solver.

C.3 Fuzzy objective, constraint and limit coefficients

If, not only the fuzzy objective function $\tilde{c}$ is imprecise but also the constraint $\tilde{A}$ and limit $\tilde{b}$ coefficients, the same approach as that presented in the previous section can be used. First however, we need to convert the imprecise constraints into crisp ones using the fuzzy ranking concept. All constraints with imprecise coefficients can be replaced by three auxiliary constraints [65]

$$A_l^\alpha x \le b_l^\alpha, \tag{C.9}$$

$$A_m^\alpha x \le b_m^\alpha, \tag{C.10}$$

$$A_u^\alpha x \le b_u^\alpha, \tag{C.11}$$

where α is the minimal acceptable possibility, specified by the designer. From figure C.1, it can be seen that

$$c_m^\alpha = c_m, \tag{C.12}$$

$$c_l^\alpha = \alpha c_m + (1 - \alpha) c_l, \tag{C.13}$$

$$c_u^\alpha = \alpha c_m + (1 - \alpha) c_u. \tag{C.14}$$

Once α is known, we again have crisp constraints with a fuzzy objective.

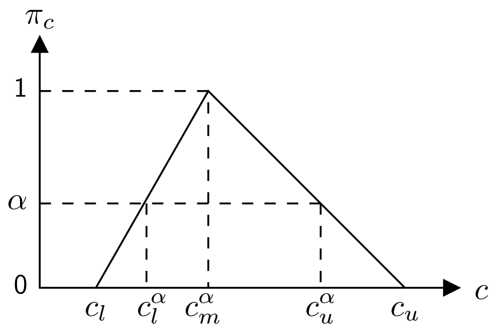


Figure C.1. Minimal acceptable possibility α of a triangular fuzzy number $\tilde{c} = \{c_l, c_m, c_u\}$.

References

1. G. Aigner, A. Diwan, D.L. Heine, M.S. Lam, D.L. Moore, B.R. Murphy, and C. Sapuntzakis. An overview of the SUIF2 compiler infrastructure. `http://suif.stanford.edu`.
2. Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.
3. A. Allan, D. Edenfeld, W.H. Joyner, Jr., A.B. Kahng, M. Rodgers, and Y. Zorian. 2001 technology roadmap for semiconductors. *Computer*, 35(1):42–53, January 2002.
4. Charles J. Alpert, Anirudh Devgan, and Chandramouli V. Kashyap. RC delay metrics for performance optimization. *IEEE Transactions on Computer-Aided Design*, 20(5):571–582, May 2001.
5. H.B. Bakoglu. *Circuits, interconnections, and packaging for VLSI.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1990.
6. Paolo Banelli. Theoretical analysis and performance of OFDM signals in nonlinear fading channels. *IEEE Transactions on Wireless Communications*, 2(2):284–293, March 2003.
7. Paolo Banelli and Saverio Cacopardi. Theoretical analysis and performance of OFDM signals in nonlinear AWGN channels. *IEEE Transactions on Communications*, 48(3):430–441, March 2000.
8. Jan-Jaap van de Beek, Magnus Sandell, Mikael Isaksson, and Per Ola Börjesson. Low-complex frame synchronization in OFDM systems. In *Proceedings of the 4^{th} IEEE International Conference on Universal Personal Communications*, pages 982–986, November 1995.
9. J.A.C. Bingham. Multicarrier modulation for data transmission: An idea whose time has come. *IEEE Communications Magazine*, 9:5–14, May 1990.
10. J.A.C. Bingham. *ADSL, VDSL and multicarrier modulation.* Wiley, New York, 2000.
11. A. Bohdanowicz. Wideband indoor and outdoor radio channel measurements at 17 GHz. Technical report, Delft University of Technology, Delft, The Netherlands, February 2000.
12. A. Bohdanowicz, G.J.M. Janssen, and S. Pietrzyk. Wideband indoor and outdoor multipath channel measurements at 17 GHz. *IEEE Proceedings of Vehicular Technology Conference*, pp. 1998–2003, Amsterdam, The Netherlands, September 1999.
13. C. Bruma. *Systems-on-a-chip architecting.* PhD thesis, Delft University of Technology, December 1999. ISBN: 90-5326-030-7.
14. C. Sidney Burrus and Peter W. Eschenbauer. An in-place, in-order prime factor FFT algortihm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29(4):806–817, August 1981.
15. C. Chen and M. Ahmadi. Register-transfer-level power estimation based on technology decomposition. *IEE Proceedings on Circuits, Devices and Systems*, 150(5):411–415, October 2003.
16. K.-T. Cheng and V.D. Agrawal. An entropy measure for the complexity of multi-output boolean functions. In *Proceedings of the 27^{th} ACM/IEEE Design Automation Conference*, pp. 302–305, 1990.
17. P. Christie. Rent exponent prediction methods. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(6):679–688, December 2000.
18. James. W. Cooley and John. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.
19. W.E. Donath. Placement and average interconnection lengths of computer logic. *IEEE Transactions on Circuits and Systems*, CAS-26(4):272–277, April 1979.
20. P. Duhamel and H. Hollmann. Split-radix FFT algorithm. *Electronic Letters*, 20(1):14–16, January 1984.

21. W.C. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19:55–63, January 1948.
22. The European table of frequency allocations and utilisations covering the frequency range 9 kHz to 275 GHz. European Radiocommunications Committee (ERC), January 2002.
23. Radio broadcasting systems; digital audio broadcasting (DAB) to mobile, portable and fixed receivers. ETSI, 1995.
24. Broadband radio access networks (BRAN); HIPERLAN type 2; System overview. ETSI, February 2000.
25. UMTS terrestrial radio access (UTRA); concept evaluation. ETSI, December 1997.
26. David R. Hanson and Christopher W. Fraser. *A retargetable C compiler: Design and implementation.* Addison-Wesley Publishing Company, Inc., 1995.
27. Leo Hellerman. A measure of computational work. *IEEE Transactions on Computers*, C-22:439–446, May 1972.
28. IEEE Std 802.11-2007, IEEE standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements. IEEE, 2007.
29. IEEE Std 1666-2005, IEEE standard SystemC language reference manual. IEEE, 2005.
30. Leon W. Couch II. *Digital and analog communication systems.* Macmillan Publishing Company, New York, 1993.
31. Gerard J.M. Janssen. *Robust receiver techniques for interference-limited radio channels.* PhD thesis, Delft University of Technology, 1998.
32. Irving Kalet. The multitone channel. *IEEE Transactions on Communications*, 37(2):119–124, February 1989.
33. I. Karkowski. *Performance driven synthesis of digital systems.* PhD thesis, Delft University of Technology, December 1995. ISBN: 90-5326-022-6.
34. H. Keding, M. Willems, M. Coors, and H. Meyr. Fridge: a fixed-point design and simulation environment. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 429–435, 1998.
35. A.C.J. Kienhuis. *Design space exploration of stream-based dataflow architectures: Methods and tools.* PhD thesis, Delft University of Technology, January 1999. ISBN: 90-5326-029-3.
36. Seehyun Kim, Ki-Il Kum, and Wonyong Sung. Fixed-point optimization utility for c and C^{++} based digital signal processing programs. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(11):1455–1464, November 1998.
37. R.L. Lagendijk. Ubiquitous communications (UbiCom) – updated technical annex. Technical report, Delft University of Technology, Delft, The Netherlands, January 2000.
38. Young-Jou Lai and Ching-Lai Hwang. A new approach to some possibilistic linear programming problems. *Fuzzy Sets and Systems*, 49(2):121–133, 1992.
39. M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno. Cosimulation-based power estimation for system-on-chip design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(3):253–266, June 2002.
40. P. Lall. Tutorial: Temperature as an input to microelectronics-reliability models. *IEEE Transaction on Reliability*, 45(1):3–9, March 1996.
41. Bernard S. Landman and Roy L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, C-20(12):1469–1479, December 1971.
42. S.Y. Liao. Towards a new standard for system-level design. In *Proceedings of the 8th international workshop on hardware/software codesign*, pp. 2–6, 2000.
43. S.M. Loo, B.E. Wells, N. Freije, and J. Kulick. Handel-C for rapid prototyping of VLSI coprocessors for real time systems. In *Proceedings of the 34th Southeastern Symposium on System Theory*, pp. 6–10, 2002.
44. P.R. van der Meer. *Low-power deep sub-micron CMOS logic - sub-threshold current reduction.* PhD thesis, Delft University of Technology, January 2003. ISBN: 90-9016-408-1.
45. Giovanni De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill, Inc., New York, 1994.
46. Gorden E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8):53–59, April 1965.
47. Paul. H. Moose. A technique for orthogonal frequency division multiplexing frequency offset correction. *IEEE Transactions on Communications*, 42(10):2908–2914, October 1994.
48. Nationaal frequentieplan 2002 (NFP). Ministerie van Verkeer en Waterstaat – Telecommunicatie en Post, 2002.

49. M. Nemani and F.N. Najm. High-level area and power estimation for vlsi circuits. *IEEE Transactions on Computer-Aided Design*, 18(6):697–713, June 1999.
50. Hideki Ochiai and Hideki Imai. Performance analysis of deliberately clipped OFDM signals. *IEEE Transactions on Communications*, 50(1):89–101, January 2002.
51. S. Oraintara, Y.J. Chen, and T.Q. Nguyen. Integer fast fourier transform. *IEEE Transactions on Signal Processing*, 50(3):607–618, March 2002.
52. W. Pasman and F.W. Jansen. Distributed low-latency rendering for mobile AR. *IEEE and ACM International Symposium on Augmented Reality*, pages 107–113, October 2001.
53. David A. Patterson and John L. Hennessy. *Computer organization & design: The hardware/software interface.* Morgan Kaufmann Publishers, Inc., second edition, 1997.
54. P.Z. Peebles, Jr. *Probability, random variables, and random signal principles.* McGraw-Hill, Inc., third edition, 1993.
55. Thierry Pollet, Mark van Bladel, and Marc Moeneclaey. BER sensitivity of OFDM systems to carrier frequency offset and wiener phase noise. *IEEE Transactions on Communications*, 43(2/3/4):191–193, February/March/April 1995.
56. Miodrag Potkonjak and Jan Rabaey. Area-time high level synthesis laws: theory and practice. *VLSI Signal Processing Workshop*, pages 53–62, October 1994.
57. J.A. Pouwelse. *Power management for portable devices.* PhD thesis, Delft University of Technology, October 2003. ISBN: 90-6464-993-6.
58. Robert A. Powers. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4):687–693, April 1995.
59. Robert D. Preuss. Very fast computation of the radix-2 discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-30(4):595–607, August 1982.
60. Robert Price. A useful theorem for nonlinear devices having gaussian inputs. *IRE Transactions on Information Theory*, 4(2):69–72, June 1958.
61. John G. Proakis. *Digital communications.* McGraw-Hill, Inc., fourth edition, 2001.
62. Jan M. Rabaey and Massoud Pedram, editors. *Low power design methodologies.* Kluwer Academic Publishers, 1996.
63. Charles M. Rader and N.M. Brenner. A new principle for fast fourier transformation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(3):264–266, June 1976.
64. A. Raghunathan, S. Dey, and N.K. Jha. High-level macro-modeling and estimation techniques for switching activity and power consumption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):538–557, August 2003.
65. Jaroslav Ramík and Josef Římánek. Inequality relation between fuzzy numbers and its use in fuzzy optimization. *Fuzzy Sets and Systems*, 16:123–128, 1985.
66. Ulrich Reimers. DVB-T: the COFDM-based system for terrestrial television. *Electronics & Communication Engineering Journal*, pages 28–32, February 1997.
67. Michael J. Riezenman. The search for better batteries. *IEEE Spectrum*, 32(5):51–56, May 1995.
68. R. Rinker et al. An automated process for compiling dataflow graphs into reconfigurable hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):130–139, February 2001.
69. H.E. Rowe. Memoryless nonlinearities with gaussian inputs: Elementary results. *Bell System Technical Journal*, 61(7):1519–1525, September 1982.
70. Kaushik Roy and Sharat C. Prasad. *Low-power CMOS VLSI circuit design.* Wiley, 2000.
71. Jorge Rubinstein, Paul Penfield, Jr., and Mark A. Horowitz. Signal delay in RC tree networks. *IEEE Transactions on Computer-Aided Design*, CAD-2(3):202–210, July 1983.
72. Takayasu Sakurai and A. Richard Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, 25(2):584–594, April 1990.
73. P. Schaumont, S. Vernalde, L. Rijnders, M. Engels, and I. Bolsens. A programming environment for the design of complex high speed ASICs. In *Proceedings of the 35th Design Automation Conference (DAC)*, pages 315–320, June 1998.
74. Timothy M. Schmidl and Donald C. Cox. Robust frequency and timing synchronization for OFDM. *IEEE Transactions on Communications*, 45(12):1613–1621, December 1997.
75. L. Séméria, K. Sato, and G. De Micheli. Synthesis of hardware models in C with pointers and complex data structures. *IEEE Transactions on VLSI systems*, 8(6):743–756, December 2001.
76. Henrik V. Sorensen, Michael T. Heideman, and C. Sidney Burrus. On computing the split-radix FFT. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-34(1):152–156, February 1986.
77. Michael Speth, Stefan A. Fechtel, Gunnar Fock, and Heinrich Meyr. Optimum receiver design for wireless broad-band systems using OFDM – Part I. *IEEE Transactions on Communications*, 47(11):1668–1677, November 1999.

78. D. Stroobandt, H. van Marck, and J. van Campenhout. An accurate interconnection length estimation for computer logic. In *Proceedings of the 6th Great Lakes Symposium on VLSI*, pp. 50–55, March 1996.
79. Wonyong Sung and Ki-Il Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transactions on Signal Processing*, 43(12):3087–3090, December 1995.
80. Synopsys, Inc. *Converting ANSI-C into fixed-point using CoCentric Fixed-point Designer – methodology backgrounder*, April 2000.
81. Synopsys, Inc. *CoCentric SystemC Compiler – behavioral modeling guide*, August 2001.
82. Jeffrey D. Ullman. *Computational aspects of VLSI.* Computer Science Press, Inc., Rockville, MD, 1984.
83. P.D. Welch. A fixed-point fast fourier transform error analysis. *IEEE Transactions on Audio and Electroacoustics*, AU-17:151–157, June 1969.
84. R.P. Wilson et al. SUIF: An infrastructure for research on parallelizing and optimizing compilers. *ACM SIGPLAN Notices*, 29(12):31–37, 1994.
85. S. Winograd. On computing the discrete fourier transform. *Mathematics of Computation*, 32:175–199, 1978.
86. Klaus Witrisal. *OFDM air-interface design for multimedia communications.* PhD thesis, Delft University of Technology, April 2002. ISBN: 90-76928-03-7.
87. H.J. Zimmermann. Fuzzy programming and linear programming with several objective functions. *Fuzzy Sets and Systems*, 1:45–55, 1978.

Index