Cheng-Chi Wong · Hsie-Chia Chang

Turbo Decoder Architecture for Beyond-4G Applications

Springer

Turbo Decoder Architecture for Beyond-4G Applications

Cheng-Chi Wong • Hsie-Chia Chang

Turbo Decoder Architecture for Beyond-4G Applications

Springer

Cheng-Chi Wong
Department of Electronics Engineering
National Chiao-Tung University
Hsinchu, Taiwan

Hsie-Chia Chang
Department of Electronics Engineering
National Chiao-Tung University
Hsinchu, Taiwan

ISBN 978-1-4614-8309-0 ISBN 978-1-4614-8310-6 (eBook)
DOI 10.1007/978-1-4614-8310-6
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013947477

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Turbo code is one of the error-correcting techniques of the standards for 4G telecommunication system. From current development trend, there is a growing demand for faster data transmission; therefore, the throughput of turbo decoder should be raised to support the next-generation application. The common way to reach this objective is using parallel architecture. However, it causes increasing complexity and decreasing efficiency. These problems will limit the achievable improvement. The aim of this book is to resolve the two design issues, and our main emphasis is on the practical turbo decoders for 3GPP LTE-Advanced and IEEE 802.16m standards. Chapter 1 gives an overview of the code specifications, theoretical principles, and essential algorithms. Then Chap. 2 introduces basic functional units and main processor of a conventional turbo decoder. It further indicates the architecture with superior performance and reasonable overhead. Next, Chap. 3 illustrates the characteristics of advanced parallel architecture and explains the negative effects on complexity and efficiency. There are several implementation results supporting the introductions in Chaps. 2 and 3. All of them are derived with 90 nm technology. Chapter 4 presents how to simplify the parallel turbo decoder, while Chap. 5 shows how to get a better utilization of the component circuits. Moreover, the last two chapters highlight the conditions in which these proposed methods are suitable to use. With the materials of this book, the readers would learn about choosing the decoder architecture that can fulfill their requirements.

We wish to thank United Microelectronics Corporation for their technical support. We also wish to thank National Chiao Tung University for providing a supportive environment. In particular, we thank Professor Chen-Yi Lee for his help. Finally, we would like to express our sincere gratitude to everyone who assisted us in writing, editing, and publishing this book.

Hsinchu, Taiwan — Cheng-Chi Wong
Hsinchu, Taiwan — Hsie-Chia Chang

Contents

Chapter 1
Introduction

A successful transfer of information over a physical channel or a transmission medium involves a series of procedures. The model of data transmission can be depicted as Fig. 1.1. Generally, it consists of a transmitter, a channel, and a receiver. The elements inside the transmitter and receiver are utilized to guarantee reliable and efficient transmission. For less resources usage, the source encoder uses a shorter symbol sequence to replace the source information, while the source decoder performs the data decompression. When data pass through the channel, they will suffer from the channel noise and may become incorrect. To make sure the accurate information can be delivered to the destination, the channel encoder will transform its inputs into a structured sequence where parity check symbols are introduced. With these redundancies, the channel decoder is capable of recovering the messages even though the received data contain errors caused by channel impairments. In addition to the elements for signal processing, the transmitter needs a modulator to translate the data into analog forms which is suitable for transmission; and the receiver uses a demodulator to convert the channel outputs back to quantized symbols. All of the components determine the quality of data transmission. The development of the corresponding techniques will lead to the advancement of communication systems.

In a communication system, the correctness of data transmission is one of the most essential issues. This task of protecting the transmitted information against the channel noise is done by the channel encoder and channel decoder. The study about these subjects is called *forward error correction* or *channel coding*. It originated from the landmark paper by C. E. Shannon in 1948 [1,2]. Shannon's channel coding theorem indicated that arbitrary transmission can be asymptotically error-free by appropriate coding techniques if the code rate is less than the channel capacity. The theoretical limits on performance can be calculated for various signaling schemes, rates, and channels. A lot of coding techniques are developed since then, and there are two main classes of codes: *block codes* and *convolutional codes* [3, 4]. The ultimate objective is approaching the Shannon limit by a practical coding technique, and such investigation has lasted for several decades. In the 1990s, the advent of

C.-C. Wong and H.-C. Chang, *Turbo Decoder Architecture for Beyond-4G Applications*, DOI 10.1007/978-1-4614-8310-6_1,

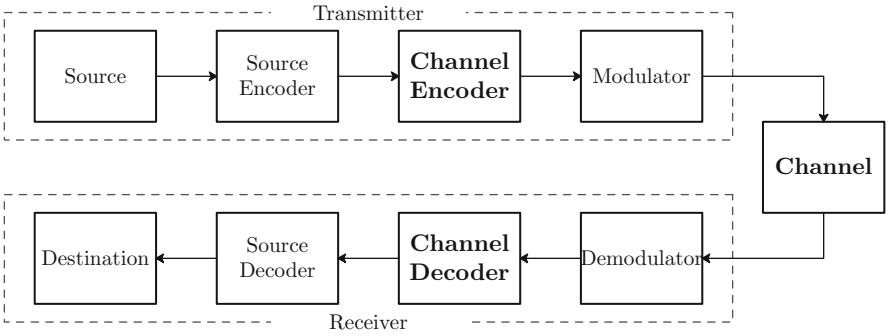

Fig. 1.1 Basic elements of a communication system

turbo codes in [5] and the rediscovery of low-density parity check codes in [6] made major breakthroughs in channel coding. The two techniques both can achieve the performance close to Shannon limit at the expense of reasonable complexity [7–9]. Because of their outstanding features, they have been chosen as the forward error correction solutions in many modern communication systems [9–17].

This book focuses on the turbo codes for 3GPP LTE-Advanced standard [15] and IEEE 802.16m standard [17]. In this chapter, we begin with an overview of turbo codec structure. We will outline the relevant specifications of these up-to-date applications and mention the marked differences between these turbo codes. Then the optimal decoding algorithm toward the best performance for each standard is presented. The fundamental principles of turbo codes will be illustrated here. For avoiding complicated operations in the original algorithm, we demonstrate several suboptimal methods and examine their performance. They can make the practical implementation easier but still keep great error correction capability. Further, we discuss the order of calculation steps in every algorithm due to its dominance over the decoding latency. The technique that utilizes auxiliary calculations and rearranges the schedule to let the decoder start executions before it receives the whole data block will be introduced. It also can reduce the hardware for storing temporary calculation results. These substantial benefits let this method be applied to almost all practical designs. The last topic is about shortening the average processing time. By using well-designed criteria, the decoding flow can finish earlier than normal without causing performance loss. Our discussions will involve two of the simplest criteria, and the simulation results will prove their effectiveness.

1.1 Turbo Codes: Parallel Concatenated Convolutional Codes

1.1.1 Principles of Encoding and Decoding

The classic turbo code is also known as the parallel concatenated convolutional code due to the special combination of two convolutional codes [5]. Figure 1.2 shows the typical framework, and its main characteristic is using interleavers in encoding and decoding procedures. The task of the interleaver is to permute its input sequence in a pseudo random way; while the de-interleaver can perform the inverse function to get the normal order back. In the turbo encoder, the interleaver is located between the two convolutional encoders. Thus, the first constituent encoder encodes the original information sequence $\boldsymbol{u}$ and get the first parity check $\boldsymbol{c}^{[\mathrm{P1}]}$, while the second one transforms reordered information sequence $\tilde{\boldsymbol{u}}$ into the second parity check $\boldsymbol{c}^{[\mathrm{P2}]}$. The codeword of turbo code is the concatenation of the first convolutional code ($\boldsymbol{u}$ and $\boldsymbol{c}^{[\mathrm{P1}]}$) and the second convolutional code ($\tilde{\boldsymbol{u}}$ and $\boldsymbol{c}^{[\mathrm{P2}]}$). Since $\tilde{\boldsymbol{u}}$ is just a permuted version of $\boldsymbol{u}$, the turbo encoder will send only $\boldsymbol{u}$ rather than both of them to its output, and this copy of information sequence $\boldsymbol{u}$ is usually called the systematic data $\boldsymbol{c}^{[\mathrm{U}]}$. A complete codeword sequence of turbo code contains three parts: $\boldsymbol{c}^{[\mathrm{U}]}$, $\boldsymbol{c}^{[\mathrm{P1}]}$, and $\boldsymbol{c}^{[\mathrm{P2}]}$. The modulator translates the codeword sequence into analog signals ($\boldsymbol{x}^{[\mathrm{U}]}$, $\boldsymbol{x}^{[\mathrm{P1}]}$, and $\boldsymbol{x}^{[\mathrm{P2}]}$) and then forward them to the channel. After the receiver gets these modulated signals, the demodulator produces the corresponding quantized data ($\boldsymbol{r}^{[\mathrm{U}]}$, $\boldsymbol{r}^{[\mathrm{P1}]}$, and $\boldsymbol{r}^{[\mathrm{P2}]}$) for the subsequent flows. Like encoder architecture, the turbo decoder contains two individual soft-in/soft-out (SISO) decoders, each of which deals with one constituent code. Based on the decoding algorithm for convolutional codes, the component decoder can calculate the probability of every information symbol. The first SISO decoder works with the first parity check part $\boldsymbol{r}^{[\mathrm{P1}]}$ and the systematic part $\boldsymbol{r}^{[\mathrm{U}]}$ to find the soft values of $\boldsymbol{u}$, whereas the second SISO decoder processes the second parity check part $\boldsymbol{r}^{[\mathrm{P2}]}$ and the interleaved systematic part $\tilde{\boldsymbol{r}}^{[\mathrm{U}]}$ to derive

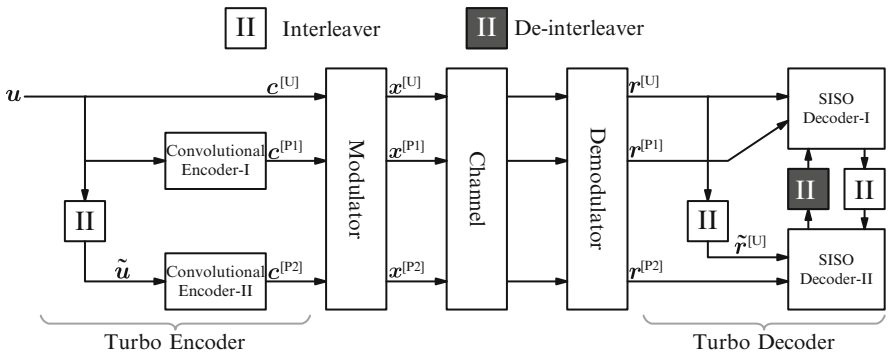

Fig. 1.2 Typical codec structure of the turbo code

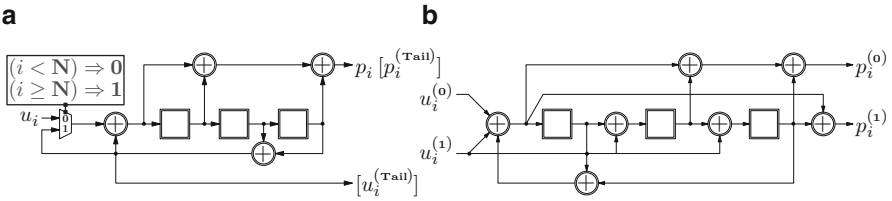

Fig. 1.3 Constituent convolutional encoders of turbo codes in advanced wireless communication systems. (**a**) In 3GPP LTE-Advanced standard. (**b**) In IEEE 802.16m standard

the soft values of $\tilde{\boldsymbol{u}}$. Thanks to the equivalence between $\boldsymbol{u}$ and $\tilde{\boldsymbol{u}}$, the outputs of either constituent code can be treated as the *a priori* probability estimation for the other one. The interleaver and de-interleaver between the two SISO decoders can rearrange these messages in proper order. This procedure will be performed alternatively between the two constituent codes until certain criteria are satisfied. With the random-like properties of turbo code, this iterative decoding method can be very efficient.

1.1.2 Turbo Codes in Advanced Communication Systems

Both 3GPP LTE-Advanced standard [15] and IEEE 802.16m standard [17] adopt turbo code as one of their channel coding techniques. The constituent encoders in either standard are two identical recursive convolutional encoders, and their architecture is shown in Fig. 1.3, where each mainly constructs of three 1-bit memory cells and several XOR gates. As the information symbols (u_i or $u_i^{(0)}u_i^{(1)}$) are fed into the encoder, the data in the memory will be updated, and the code symbols (p_i or $p_i^{(0)}\,p_i^{(1)}$) will be produced. There are some rules of initializing and finalizing the encoding procedure by setting the memory cells. In Fig. 1.4, the state transition diagrams depict the operations of these convolutional encoders. Here the contents of the memory cells (from left to right) are regarded as the states, and either case involves eight states: $\mathrm{S}^{(0)} = 000$, $\mathrm{S}^{(1)} = 001$, $\mathrm{S}^{(2)} = 010$, $\mathrm{S}^{(3)} = 011$, $\mathrm{S}^{(4)} = 100$, $\mathrm{S}^{(5)} = 101$, $\mathrm{S}^{(6)} = 110$ and $\mathrm{S}^{(7)} = 111$. These branches that indicate the possible changes in states are labeled with their respective input/output patterns (u_i / p_i or $u_i^{(0)}u_i^{(1)} / p_i^{(0)}\,p_i^{(1)}$). By serially connecting the state diagrams of successive input symbols, we can further obtain the *trellis diagram*. The encoding procedure of the constituent code can correspond to a unique trellis path on this diagram. Such a graphical representation also serves as a basis for the decoding procedure. Besides the realization of encoder, the turbo code construction requires the input block size **N**, the code rate **R**, and the interleaving rules. The specifications for the two applications state these requisite data in detail.

In 3GPP LTE-Advanced standard, the constituent encoder in Fig. 1.3a deals with one information bit u_i and outputs one parity check bit p_i per unit time. The initial values of all memory cells are zeroes. After inputting the last information bit $u_{\mathbf{N}-1}$, we change the selection signal of the multiplexer to force the input to the leftmost

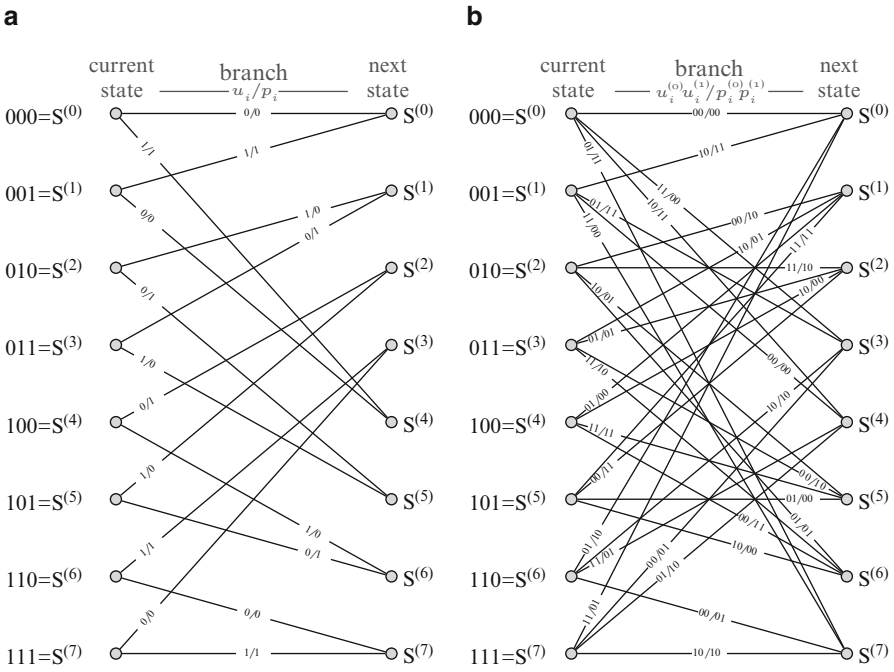

Fig. 1.4 State transition diagrams for the encoders in Fig. 1.3. (**a**) State transitions for Fig. 1.3a. (**b**) State transitions for Fig. 1.3b

memory cell to zero. The corresponding trellis diagram is given in Fig. 1.5, where the state at the initial trellis stage $\mathbf{T}_0$ and the state at the last trellis stage $\mathbf{T}_{\mathbf{N}+3}$ are both all-zero state $\mathrm{S}^{(0)}$. This termination step will take three more ticks to reset these memory cells and result in six extra tail bits. Hence, for every size-$\mathbf{N}$ information block $[u_0, u_1, \ldots, u_{\mathbf{N}-1}]$, the corresponding parity check sequence of one constituent code includes $[p_0, p_1, \ldots, p_{\mathbf{N}-1}]$ and $[u_{\mathbf{N}}^{(\mathrm{Tail})}, p_{\mathbf{N}}^{(\mathrm{Tail})}, u_{\mathbf{N}+1}^{(\mathrm{Tail})}, p_{\mathbf{N}+1}^{(\mathrm{Tail})}, u_{\mathbf{N}+2}^{(\mathrm{Tail})}, p_{\mathbf{N}+2}^{(\mathrm{Tail})}]$. Since the two constituent encoders will generate $(2\mathbf{N}+12)$ parity check bits in total, the code rate $\mathbf{R}$ of this turbo code is equal to $\mathbf{N}/(3\mathbf{N}+12)$. This specification defines 188 block sizes, ranging from 40 to 6144, and these $\mathbf{N}$'s can be summarized as follows:

$$\mathbf{N} = \begin{cases} 40+\ 8\times(\mathbb{k}-\ \ 0) & \text{for } \mathbb{k}=\ \ 0\sim\ 58, \\ 512+16\times(\mathbb{k}-\ 59) & \text{for } \mathbb{k}=\ 59\sim\ 90, \\ 1024+32\times(\mathbb{k}-\ 91) & \text{for } \mathbb{k}=\ 91\sim 122, \\ 2048+64\times(\mathbb{k}-123) & \text{for } \mathbb{k}=123\sim 187. \end{cases}$$

Every information block $[u_0, u_1, \ldots, u_{\mathbf{N}-1}]$ will be reordered into $[\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_{\mathbf{N}-1}]$ by the quadratic permutation polynomial (QPP) interleaver [18]. The relationship between the two data blocks is $\{\tilde{u}_i = u_{\mathcal{Q}(i)} \mid 0 \le i, \mathcal{Q}(i) < \mathbf{N}\}$, and their indexes ($i$ and $\mathcal{Q}(i)$) satisfy

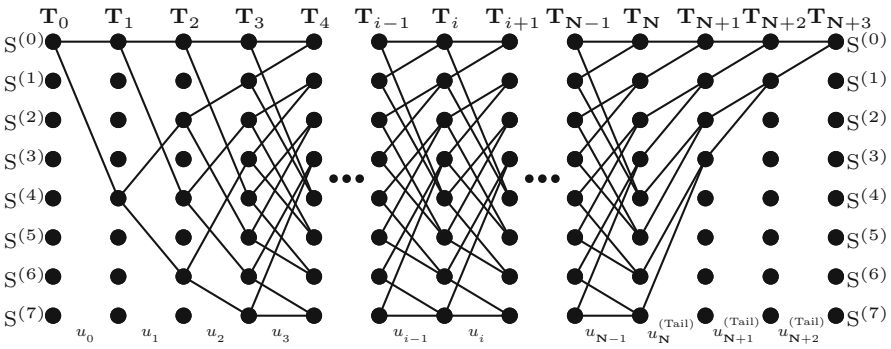

Fig. 1.5 The trellis diagram of constituent code in IEEE 802.16m standard

$$\mathcal{Q}(i) = f_1 \times i + f_2 \times i^2 \pmod{\mathbf{N}}, \tag{1.1}$$

where the determination of f_1 and f_2 is related to $\mathbf{N}$ [18]. The turbo encoders and decoders for this standard must support 188 sets of ($\mathbf{N}$, f_1, f_2). The parameters will share some characteristics. It is obvious that all $\mathbf{N}$'s are multiples of 8. For each ($\mathbf{N}$, f_1, f_2), f_1 is an odd number and is relatively prime to $\mathbf{N}$; while f_2 is an even number. These common characteristics are useful for practical implementation.

In IEEE 802.16m standard, the constituent encoder in Fig. 1.3b is a double binary convolutional encoder whose every input symbol consists of two information bits $(u_i^{(0)}, u_i^{(1)})$. It can get two parity check bits $(p_i^{(0)}, p_i^{(1)})$ per unit time. Note that block size $\mathbf{N}$ stands for the amount of input symbols, so there are $2\mathbf{N}$ bits in an information block in this case. This constituent encoder utilizes the tail-biting technique to let the final contents of the memory cells agree with their initial contents [19]. The encoding procedure is divided into two steps: the pre-encoding and the actual encoding. At the first step, the memory cells are reset to zeroes, and then all information symbols are input. Based on the final contents of the memory cells at the pre-encoding step, we calculate the corresponding initial values for the succeeding step and encode the same information block again. Only the parity check bits generated at the actual encoding process are valid code symbols. This method makes the trellis diagram become a circular structure as Fig. 1.6, where the initial trellis stage $\mathbf{T}_0$ and the last trellis stage $\mathbf{T}_{\mathbf{N}}$ are merged. For a size-$\mathbf{N}$ information block $[(u_0^{(0)}, u_0^{(1)}), (u_1^{(0)}, u_1^{(1)}), \ldots, (u_{\mathbf{N}-1}^{(0)}, u_{\mathbf{N}-1}^{(1)})]$, the corresponding parity check sequence of one constituent code includes $[(p_0^{(0)}, p_0^{(1)}), (p_1^{(0)}, p_1^{(1)}), \ldots, (p_{\mathbf{N}-1}^{(0)}, p_{\mathbf{N}-1}^{(1)})]$. Because the two constituent encoders will generate $4\mathbf{N}$ parity check bits in total, the code rate $\mathbf{R}$

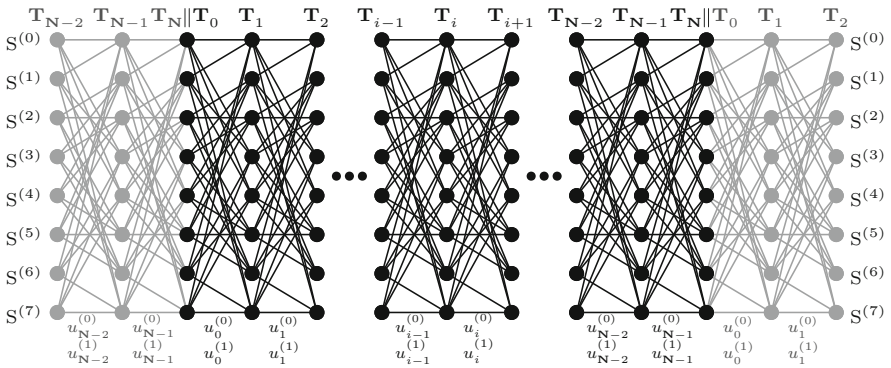

Fig. 1.6 The trellis diagram of constituent code in 3GPP LTE-Advanced standard

of this turbo code is equal to $2\mathbf{N}/6\mathbf{N}$ $(= 1/3)$. This specification defines 39 block sizes, ranging from 24 to 2400, and these $\mathbf{N}$'s are listed as follows:

$$\mathbf{N} = \left\{ \begin{array}{cccccccccc} 24, & 32, & 36, & 40, & 44, & 48, & 52, & 60, & 68, & 76, \\ 88, & 100, & 108, & 124, & 144, & 160, & 176, & 200, & 228, & 256, \\ 284, & 320, & 360, & 400, & 456, & 512, & 576, & 656, & 720, & 816, \\ 928, & 1056, & 1184, & 1312, & 1472, & 1664, & 1888, & 2112, & 2400. & \end{array} \right\}$$

In this turbo encoder, the interleaver first swaps the two component bits of each information symbol with odd-numbered index. This intra-symbol permutation results in $[(u_0'^{(0)}, u_0'^{(1)}), (u_1'^{(0)}, u_1'^{(1)}), \ldots, (u_{\mathbf{N}-1}'^{(0)}, u_{\mathbf{N}-1}'^{(1)})]$, a permuted data block with the property:

$$\left\{ \begin{array}{ll} (u_i'^{(0)}, u_i'^{(1)}) = (u_i^{(1)}, u_i^{(0)}) & \text{if } i \bmod 2 = 1, \\ (u_i'^{(0)}, u_i'^{(1)}) = (u_i^{(0)}, u_i^{(1)}) & \text{if } i \bmod 2 = 0. \end{array} \right. \tag{1.2}$$

After the production of $[(u_0'^{(0)}, u_0'^{(1)}), (u_1'^{(0)}, u_1'^{(1)}), \ldots, (u_{\mathbf{N}-1}'^{(0)}, u_{\mathbf{N}-1}'^{(1)})]$, there follows the almost regular permutation (ARP) interleaver [20] to further arrange this intermediate data block into $[(\tilde{u}_0^{(0)}, \tilde{u}_0^{(1)}), (\tilde{u}_1^{(0)}, \tilde{u}_1^{(1)}), \ldots, (\tilde{u}_{\mathbf{N}-1}^{(0)}, \tilde{u}_{\mathbf{N}-1}^{(1)})]$. The symbols of these data blocks satisfy $\{(\tilde{u}_i^{(0)}, \tilde{u}_i^{(1)}) = (u_{\mathcal{A}(i)}'^{(0)}, u_{\mathcal{A}(i)}'^{(1)}) \mid 0 \le i, \mathcal{A}(i) < \mathbf{N}\}$, and the index $\mathcal{A}(i)$ is derived by (1.3).

$$\mathcal{A}(i) = \left\{ \begin{array}{lll} \varepsilon \times i + g_0 \pmod{\mathbf{N}} & \text{if } i \bmod 4 = 0 \\ \varepsilon \times i + g_1 \pmod{\mathbf{N}} & \text{if } i \bmod 4 = 1 \\ \varepsilon \times i + g_2 \pmod{\mathbf{N}} & \text{if } i \bmod 4 = 2 \\ \varepsilon \times i + g_3 \pmod{\mathbf{N}} & \text{if } i \bmod 4 = 3 \end{array} \right. \text{and} \left\{ \begin{array}{l} g_0 = 1 \\ g_1 = 1 + g_1' + \frac{\mathbf{N}}{2} \\ g_2 = 1 + g_2' \\ g_3 = 1 + g_3' + \frac{\mathbf{N}}{2} \end{array} \right. \tag{1.3}$$

For each $\mathbf{N}$, there is a specific set of $(\varepsilon, g_1', g_2', g_3')$ in this standard. All of the 39 parameter sets also have some characteristics in common: every $\mathbf{N}$ is a multiple of 4; ε is an odd prime number; and the other three (g_1', g_2', g_3') are even numbers.

1.2 Decoding Procedure of Turbo Decoders

The main idea of turbo decoding process is the message propagation between the two constituent codes. In the first paper of turbo code [5], the maximum *a posteriori* probability (MAP) algorithm [21] is applied to the SISO decoder for calculating the soft values of each component convolutional code. These outputs of one SISO decoder should undergo some modifications before being passed to the other one. For facilitating the computation, the expressions of some variables used in the decoding algorithms may vary slightly according to the codec structure. Hence, the decoding algorithms for the 3GPP LTE-Advanced turbo codes and for IEEE 802.16m turbo codes are presented separately. Our discussions in either part start with the MAP algorithm and then demonstrate the iterative decoding flow. During the derivation, we only consider the first constituent code or the sequence in original order; and it can be easily generalized to the other component by substitution of notations.

1.2.1 MAP Algorithm for 3GPP LTE-Advanced Turbo Code

Based on the received data and the trellis structure, the SISO decoder can use the MAP algorithm to get the probabilities of all information symbols. Every information block $\boldsymbol{u} = [u_0, u_1, \ldots, u_{\mathbf{N}-1}]$ will be encoded into a parity check sequence $\boldsymbol{p} = [p_0, p_1, \ldots, p_{\mathbf{N}-1}]$. The constituent convolutional encoder will send out six more bits $[u_{\mathbf{N}}^{(\text{Tail})}, p_{\mathbf{N}}^{(\text{Tail})}, u_{\mathbf{N}+1}^{(\text{Tail})}, p_{\mathbf{N}+1}^{(\text{Tail})}, u_{\mathbf{N}+2}^{(\text{Tail})}, p_{\mathbf{N}+2}^{(\text{Tail})}]$ for trellis termination. These parts will make up the codeword block $\boldsymbol{c} = [c_0, c_1, \ldots, c_{\mathbf{N}+2}]$, where each symbol with the form $c_i = (c_i^{(0)}, c_i^{(1)})$ is (u_i, p_i) for $i = 0 \sim (\mathbf{N}-1)$ and represents $(u_i^{(\text{Tail})}, p_i^{(\text{Tail})})$ for $i = \mathbf{N} \sim (\mathbf{N}+2)$. Since the binary phase shift keying (BPSK) modulation is applied, the code bits $(c_i^{(0)}, c_i^{(1)})$ are translated into $x_i = (x_i^{(0)}, x_i^{(1)})$. The mapping between $c_i^{(j)}$ and $x_i^{(j)}$ for $i = 0 \sim (\mathbf{N}+2)$ and $j = 0 \sim 1$ is given in (1.4); then we have a modulated signal sequence $\boldsymbol{x} = [x_0, x_1, \ldots, x_{\mathbf{N}+2}]$.

$$x_i^{(j)} = 1 - 2 \times c_i^{(j)} = \begin{cases} +1 & \text{if } c_i^{(j)} = 0 \\ -1 & \text{if } c_i^{(j)} = 1 \end{cases} \tag{1.4}$$

The data received from the channel are denoted by $\boldsymbol{r} = [r_0, r_1, \ldots, r_{\mathbf{N}+2}]$ with $r_i = (r_i^{(0)}, r_i^{(1)})$. Under the assumption of additive white Gaussian noise (AWGN) channel, each $r_i^{(j)}$ can be viewed as the summation of the modulated signal $x_i^{(j)}$ and the zero-mean white Gaussian noise $n_i^{(j)}$ with variance σ^2.

$$r_i^{(j)} = x_i^{(j)} + n_i^{(j)} \tag{1.5}$$

The value of σ is determined by the bit signal-to-noise ratio (SNR), usually termed $\mathbf{E_b}/\mathbf{N_0}$. Given a transmitted codeword block $\boldsymbol{x}$, the probability of $\boldsymbol{r}$ can be expressed as

$$\Pr\left(r_0,\ldots,r_{\mathbf{N}+2} \mid x_0,\ldots,x_{\mathbf{N}+2}\right) \triangleq \prod_{i=0}^{\mathbf{N}+2} \Pr\left(r_i^{(0)}, r_i^{(1)} \mid x_i^{(0)}, x_i^{(1)}\right), \tag{1.6}$$

where

$$\Pr\left(r_i^{(0)}, r_i^{(1)} \mid x_i^{(0)}, x_i^{(1)}\right) \triangleq \prod_{j=0}^{1} \Pr\left(r_i^{(j)} \mid x_i^{(j)}\right) \tag{1.7}$$

and

$$\Pr\left(r_i^{(j)} \mid x_i^{(j)}\right) \triangleq \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{1}{2\sigma^2}\left(r_i^{(j)} - x_i^{(j)}\right)^2\right). \tag{1.8}$$

These probabilities provide the foundation for deriving the likelihood of each information bit.

The practical SISO decoder only has the knowledge about the received data sequence $\boldsymbol{r}$, so it will calculate the *a posteriori* probability (APP) of u_i as

$$\Pr\left(u_i \mid r_0,\ldots,r_{\mathbf{N}+2}\right), i = 0 \sim \mathbf{N}-1. \tag{1.9}$$

With the mapping in (1.4) and $u_i = c_i^{(0)}$ for $i = 0 \sim (\mathbf{N}-1)$, this probability is equivalent to the APP of $x_i^{(0)}$, and the set of possible values becomes from $u_i \in \{0, 1\}$ to $x_i^{(0)} \in \{+1, -1\}$. The SISO decoder will compare the APPs relating to $u_i = 0$ and $u_i = 1$ to make the decision on the i-th information bit. If $\Pr\left(u_i = 0 \mid r_0,\ldots,r_{\mathbf{N}+2}\right)$ is larger than or equal to $\Pr\left(u_i = 1 \mid r_0,\ldots,r_{\mathbf{N}+2}\right)$, the decision is 0; otherwise, the decision is 1. The log-likelihood ratio (LLR) in (1.10) is commonly used for the computation.

$$L(u_i) \triangleq \ln \frac{\Pr\left(u_i = 0 \mid r_0,\ldots,r_{\mathbf{N}+2}\right)}{\Pr\left(u_i = 1 \mid r_0,\ldots,r_{\mathbf{N}+2}\right)} \tag{1.10}$$

The hard decision u_i^* depends on the sign of $L(u_i)$:

$$u_i^* = \begin{cases} 0 & \text{if } L(u_i) \geq 0, \\ 1 & \text{if } L(u_i) < 0. \end{cases} \tag{1.11}$$

We can derive the LLR as (1.12) by the characteristic of conditional probability.

$$\begin{aligned} L(u_i) &= \ln \frac{\Pr\left(u_i = 0; r_0,\ldots,r_{\mathbf{N}+2}\right) / \Pr\left(r_0,\ldots,r_{\mathbf{N}+2}\right)}{\Pr\left(u_i = 1; r_0,\ldots,r_{\mathbf{N}+2}\right) / \Pr\left(r_0,\ldots,r_{\mathbf{N}+2}\right)} \\ &= \ln \frac{\Pr\left(u_i = 0; r_0,\ldots,r_{\mathbf{N}+2}\right)}{\Pr\left(u_i = 1; r_0,\ldots,r_{\mathbf{N}+2}\right)} \end{aligned} \tag{1.12}$$

Since $u_i = 0$ and $u_i = 1$ correspond to certain state transitions between the i-th trellis stage $\mathbf{T}_i$ and the $(i+1)$-th trellis stage $\mathbf{T}_{i+1}$, the probability of u_i can be written as

$$\Pr(u_i) = \sum_{(S_i, S_{i+1})} \Pr(u_i; S_i, S_{i+1}) = \sum_{(u_i; S_i, S_{i+1})} \Pr(S_i, S_{i+1}), \tag{1.13}$$

where S_i and S_{i+1} represent one of the eight states ($\mathrm{S}^{(0)} \sim \mathrm{S}^{(7)}$) at $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$ respectively; and (S_i, S_{i+1}) is the transition from S_i to S_{i+1}. The equation causes a modification to the LLR in (1.12):

$$\begin{aligned} L(u_i) &= \ln \frac{\sum_{(S_i, S_{i+1})} \Pr(u_i = 0; S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2})}{\sum_{(S_i, S_{i+1})} \Pr(u_i = 1; S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2})} \\ &= \ln \frac{\sum_{(u_i = 0; S_i, S_{i+1})} \Pr(S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2})}{\sum_{(u_i = 1; S_i, S_{i+1})} \Pr(S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2})}. \end{aligned} \tag{1.14}$$

The LLR calculation involves the joint probability $\Pr(S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2})$. Note that this probability will be zero as there is no branch linking S_i and S_{i+1}. It can be decomposed as (1.15) with Bayes' law.

$$\begin{aligned} \Pr(S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2}) &= \Pr(S_i; r_0, \ldots, r_{i-1}) \\ &\quad \times \Pr(S_{i+1}; r_i \mid S_i; r_0, \ldots, r_{i-1}) \\ &\quad \times \Pr(r_{i+1}, \ldots, r_{\mathbf{N}+2} \mid S_i, S_{i+1}; r_0, \ldots, r_i) \end{aligned} \tag{1.15}$$

We can simplify the two conditional probabilities in (1.15) by removing the redundant conditions. For $\Pr(S_{i+1}; r_i \mid S_i; r_0, \ldots, r_{i-1})$, when S_i is given, the transition to S_{i+1} with r_i is independent of previous data $(r_0, \ldots, r_{i-1})$. Similarly, the event $(r_{i+1}, \ldots, r_{\mathbf{N}+2})$ of the last conditional probability are affected by S_{i+1} only. The removal of unnecessary conditions results in (1.16) and (1.17).

$$\Pr(S_{i+1}; r_i \mid S_i; r_0, \ldots, r_{i-1}) = \Pr(S_{i+1}; r_i \mid S_i) \tag{1.16}$$

$$\Pr(r_{i+1}, \ldots, r_{\mathbf{N}+2} \mid S_i, S_{i+1}; r_0, \ldots, r_i) = \Pr(r_{i+1}, \ldots, r_{\mathbf{N}+2} \mid S_{i+1}) \tag{1.17}$$

Then the factorization of $\Pr(S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2})$ becomes

$$\Pr(S_i; r_0, \ldots, r_{i-1}) \times \Pr(S_{i+1}; r_i \mid S_i) \times \Pr(r_{i+1}, \ldots, r_{\mathbf{N}+2} \mid S_{i+1}). \tag{1.18}$$

Now we define the logarithmic version of the three probabilities as (1.19), (1.20), and (1.21).

$$\alpha(S_i) \triangleq \ln \Pr(S_i; r_0, \ldots, r_{i-1}) \tag{1.19}$$

$$\gamma(S_i, S_{i+1}) \triangleq \ln \Pr(S_{i+1}; r_i \mid S_i) \tag{1.20}$$

$$\beta(S_i) \triangleq \ln \Pr(r_i, \ldots, r_{\mathbf{N}+2} \mid S_i) \tag{1.21}$$

Thus, (1.18) can be expressed with these notations:

$$\Pr(S_i, S_{i+1}; r_0, \ldots, r_{\mathbf{N}+2}) = \exp\big(\alpha(S_i)\big) \times \exp\big(\gamma(S_i, S_{i+1})\big) \times \exp\big(\beta(S_{i+1})\big). \tag{1.22}$$

By substituting (1.22) for the APPs in (1.14), the LLR will be changed into

$$\begin{aligned} L(u_i) &= \ln \left[\sum_{(u_i=0; S_i, S_{i+1})} \exp\big(\alpha(S_i) + \gamma(S_i, S_{i+1}) + \beta(S_{i+1})\big) \right] \\ &\quad - \ln \left[\sum_{(u_i=1; S_i, S_{i+1})} \exp\big(\alpha(S_i) + \gamma(S_i, S_{i+1}) + \beta(S_{i+1})\big) \right]. \end{aligned} \tag{1.23}$$

Such a expression suggests that the SISO decoder should find out $\alpha(S_i)$, $\beta(S_{i+1})$, and $\gamma(S_i, S_{i+1})$ first.

In the MAP algorithm, there are efficient ways to compute (1.19)–(1.21). The $\alpha(S_i)$ can be obtained alternatively from its original definition:

$$\begin{aligned} \exp\big(\alpha(S_i)\big) &= \Pr(S_i; r_0, \ldots, r_{i-1}) \\ &= \sum_{S_{i-1}} \Pr(S_{i-1}, S_i; r_0, \ldots, r_{i-1}) \\ &= \sum_{S_{i-1}} \Pr(S_{i-1}; r_0, \ldots, r_{i-2}) \times \Pr(S_i; r_{i-1} \mid S_{i-1}; r_0, \ldots, r_{i-2}) \\ &= \sum_{S_{i-1}} \Pr(S_{i-1}; r_0, \ldots, r_{i-2}) \times \Pr(S_i; r_{i-1} \mid S_{i-1}) \\ &= \sum_{S_{i-1}} \exp\big(\alpha(S_{i-1})\big) \times \exp\big(\gamma(S_{i-1}, S_i)\big). \end{aligned} \tag{1.24}$$

The subsequent flow is taking the natural logarithm on both sides in (1.24).

$$\alpha(S_i) = \ln \sum_{S_{i-1}} \exp\big(\alpha(S_{i-1}) + \gamma(S_{i-1}, S_i)\big) \tag{1.25}$$

In the above equation, it just requires $\alpha(S_{i-1})$ and $\gamma(S_{i-1}, S_i)$ to compute $\alpha(S_i)$. This calculation is a recursive process from $i = 0$ to $i = \mathbf{N} + 3$; hence, $\alpha(S_i)$ is also named the *forward metric*. It needs an appropriate initial value $\alpha(S_0)$.

From Fig. 1.5, the trellis for the constituent code starts at $\mathrm{S}^{(0)}$. The condition implies that $\Pr(S_0 = \mathrm{S}^{(0)}) = 1$ and $\Pr(S_0 \neq \mathrm{S}^{(0)}) = 0$. Here we make use of (1.19) again and get

$$\alpha(S_0) = \begin{cases} 0 & \text{if } S_0 = \mathrm{S}^{(0)}, \\ -\infty & \text{if } S_0 \neq \mathrm{S}^{(0)}. \end{cases} \tag{1.26}$$

The derivation of $\beta(S_i)$ is much the same as that of $\alpha(S_i)$. The first step is

$$\begin{aligned} \exp\big(\beta(S_i)\big) &= \Pr(r_i, \ldots, r_{\mathrm{N}+2} \mid S_i) \\ &= \sum_{S_{i+1}} \Pr(S_{i+1}; r_i, \ldots, r_{\mathrm{N}+2} \mid S_i) \\ &= \sum_{S_{i+1}} \Pr(S_{i+1}; r_i \mid S_i) \times \Pr(r_{i+1}, \ldots, r_{\mathrm{N}+2} \mid S_i, S_{i+1}; r_i) \\ &= \sum_{S_{i+1}} \Pr(S_{i+1}; r_i \mid S_i) \times \Pr(r_{i+1}, \ldots, r_{\mathrm{N}+2} \mid S_{i+1}) \\ &= \sum_{S_{i+1}} \exp\big(\gamma(S_i, S_{i+1})\big) \times \exp\big(\beta(S_{i+1})\big). \end{aligned} \tag{1.27}$$

After the computation of natural logarithm, (1.27) changes to

$$\beta(S_i) = \ln \sum_{S_{i+1}} \exp\big(\gamma(S_i, S_{i+1}) + \beta(S_{i+1})\big). \tag{1.28}$$

The calculation of $\beta(S_i)$ needs both $\beta(S_{i+1})$ and $\gamma(S_i, S_{i+1})$, and it is performed recursively in descending order; so we call $\beta(S_i)$ the *backward metric*. The trellis diagram in Fig. 1.5 whose the last state $S_{\mathrm{N}+3}$ is $\mathrm{S}^{(0)}$ indicates the initialization:

$$\beta(S_{\mathrm{N}+3}) = \begin{cases} 0 & \text{if } S_{\mathrm{N}+3} = \mathrm{S}^{(0)}, \\ -\infty & \text{if } S_{\mathrm{N}+3} \neq \mathrm{S}^{(0)}. \end{cases} \tag{1.29}$$

Furthermore, the $\gamma(S_i, S_{i+1})$ in (1.20) can be

$$\begin{aligned} \exp\big(\gamma(S_i, S_{i+1})\big) &= \Pr(S_{i+1}; r_i \mid S_i) \\ &= \frac{\Pr(S_i, S_{i+1}; r_i)}{\Pr(S_i)} \\ &= \frac{\Pr(S_i, S_{i+1})}{\Pr(S_i)} \times \frac{\Pr(S_i, S_{i+1}; r_i)}{\Pr(S_i, S_{i+1})} \\ &= \Pr(S_{i+1} \mid S_i) \times \Pr(r_i \mid S_i, S_{i+1}) \\ &= \Pr(u_i) \times \Pr(r_i \mid x_i). \end{aligned} \tag{1.30}$$

The relationship among u_i, $c_i^{(0)}$, and $x_i^{(0)}$ promises that we can interchange $\Pr(u_i)$ with $\Pr\left(x_i^{(0)}\right)$. To find the $\Pr(u_i)$, we need the *a priori* information with the definition as

$$L_a(u_i) \triangleq \ln \frac{\Pr(u_i = 0)}{\Pr(u_i = 1)} = L_a(x_i^{(0)}) \triangleq \ln \frac{\Pr\left(x_i^{(0)} = +1\right)}{\Pr\left(x_i^{(0)} = -1\right)}. \tag{1.31}$$

The $L_a(x_i^{(0)})$ is utilized to calculate the *a priori* probability $\Pr\left(x_i^{(0)}\right)$:

$$\begin{aligned}
\Pr\left(x_i^{(0)} = \pm 1\right) &= \frac{\exp\left(\pm L_a(x_i^{(0)})\right)}{1 + \exp\left(\pm L_a(x_i^{(0)})\right)} \\
&= \underbrace{\left[\frac{\exp\left(-L_a(x_i^{(0)})/2\right)}{1 + \exp\left(-L_a(x_i^{(0)})\right)}\right]}_{\boldsymbol{\vartheta_i}} \times \exp\left(x_i^{(0)} \times L_a(x_i^{(0)})/2\right) \\
&= \boldsymbol{\vartheta}_i \times \exp\left(x_i^{(0)} \times L_a(x_i^{(0)})/2\right),
\end{aligned} \tag{1.32}$$

where $\boldsymbol{\vartheta}_i$ depends only on $L_a(x_i^{(0)})$ and remains constant for either $x_i^{(0)} = +1$ or $x_i^{(0)} = -1$. Based on (1.7) and (1.8), the $\Pr(r_i \mid x_i)$ can be modified to

$$\begin{aligned}
\Pr(r_i \mid x_i) &= \prod_{j=0}^{1}\left[\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{1}{2\sigma^2}(r_i^{(j)} - x_i^{(j)})^2\right)\right] \\
&= \underbrace{\left(\frac{1}{\sqrt{2\pi}\,\sigma}\right)^2 \times \exp\left(-\frac{1}{2\sigma^2}\sum_{j=0}^{1}\left[(r_i^{(j)})^2 + (x_i^{(j)})^2\right]\right)}_{\boldsymbol{\varphi_i}} \\
&\quad \times \exp\left(\frac{1}{\sigma^2}\sum_{j=0}^{1}\left[r_i^{(j)} \times x_i^{(j)}\right]\right) \\
&= \boldsymbol{\varphi}_i \times \exp\left(\frac{1}{2}L_c\sum_{j=0}^{1}\left[r_i^{(j)} \times x_i^{(j)}\right]\right).
\end{aligned} \tag{1.33}$$

For all possible combinations of $(x_i^{(0)}, x_i^{(1)})$, $r_i^{(j)}$ is the same, and the square of $x_i^{(j)}$ is 1; so $\boldsymbol{\varphi}_i$ will be a constant. In addition, the channel reliability value L_c whose value is $2/\sigma^2$ is introduced in (1.33). It will equal to $4\mathbf{RE_b}/\mathbf{N_0}$ in the AWGN channel [22]. With (1.32) and (1.33), we can rewritten (1.30) as

$$\gamma'(S_i, S_{i+1}) = \ln \boldsymbol{\vartheta}_i + \ln \boldsymbol{\varphi}_i + \frac{1}{2}x_i^{(j)} \times L_a(x_i^{(j)}) + \frac{1}{2}L_c\sum_{j=0}^{1}\left[r_i^{(j)} \times x_i^{(j)}\right]. \tag{1.34}$$

Both $\boldsymbol{\vartheta}_i$ and $\boldsymbol{\varphi}_i$ will be canceled out in the LLR calculation in (1.23). For this reason, we drop out $\boldsymbol{\vartheta}_i$ and $\boldsymbol{\varphi}_i$ in advance and define $\gamma(S_i, S_{i+1})$, also known as the *branch metric*, in (1.35).

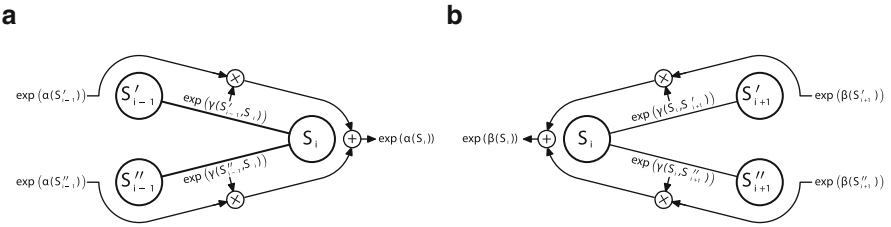

Fig. 1.7 Forward metric calculation and backward metric calculation. (**a**) Recursive $\alpha(S_i)$ computation. (**b**) Recursive $\beta(S_i)$ computation

$$\gamma(S_i, S_{i+1}) = \frac{1}{2} x_i^{(j)} L_a(x_i^{(j)}) + \frac{1}{2} L_c \sum_{j=0}^{1} \left[r_i^{(j)} \times x_i^{(j)} \right] \tag{1.35}$$

Consequently, the SISO decoder exploits the MAP algorithm over $(\mathbf{N} + 3)$ received codeword symbols to get the probabilities of $\mathbf{N}$ information bits. The initial conditions of this component convolutional code are $\alpha(S_0)$ and $\beta(S_{\mathbf{N}+3})$. After receiving r_i, the decoder can derive $\gamma(S_i, S_{i+1})$ of each branch between trellis stages $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$. Then the decoder uses these branch metrics to calculate forward metrics and backward metrics in a recursive way. Figure 1.7 is the graphical description of these computations. In this case, every state S_i at $\mathbf{T}_i$ has exactly two incoming branches connecting to two different states (S'_{i-1} and S''_{i-1}) at $\mathbf{T}_{i-1}$ and has exactly two outgoing branches connecting to two different states (S'_{i+1} and S''_{i+1}) at $\mathbf{T}_{i+1}$. As $\alpha(S_i)$ and $\beta(S_{i+1})$ are available, the $L(u_i)$ and decision u_i^* can be further determined.

1.2.2 Iterative Flow for 3GPP LTE-Advanced Turbo Code

Like the MAP algorithm, the discussion about the iterative decoding flow relies heavily on the APP of the i-th information bit u_i. Instead of the definition in (1.9), we take the $\Pr(u_i; r_0, \ldots, r_{\mathbf{N}+2})$ into account here, for both them can result in the same LLR as (1.12). By applying Bayes' law, the joint probability can be factorized into (1.36).

$$\begin{aligned}
\Pr(u_i; \boldsymbol{r}) &= \sum_{\boldsymbol{u}:u_i} \Pr\left(u_0, \ldots, u_{\mathbf{N}-1}; r_0^{(0)}, r_0^{(1)}, \ldots, r_{\mathbf{N}+2}^{(0)}, r_{\mathbf{N}+2}^{(1)}\right) \\
&= \sum_{\boldsymbol{u}:u_i} \Bigg[\Pr(u_0, \ldots, u_{\mathbf{N}-1}) \times \Pr\left(r_0^{(0)}, \ldots, r_{\mathbf{N}+2}^{(0)} \mid u_0, \ldots, u_{\mathbf{N}-1}\right) \\
&\quad \times \Pr\left(r_0^{(1)}, \ldots, r_{\mathbf{N}+2}^{(1)} \mid u_0, \ldots, u_{\mathbf{N}-1}; r_0^{(0)}, \ldots, r_{\mathbf{N}+2}^{(0)}\right) \Bigg]
\end{aligned} \tag{1.36}$$

Since the constituent encoder generates an unique modulated codeword sequence $[x_0^{(0)}, x_0^{(1)}, \ldots, x_{\mathbf{N}+2}^{(0)}, x_{\mathbf{N}+2}^{(1)}]$ from every information block $[u_0, \ldots, u_{\mathbf{N}-1}]$, this one-to-one mapping allows for the interchangeability in (1.37)–(1.39).

$$\begin{aligned} \Pr(u_0, \ldots, u_{\mathbf{N}-1}) &= \Pr\left(x_0^{(0)}, \ldots, x_{\mathbf{N}-1}^{(0)}\right) && (1.37)\\ &= \Pr\left(x_0^{(0)}, \ldots, x_{\mathbf{N}-1}^{(0)}, x_{\mathbf{N}}^{(0)}, x_{\mathbf{N}+1}^{(0)}, x_{\mathbf{N}+2}^{(0)}\right) && (1.38)\\ &= \Pr\left(x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}; x_0^{(1)}, \ldots, x_{\mathbf{N}+2}^{(1)}\right) && (1.39) \end{aligned}$$

With the relations of equality, we can modify the $\Pr(u_i; \boldsymbol{r})$ from (1.36) to

$$\begin{aligned} \Pr(u_i; \boldsymbol{r}) &= \sum_{\boldsymbol{u}:u_i} \Bigg[\Pr\left(x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}\right) \times \Pr\left(r_0^{(0)}, \ldots, r_{\mathbf{N}+2}^{(0)} \mid x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}\right) \\ &\quad \times \Pr\left(r_0^{(1)}, \ldots, r_{\mathbf{N}+2}^{(1)} \mid x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}; x_0^{(1)}, \ldots, x_{\mathbf{N}+2}^{(1)}; r_0^{(0)}, \ldots, r_{\mathbf{N}+2}^{(0)}\right) \Bigg] \\ &= \sum_{\boldsymbol{u}:u_i} \Bigg[\Pr\left(x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}\right) \times \Pr\left(r_0^{(0)}, \ldots, r_{\mathbf{N}+2}^{(0)} \mid x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}\right) \\ &\quad \times \Pr\left(r_0^{(1)}, \ldots, r_{\mathbf{N}+2}^{(1)} \mid x_0^{(1)}, \ldots, x_{\mathbf{N}+2}^{(1)}\right) \Bigg], \end{aligned} \tag{1.40}$$

where the probability about $[r_0^{(1)}, \ldots, r_{\mathbf{N}+2}^{(1)}]$ ignores the conditions $[x_0^{(0)}, \ldots, x_{\mathbf{N}+2}^{(0)}]$ and $[r_0^{(0)}, \ldots, r_{\mathbf{N}+2}^{(0)}]$ while $[x_0^{(1)}, \ldots, x_{\mathbf{N}+2}^{(1)}]$ is given. For the discrete memoryless channel, (1.41) is another expression of $\Pr(u_i; \boldsymbol{r})$. In (1.42), the probabilities directly relating to u_i, $x_i^{(0)}$, and $r_i^{(0)}$ bit are taken out; and the rest excludes the i-th systematic data.

$$\Pr(u_i; \boldsymbol{r}) = \sum_{\boldsymbol{u}:u_i} \left[\prod_{j=0}^{\mathbf{N}+2} \Pr\left(x_j^{(0)}\right) \times \Pr\left(r_j^{(0)} \mid x_j^{(0)}\right) \times \Pr\left(r_j^{(1)} \mid x_j^{(1)}\right) \right] \tag{1.41}$$

$$\begin{aligned} &= \Pr\left(x_i^{(0)}\right) \times \Pr\left(r_i^{(0)} \mid x_i^{(0)}\right) \\ &\quad \times \sum_{\boldsymbol{u}:u_i} \left[\prod_{\substack{j=0 \\ j \neq i}}^{\mathbf{N}+2} \Pr\left(x_j^{(0)}\right) \times \Pr\left(r_j^{(0)} \mid x_j^{(0)}\right) \times \prod_{l=0}^{\mathbf{N}+2} \Pr\left(r_l^{(1)} \mid x_l^{(1)}\right) \right] \end{aligned} \tag{1.42}$$

Now we substitute (1.42) for the probabilities in (1.12) and then derive the LLR of i-th information bit as (1.43), where the last term is called the extrinsic information $L_e(u_i)$.

$$
\begin{aligned}
L(u_i) &= \ln \frac{\Pr(u_i = 0; \boldsymbol{r})}{\Pr(u_i = 1; \boldsymbol{r})} \\
&= \ln \frac{\Pr\left(x_i^{(0)} = +1\right)}{\Pr\left(x_i^{(0)} = -1\right)} + \ln \frac{\Pr\left(r_i^{(0)} \mid x_i^{(0)} = +1\right)}{\Pr\left(r_i^{(0)} \mid x_i^{(0)} = -1\right)} \\
&\quad + \underbrace{\ln \frac{\displaystyle\sum_{\boldsymbol{u}:u_i=0} \left[\prod_{\substack{j=0 \\ j\neq i}}^{\mathrm{N}+2} \Pr\left(x_j^{(0)}\right) \times \Pr\left(r_j^{(0)} \mid x_j^{(0)}\right) \times \prod_{l=0}^{\mathrm{N}+2} \Pr\left(r_l^{(1)} \mid x_l^{(1)}\right) \right]}{\displaystyle\sum_{\boldsymbol{u}:u_i=1} \left[\prod_{\substack{j=0 \\ j\neq i}}^{\mathrm{N}+2} \Pr\left(x_j^{(0)}\right) \times \Pr\left(r_j^{(0)} \mid x_j^{(0)}\right) \times \prod_{l=0}^{\mathrm{N}+2} \Pr\left(r_l^{(1)} \mid x_l^{(1)}\right) \right]}}_{L_e(u_i)}
\end{aligned} \tag{1.43}
$$

The first term in (1.43) is $L_a(u_i)$ in (1.31); the second term can also be written as (1.44) by the definition in (1.8) and $L_c = 2/\sigma^2$.

$$
\begin{aligned}
\ln \frac{\Pr\left(r_i^{(0)} \mid x_i^{(0)} = +1\right)}{\Pr\left(r_i^{(0)} \mid x_i^{(0)} = -1\right)} &= \ln \frac{\frac{1}{\sqrt{2\pi}\,\sigma} \times \exp\left(-\frac{1}{2\sigma^2}(r_i^{(0)} - 1)^2\right)}{\frac{1}{\sqrt{2\pi}\,\sigma} \times \exp\left(-\frac{1}{2\sigma^2}(r_i^{(0)} + 1)^2\right)} \\
&= \frac{1}{2\sigma^2} \times \left[(r_i^{(0)} + 1)^2 - (r_i^{(0)} - 1)^2\right] \\
&= \frac{2}{\sigma^2} \times r_i^{(0)} = L_c \times r_i^{(0)}
\end{aligned} \tag{1.44}
$$

As a result, $L(u_i)$ is regarded as the summation of *a priori* probability, the weighted received data, and the extrinsic information:

$$
L(u_i) = L_a(u_i) + L_c \times r_i^{(0)} + L_e(u_i). \tag{1.45}
$$

Figure 1.8 shows the detailed decoding procedure for passing soft values from one SISO decoder to the other. The SISO decoders can update useful estimation about every information bit u_i with the applications of (1.23) and (1.45). In general, the first SISO decoder performs the MAP algorithm on the systematic part and first parity check part to compute all necessary metrics. We assume that $\Pr(u_i = 0)$ and $\Pr(u_i = 1)$ are both $1/2$; thus, the *a priori* value $L_{a1}(u_i)$ which is involved in the $\gamma(S_i, S_{i+1})$ calculation is initialized with 0. This SISO decoder will output $L_1(u_i)$, the LLR of the i-th information bit. From (1.45), the corresponding extrinsic information $L_{e1}(u_i)$ can be obtained by

$$
L_{e1}(u_i) = L_1(u_i) - L_c \times r_i^{(0)} - L_{a1}(u_i). \tag{1.46}
$$

Since $L_{e1}(u_i)$ is derived from the whole codeword except the i-th systematic data, it can provide newer information than the probabilities calculated from $r_i^{(0)}$ and $L_{a1}(u_i)$ directly. We use $L_{e1}(u_i)$ as an estimate for the *a priori* value of the other constituent code. The second SISO decoder also does the trellis-based decoding

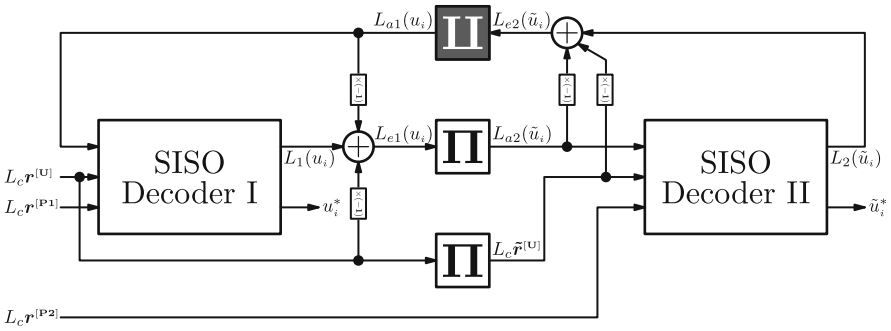

Fig. 1.8 The propagation of soft values in the iterative decoding flow

procedure with $\tilde{\boldsymbol{r}}^{[\mathbf{U}]}$, $\boldsymbol{r}^{[\mathbf{P2}]}$, and $L_{a2}(\tilde{u}_i)$. Then it evaluates $L_2(\tilde{u}_i)$, the LLR for the permuted information sequence, and it further gets the extrinsic information $L_{e2}(\tilde{u}_i)$ of the second constituent code by

$$L_{e2}(\tilde{u}_i) = L_2(\tilde{u}_i) - L_c \times \tilde{r}_i^{(0)} - L_{a2}(\tilde{u}_i). \tag{1.47}$$

The $L_{e2}(\tilde{u}_i)$ is passed back to the first SISO decoder. A de-interleaver rearranges the sequence order so that $L_{e2}(\tilde{u}_i)$ can be the *a priori* value $L_{a1}(u_{\mathcal{Q}(i)})$ for the first constituent code. After updating the $L_{a1}(u_i)$ for $i = 0 \sim (\mathbf{N} - 1)$, the first SISO decoder performs the MAP algorithm again. Such soft value calculation of each constituent code is named as a half-iteration, and two successive processes form one complete iteration. Usually, we will set a maximum iteration number $\mathcal{I}$ as the conventional stopping criterion. At the last half-iteration, the decoder makes hard decisions from the LLRs by (1.11). Note that the tail bits of both constituent codes are excluded in the message propagation; their probabilities remain the same throughout the iterative flow, and we only need to calculate the corresponding metrics once. Because the correlation between APP estimations and received data will become stronger after every half-iteration, the benefit of extrinsic information diminishes gradually. This phenomenon leads to very small enhancement in performance when the decoding flow has proceeded repeatedly for certain iterations. Hence, the determination of $\mathcal{I}$ depends on the trade-off between performance requirement and processing time.

The bit error rate (BER) versus SNR diagrams in Fig. 1.9 present how the performance of 3GPP LTE-Advanced turbo code with $\mathbf{N} = \{40, 512, 6144\}$ varies as the iteration increases from 1 to 16. The simulation results show that the iterative message propagation can provide greater error correction capability in larger blocks than in small blocks. Moreover, it is easy to notice that there is slighter performance gain at higher iterations, especially when $\mathbf{N}$ is small. In spite of the excellent performance at $\mathcal{I} = 16$, the decoding latency is too long. Considering the improvement, it is inefficient to use so many iterations for all blocks. From Fig. 1.9c, the result at $\mathcal{I} = 16$ is merely 0.1 dB better than at $\mathcal{I} = 8$, but its process

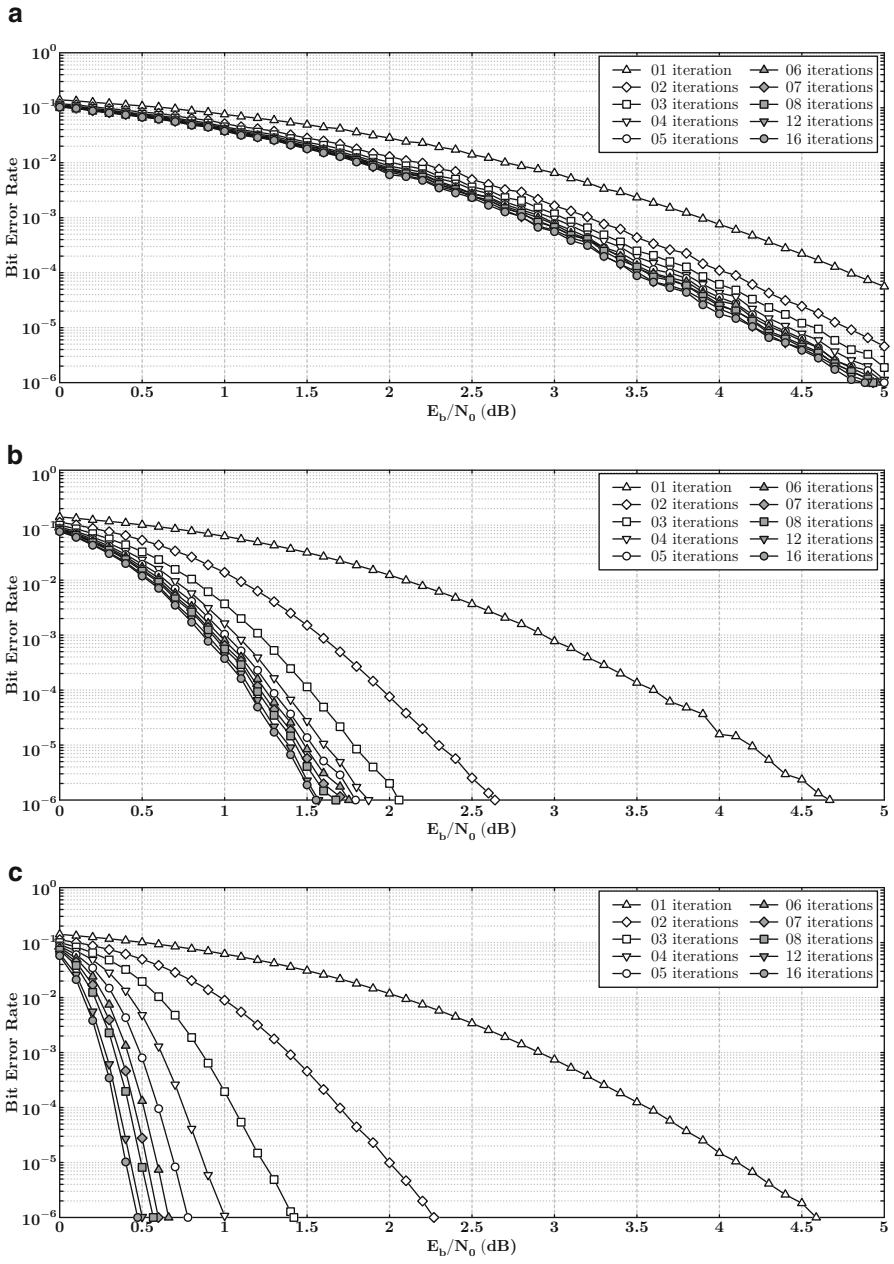

Fig. 1.9 Performance of 3GPP LTE-Advanced turbo code with various block sizes and iteration numbers. (**a**) 3GPP LTE-Advanced turbo code: $\mathbf{N} = 40$ and $\mathcal{I} = \{1, 2, 3, 4, 5, 6, 7, 8, 12, 16\}$. (**b**) 3GPP LTE-Advanced turbo code: $\mathbf{N} = 512$ and $\mathcal{I} = \{1, 2, 3, 4, 5, 6, 7, 8, 12, 16\}$. (**c**) 3GPP LTE-Advanced turbo code: $\mathbf{N} = 6144$ and $\mathcal{I} = \{1, 2, 3, 4, 5, 6, 7, 8, 12, 16\}$

takes double time. Thus, choosing $\mathcal{I} = 8$ is more suitable for $\mathbf{N} = 6144$ in this application. Similarly, based on Figs. 1.9a and 1.9b, we can use $\mathcal{I} = 4$ for $\mathbf{N} = 40$ and $\mathcal{I} = 5$ for $\mathbf{N} = 512$. In summary, the 3GPP LTE-Advanced turbo decoder generally needs four to eight iterations for all $\mathbf{N}$'s in the standard.

1.2.3 MAP Algorithm for IEEE 802.16m Turbo Code

To support the double binary codec structure and tail-biting technique of this turbo code, we must make some modifications to the MAP algorithm. The first step is redefinition of notations. The encoder will transform a size-$\mathbf{N}$ information block $\boldsymbol{u} = [u_0, u_1, \ldots, u_{\mathbf{N}-1}]$ with $u_i = (u_i^{(0)}, u_i^{(1)})$ into $\mathbf{N}$ pairs of parity check bits: $\boldsymbol{p} = [(p_0^{(0)}, p_0^{(1)}), (p_1^{(0)}, p_1^{(1)}), \ldots, (p_{\mathbf{N}-1}^{(0)}, p_{\mathbf{N}-1}^{(1)})]$. The two parts form the constituent codeword $\boldsymbol{c} = [c_0, c_1, \ldots, c_{\mathbf{N}-1}]$, of which every symbol $c_i = (c_i^{(0)}, c_i^{(1)}, c_i^{(2)}, c_i^{(3)})$ represents $(u_i^{(0)}, u_i^{(1)}, p_i^{(0)}, p_i^{(1)})$. According to (1.4), the code bit $c_i^{(j)}$ is mapped into a BPSK signal $x_i^{(j)}$ for $i = 0 \sim (\mathbf{N}-1)$ and $j = 0 \sim 3$. The modulated signal sequence $\boldsymbol{x} = [x_0, x_1, \ldots, x_{\mathbf{N}-1}]$ with $x_i = (x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, x_i^{(3)})$ will be passed into the channel, and each signal $x_i^{(j)}$ may suffer from noise $n_i^{(j)}$ as (1.5) during the transmission. Then the SISO decoder will receive $\boldsymbol{r} = [r_0, r_1, \ldots, r_{\mathbf{N}-1}]$ with $r_i = (r_i^{(0)}, r_i^{(1)}, r_i^{(2)}, r_i^{(3)})$ from the channel. Therefore, the probability of $\boldsymbol{r}$ can be derived as (1.48) by changing the upper bounds of i and j in (1.6) and (1.7).

$$\begin{aligned}\Pr\left(r_0, \ldots, r_{\mathbf{N}-1} \mid x_0, \ldots, x_{\mathbf{N}-1}\right) &\triangleq \prod_{i=0}^{\mathbf{N}-1} \Pr\left(r_i^{(0)}, r_i^{(1)}, r_i^{(2)}, r_i^{(3)} \mid x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, x_i^{(3)}\right) \\ &= \prod_{i=0}^{\mathbf{N}-1} \left[\prod_{j=0}^{3} \Pr\left(r_i^{(j)} \mid x_i^{(j)}\right) \right] \end{aligned} \tag{1.48}$$

The APP becomes (1.49) as well.

$$\Pr\left(u_i \mid r_0, \ldots, r_{\mathbf{N}-1}\right) = \Pr\left(u_i^{(0)}, u_i^{(1)} \mid r_0, \ldots, r_{\mathbf{N}-1}\right), \quad i = 0 \sim \mathbf{N}-1. \tag{1.49}$$

The combination of $u_i = (u_i^{(0)}, u_i^{(1)})$ with the largest possibility will be selected as the decisions. We often use LLRs rather than APPs to make this comparison. For simplicity, we define the following 2-tuple representations:

$$\{\mathcal{U}(\ell) = (\ell^{(0)}, \ell^{(1)}) \mid \mathcal{U}(0) = (0,0), \mathcal{U}(1) = (0,1), \mathcal{U}(2) = (1,0), \mathcal{U}(3) = (1,1)\}.$$

Thus, the LLR for the double binary scheme will be (1.50) with $\ell = 0 \sim 3$. The APP of $u_i = (0,0)$ in the denominator is served as the reference value.

$$\begin{aligned} L^{[\ell]}(u_i) &\triangleq \ln \frac{\Pr\left(u_i = \mathcal{U}(\ell) \mid r_0, \ldots, r_{\mathbf{N}-1}\right)}{\Pr\left(u_i = \mathcal{U}(0) \mid r_0, \ldots, r_{\mathbf{N}-1}\right)} \\ &= \ln \frac{\Pr\left(u_i = \mathcal{U}(\ell); r_0, \ldots, r_{\mathbf{N}-1}\right)}{\Pr\left(u_i = \mathcal{U}(0); r_0, \ldots, r_{\mathbf{N}-1}\right)} \end{aligned} \tag{1.50}$$

With these LLRs, we can get the hard decisions u_i^* by

$$u_i^* = \mathcal{U}(\ell'), \quad \text{if } L^{[\ell']}(u_i) = \max\left(L^{[0]}(u_i), L^{[1]}(u_i), L^{[2]}(u_i), L^{[3]}(u_i)\right). \tag{1.51}$$

Since the same principle can apply here, the LLR can be also modified into the equation in (1.23) while changing the number of received symbols to **N**. The definitions of $\alpha(S_i)$ and $\beta(S_i)$ are still (1.25) and (1.28) respectively; but it needs distinct initial values for the trellis path with undetermined beginning state and ending state. To solve this problem, we assume that each $\Pr(S_0 = \mathrm{S}^{(m)}) = 1/8$ and $\Pr(S_\mathbf{N} = \mathrm{S}^{(m)}) = 1/8$ for $m = 0 \sim 7$; and we introduces an auxiliary forward metric $\alpha'(S_i)$ with $\alpha'(S_0 = \mathrm{S}^{(m)}) = 0$ and an auxiliary backward metric $\beta'(S_i)$ with $\beta'(S_\mathbf{N} = \mathrm{S}^{(m)}) = 0$. Then we compute $\alpha'(S_i)$ in a forward recursive behavior until we get $\alpha'(S_\mathbf{N})$. The property $S_0 = S_\mathbf{N}$ allows us to treat $\alpha'(S_\mathbf{N})$ as $\alpha(S_0)$ in the circular trellis structure. For backward metric, the initial $\beta(S_\mathbf{N})$ can be updated with $\beta'(S_0)$, the final results of auxiliary computation. In addition to the initialization, there is a small change in the branch metric calculation. The basic calculation of $\gamma(S_i, S_{i+1})$ is identical to (1.30). However, the $\Pr(u_i)$ and $\Pr(r_i \mid x_i)$ must be redefined to match the current version of constituent codeword. The new expressions will cover all four cases of the i-th symbol. We first give the *a priori* value as (1.52) whose format agrees with that of LLR.

$$L_a^{[\ell]}(u_i) \triangleq \ln \frac{\Pr(u_i = \mathcal{U}(\ell))}{\Pr(u_i = \mathcal{U}(0))}, \quad \ell = 0 \sim 3 \tag{1.52}$$

Then the probability of $u_i = \mathcal{U}(\ell)$ is determined by

$$\begin{aligned} \Pr(u_i = \mathcal{U}(\ell)) &= \underbrace{\left[\sum_{j=0}^{3} \exp\left(L_a^{[j]}(u_i)\right)\right]^{-1}}_{\vartheta_i'} \times \exp\left(L_a^{[\ell]}(u_i)\right) \\ &= \vartheta_i' \times \exp\left(L_a^{[\ell]}(u_i)\right), \end{aligned} \tag{1.53}$$

where ϑ_i' is fixed for any ℓ. We also renew the index of $\Pr(r_i \mid x_i)$ as

$$\begin{aligned} \Pr(r_i \mid x_i) &= \underbrace{\left(\frac{1}{\sqrt{2\pi}\,\sigma}\right)^4 \times \exp\Big(-\frac{1}{2\sigma^2}\sum_{j=0}^{3}\left[(r_i^{(j)})^2 + (x_i^{(j)})^2\right]\Big)}_{\varphi_i'} \\ &\quad \times \exp\Big(\frac{1}{\sigma^2}\sum_{j=0}^{3}\left[r_i^{(j)} \times x_i^{(j)}\right]\Big) \\ &= \varphi_i' \times \exp\Big(\frac{1}{2}L_c\sum_{j=0}^{3}\left[r_i^{(j)} \times x_i^{(j)}\right]\Big). \end{aligned} \tag{1.54}$$

This equation involves another constant number $\boldsymbol{\varphi}_i'$. By replacing the probabilities in (1.30) and removing the constant part, we have the branch metric for this constituent code:

$$\gamma(S_i, S_{i+1}) = L_a(u_i = \mathcal{U}(\ell)) + \frac{1}{2} L_c \sum_{j=0}^{3} \left[r_i^{(j)} \times x_i^{(j)} \right]. \tag{1.55}$$

1.2.4 Iterative Flow for IEEE 802.16m Turbo Code

When we consider the iterative decoding flow on the basis of this constituent code, the derivation of useful soft values should be revised for the compatibility with the LLR definition in (1.50). The first step is factorizing $\Pr(u_i; \boldsymbol{r})$ in the same way as (1.42). The equivalence $\Pr\left(u_i^{(0)}, u_i^{(1)}\right) = \Pr\left(x_i^{(0)}, x_i^{(1)}\right)$ let this joint probability be factorized to

$$\begin{aligned} & \Pr(u_i; r_0, \ldots, r_{\mathbf{N}-1}) \\ = \; & \Pr\left(x_i^{(0)}, x_i^{(1)}\right) \times \Pr\left(r_i^{(0)}, r_i^{(1)} \mid x_i^{(0)}, x_i^{(1)}\right) \\ & \times \sum_{\boldsymbol{u}: u_i} \left[\prod_{\substack{j=0 \\ j \neq i}}^{\mathbf{N}-1} \Pr\left(x_j^{(0)}, x_j^{(1)}\right) \Pr\left(r_j^{(0)}, r_j^{(1)} \mid x_j^{(0)}, x_j^{(1)}\right) \prod_{l=0}^{\mathbf{N}-1} \Pr\left(r_l^{(2)}, r_l^{(3)} \mid x_l^{(2)}, x_l^{(3)}\right) \right]. \end{aligned}$$

Therefore, the LLR in (1.50) can be rewritten as

$$L^{[\ell]}(u_i) = \ln \frac{\Pr(u_i = \mathcal{U}(\ell))}{\Pr(u_i = \mathcal{U}(0))} + \ln \frac{\Pr\left(r_i^{(0)}, r_i^{(1)} \mid u_i = \mathcal{U}(\ell)\right)}{\Pr\left(r_i^{(0)}, r_i^{(1)} \mid u_i = \mathcal{U}(0)\right)} + L_e^{[\ell]}(u_i), \tag{1.56}$$

where the last term $L_e^{[\ell]}(u_i)$ is

$$\ln \frac{\displaystyle\sum_{\boldsymbol{u}: u_i} \left[\prod_{\substack{j=0 \\ j \neq i}}^{\mathbf{N}-1} \Pr\left(u_j = \mathcal{U}(\ell)\right) \Pr\left(r_j^{(0)}, r_j^{(1)} \mid u_j = \mathcal{U}(\ell)\right) \prod_{l=0}^{\mathbf{N}-1} \Pr\left(r_l^{(2)}, r_l^{(3)} \mid x_l^{(2)}, x_l^{(3)}\right) \right]}{\displaystyle\sum_{\boldsymbol{u}: u_i} \left[\prod_{\substack{j=0 \\ j \neq i}}^{\mathbf{N}-1} \Pr\left(u_j = \mathcal{U}(0)\right) \Pr\left(r_j^{(0)}, r_j^{(1)} \mid u_j = \mathcal{U}(0)\right) \prod_{l=0}^{\mathbf{N}-1} \Pr\left(r_l^{(2)}, r_l^{(3)} \mid x_l^{(2)}, x_l^{(3)}\right) \right]}.$$

The second term in (1.56) can be simplified into (1.57) by the following properties: $\mathcal{U}(\ell) = (\ell^{(0)}, \ell^{(1)})$, $(u^{(0)}, u^{(1)}) = (c^{(0)}, c^{(1)})$, (1.4), and (1.8).

$$\ln \frac{\Pr\left((r_i^{(0)}, r_i^{(1)}) \mid (u_i^{(0)}, u_i^{(1)}) = \mathcal{U}(\ell)\right)}{\Pr\left((r_i^{(0)}, r_i^{(1)}) \mid (u_i^{(0)}, u_i^{(1)}) = \mathcal{U}(0)\right)} = -L_c \times [r_i^{(0)} \times \ell^{(0)} + r_i^{(1)} \times \ell^{(1)}] \tag{1.57}$$

Moreover, we can use $L_a^{[\ell]}(u_i)$ in (1.52) to take the place of the first term. These substitutions lead to a proper LLR representation for the double binary scheme as (1.58) with $\ell = 0 \sim 3$.

$$L^{[\ell]}(u_i) = L_a^{[\ell]}(u_i) - L_c \times [r_i^{(0)} \times \ell^{(0)} + r_i^{(1)} \times \ell^{(1)}] + L_e^{[\ell]}(u_i). \tag{1.58}$$

Notice that the definition of $L^{[\ell]}(u_i)$ in (1.50) will have identical numerator and denominator when ℓ is 0. It implies $L^{[\mathbf{0}]}(u_i) = 0$ for $i = 0 \sim (\mathbf{N} - 1)$. For the same reason, $L_a^{[\mathbf{0}]}(u_i)$ and $L_e^{[\mathbf{0}]}(u_i)$ always keep zero. The calculations of these three likelihood ratios can be skipped.

The message propagation between the two constituent codes are also affected by the double binary code structure. Initially, we set the *a priori* value $L_{a1}^{[\ell]}(u_i)$ for the first constituent code to zero and utilize the MAP algorithm to calculate the *a posteriori* information $L_1^{[\ell]}(u_i) = 0$ for all i's and ℓ's. From (1.58), the extrinsic information $L_{e1}^{[\ell]}(u_i)$ will be

$$L_{e1}^{[\ell]}(u_i) = L_1^{[\ell]}(u_i) - L_{a1}^{[\ell]}(u_i) + L_c \times [r_i^{(0)} \times \ell^{(0)} + r_i^{(1)} \times \ell^{(1)}]. \tag{1.59}$$

The $L_{e1}^{[\ell]}(u_i)$ is regarded as the *a priori* value of the second constituent code. Due to the periodical intra-symbol permutation in (1.2), we must exchange the results about $\mathcal{U}(\ell) = (\ell^{(1)}, \ell^{(0)})$ for those about $\mathcal{U}(\ell) = (\ell^{(0)}, \ell^{(1)})$ as long as the symbol index satisfy the condition. If the $\tilde{u}_i$ has an even index, its *a priori* value estimation is updated according to (1.60); otherwise, the $L_{a2}^{[\ell]}(\tilde{u}_i)$ comes directly from $L_{e1}^{[\ell]}(\tilde{u}_i)$ for each ℓ.

$$\begin{cases} \ell^{(0)} \neq \ell^{(1)}: & L_{a2}^{[\mathbf{1}]}(\tilde{u}_i) \leftarrow L_{e1}^{[\mathbf{2}]}(u_{\mathcal{A}(i)}), \quad L_{a2}^{[\mathbf{2}]}(\tilde{u}_i) \leftarrow L_{e1}^{[\mathbf{1}]}(u_{\mathcal{A}(i)}) \\ \ell^{(0)} = \ell^{(1)}: & L_{a2}^{[\mathbf{0}]}(\tilde{u}_i) \leftarrow L_{e1}^{[\mathbf{0}]}(u_{\mathcal{A}(i)}), \quad L_{a2}^{[\mathbf{3}]}(\tilde{u}_i) \leftarrow L_{e1}^{[\mathbf{3}]}(u_{\mathcal{A}(i)}) \end{cases} \tag{1.60}$$

When the second SISO decoder collects all requisite inputs, it will get the LLR $L_2^{[\ell]}(\tilde{u}_i)$ by the MAP algorithm and derive the extrinsic information $L_{e2}^{[\ell]}(\tilde{u}_i)$ by

$$L_{e2}^{[\ell]}(\tilde{u}_i) = L_2^{[\ell]}(\tilde{u}_i) - L_{a2}^{[\ell]}(\tilde{u}_i) + L_c \times [\tilde{r}_i^{(0)} \times \ell^{(0)} + \tilde{r}_i^{(1)} \times \ell^{(1)}]. \tag{1.61}$$

This result will be the *a priori* information $L_{a1}^{[\ell]}(u_{\mathcal{A}(i)})$ of the other constituent code. Those cases relating to the $\mathcal{U}(\ell)$ with $\ell^{(0)} \neq \ell^{(1)}$ for an even-indexed $\tilde{u}_i$ must be swapped, too. After all *a priori* values are ready, a new half-iteration for the first constituent code can start. The turbo decoding runs iteratively until the iteration number reaches $\mathcal{I}$, and then the hard decisions that are determined by (1.51) are outputted.

Figure 1.10 shows the BER performance of IEEE 802.16m turbo code with $\mathbf{N} = \{24, 256, 2400\}$ at various iterations. Compared to Fig. 1.9, the results of the examples with similar number of information bits are alike. In any example, the curve of $\mathcal{I} = 16$ can approximately be viewed as the ultimate performance bound. Actually, when $\mathcal{I}$ exceeds a certain threshold, the improvement caused by further iterations will be limited to 0.2 dB. Due to the insignificant difference in

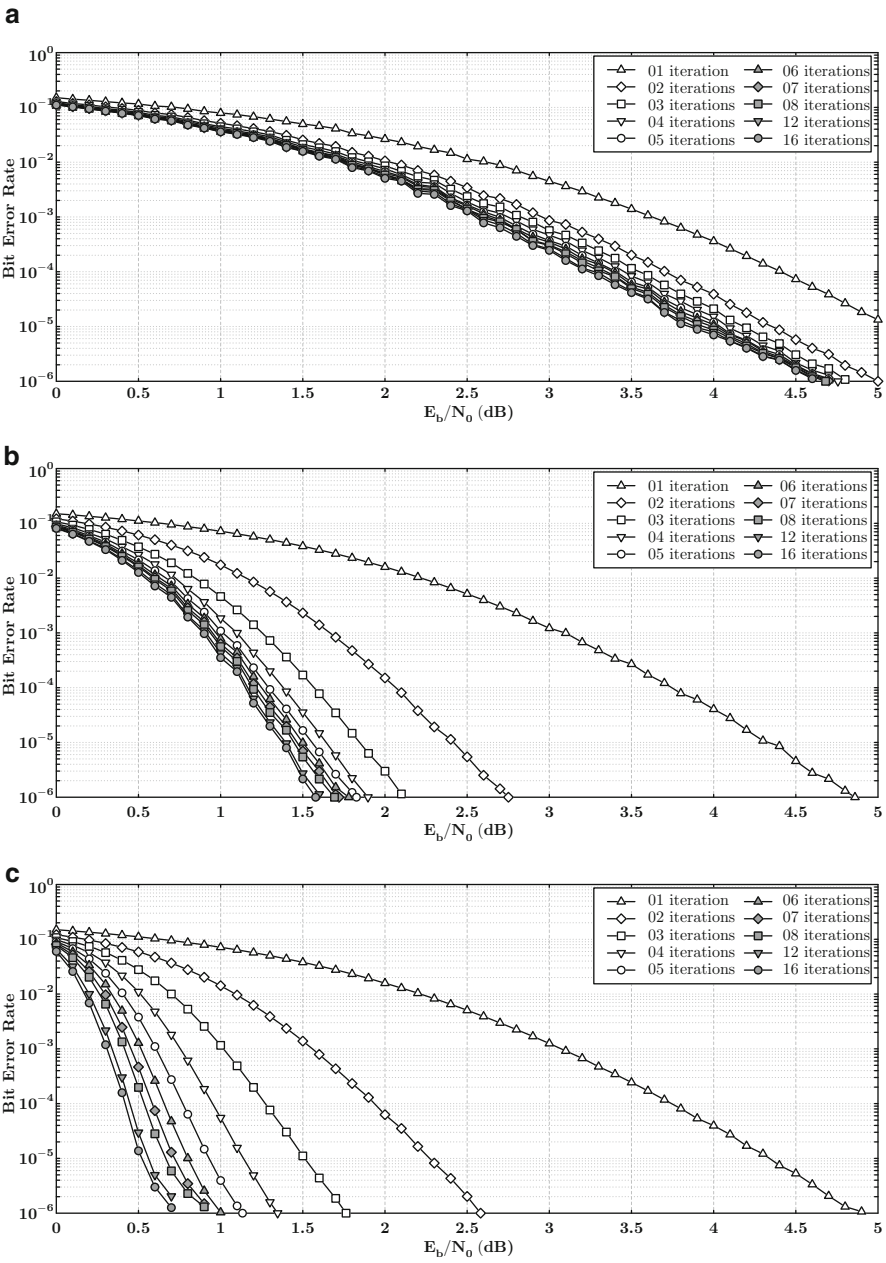

Fig. 1.10 Performance of IEEE 802.16m turbo code with various block sizes and iteration numbers. (**a**) IEEE 802.16m turbo code: $\mathbf{N} = 24$ and $\mathcal{I} = \{1, 2, 3, 4, 5, 6, 7, 8, 12, 16\}$. (**b**) IEEE 802.16m turbo code: $\mathbf{N} = 256$ and $\mathcal{I} = \{1, 2, 3, 4, 5, 6, 7, 8, 12, 16\}$. (**c**) IEEE 802.16m turbo code: $\mathbf{N} = 2400$ and $\mathcal{I} = \{1, 2, 3, 4, 5, 6, 7, 8, 12, 16\}$

performance and considerable saving in latency, we prefer that threshold to 16 while determining the necessary iteration number. According to the simulation results, the common value of $\mathcal{I}$ can be set to 4 for $\mathbf{N} = 24$, 5 for $\mathbf{N} = 256$, and 8 for $\mathbf{N} = 2400$. It is easy to infer that the suitable $\mathcal{I}$ for any other $\mathbf{N}$ in this standard is in the range of four to eight, and we will adopt these choices in later discussions.

1.3 Techniques for Efficient Decoding Process

1.3.1 Simplified MAP Algorithms

Although the turbo code can achieve the impressive performance, the complex algorithm will be an obstacle to implement the decoders. The main operations of the MAP algorithm are exponentiation and logarithm. Both of them take more hardware resources than most basic arithmetic operations. To reduce the complexity, the works in [23–25] proposed simpler algorithms that chiefly use addition and subtraction to calculate the metrics and LLRs. These alternative methods are developed with the following Jacobian function

$$\ln(e^{y_1} + e^{y_2}) \triangleq \max^*(y_1, y_2) = \max(y_1, y_2) + \ln(1 + e^{-|y_1 - y_2|}), \tag{1.62}$$

and its extension form

$$\ln(e^{y_1} + e^{y_2} + e^{y_3}) = \max^*(y_1, y_2, y_3) = \max^*(\max^*(y_1, y_2), y_3). \tag{1.63}$$

Note that the logarithmic function is equivalent to the summation of the normal $\max(\cdot)$ function and a correction term $\ln(1 + e^{-|y_1 - y_2|})$. The normal $\max(\cdot)$ function is just the comparison and selection among all inputs. In addition, the $\ln(1 + e^{-|y_1 - y_2|})$, whose value depends on $|y_1 - y_2|$, can be calculated beforehand and stored in a lookup table. Thus, the $\max^*(\cdot)$ function can be realized easily. After replacing the original computations in (1.23), (1.25), and (1.28) with the $\max^*(\cdot)$ functions, we can write the forward metric, backward metric, and LLR as (1.64)–(1.66) respectively.

$$\alpha(S_i) = \max_{S_{i-1}}^* \left[\gamma(S_{i-1}, S_i) + \alpha(S_{i-1})\right] \tag{1.64}$$

$$\beta(S_i) = \max_{S_{i+1}}^* \left[\gamma(S_i, S_{i+1}) + \beta(S_{i+1})\right] \tag{1.65}$$

$$\begin{aligned} L(u_i) = & \max_{(S_i, S_{i+1}):u_i=0}^* [\alpha(S_i) + \gamma(S_i, S_{i+1}) + \beta(S_{i+1})] \\ & - \max_{(S_i, S_{i+1}):u_i=1}^* [\alpha(S_i) + \gamma(S_i, S_{i+1}) + \beta(S_{i+1})]. \end{aligned} \tag{1.66}$$

The decoding algorithm based on the above three equations is named the Log-MAP algorithm. For the Log-MAP algorithm, a lookup table with high accuracy can lead to the same performance as the MAP algorithm, but it also means the requirement of larger memory.

A further simplification can be achieved by completely discarding the lookup table. The correction term will be omitted from the $\max^*(\cdot)$ function:

$$\max^*(y_1, y_2) \approx \max(y_1, y_2). \tag{1.67}$$

This approximation leads to the Max-Log-MAP algorithm that involves only the additions and the $\max(\cdot)$ functions. However, the lack of the correction term would make the LLR calculation too optimistic and then cause performance degradation. From [26, 27], the problem can be solve by scaling the extrinsic information. While the SISO decoder get the extrinsic information through the Max-Log-MAP algorithm, it will multiply $L_e(u_i)$ by a scaling factor ζ. Afterward, the scaled $L'_e(u_i)$ in (1.68) is send to the other SISO decoder as more reliable *a priori* value.

$$L'_e(u_i) = \zeta \times L_e(u_i) \tag{1.68}$$

The effect of ζ depends on the code structure and channel condition. With the aid of simulation, we can determine if the scaling factor is beneficial. Figure 1.11 shows the variations in performance after using the simplified MAP algorithms. For these 3GPP LTE-Advanced turbo codes, choosing $\zeta = 0.75$ achieves better improvement than the other cases with respect to Max-Log-MAP algorithms; but $\zeta = 0.5$ would be an unwise choice for its inferior performance as $\mathbf{E_b/N_0}$ exceeds 0.8 dB. The results of IEEE 802.16m turbo codes indicates that both $\zeta = 0.875$ and $\zeta = 0.75$ successfully compensate the information loss in (1.67), while $\zeta = 0.625$ and $\zeta = 0.5$ worsen the performance. In our applications, the Max-Log-MAP algorithm with $\zeta = 0.75$ would be an effective solution that provides a compromise between complexity and performance.

1.3.2 Sliding Window Technique

The processing schedule also has a great influence on the overhead of a practical SISO decoder. Here we consider a general case of a size-$\mathbf{N}$ data block and assume that initial state S_0 and the last state $S_\mathbf{N}$ of the corresponding trellis path are known. The inputs includes received data and *a priori* values, and they are usually sent to the SISO decoder in ascending order. With the initial condition $\alpha(S_0)$, the forward metric can be derived immediately after the reception of r_i and $L_a(u_i)$; but all backward metrics, which are initialized with $\beta(S_\mathbf{N})$, cannot be computed until the last input data ($r_{\mathbf{N}-1}$ and $L_a(u_{\mathbf{N}-1})$) are ready. According to the decoding algorithm, the calculation of $L(u_i)$ involves $\alpha(S_i)$ and $\beta(S_{i+1})$, so it needs to wait the completion of the two types of metrics. The dependencies force the SISO decoder

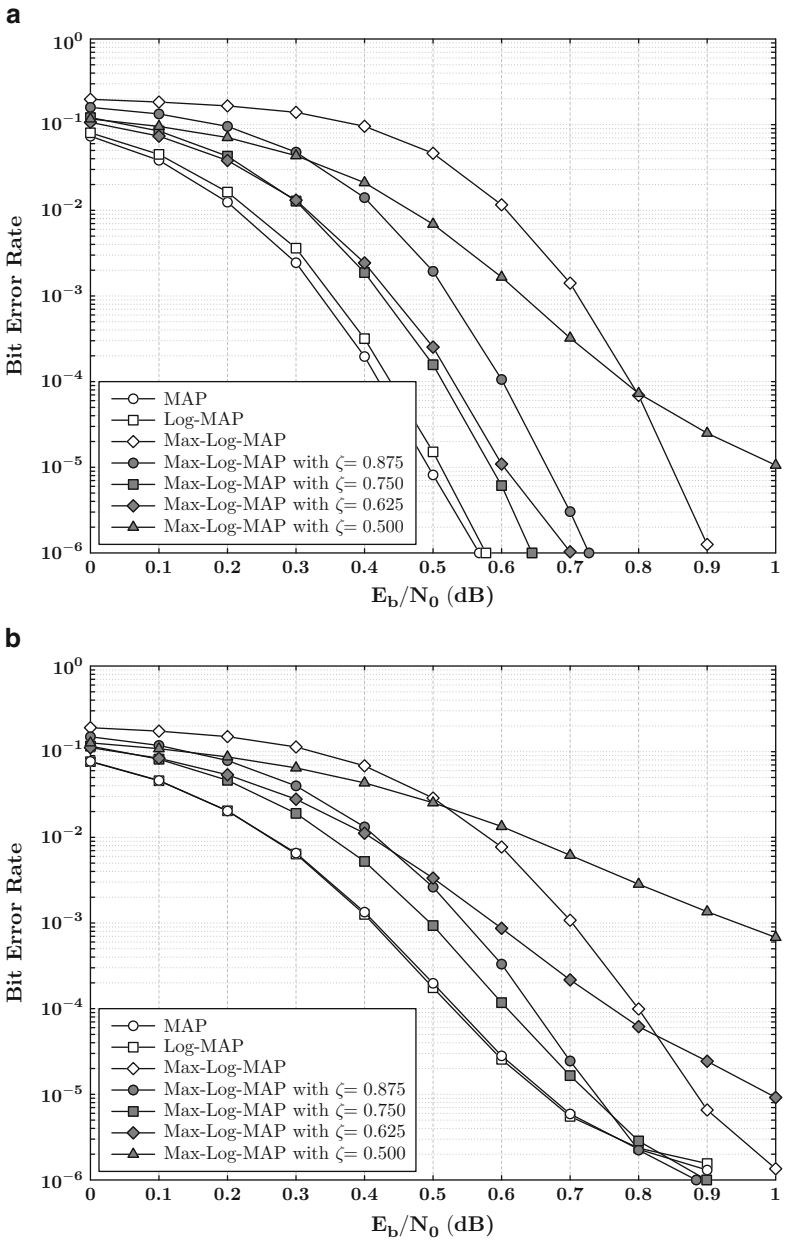

Fig. 1.11 Performance of turbo codes with various decoding algorithms. (**a**) 3GPP LTE-Advanced turbo code: $\mathbf{N} = 6144$ and $\mathcal{I} = 8$. (**b**) IEEE 802.16m turbo code: $\mathbf{N} = 2400$ and $\mathcal{I} = 8$

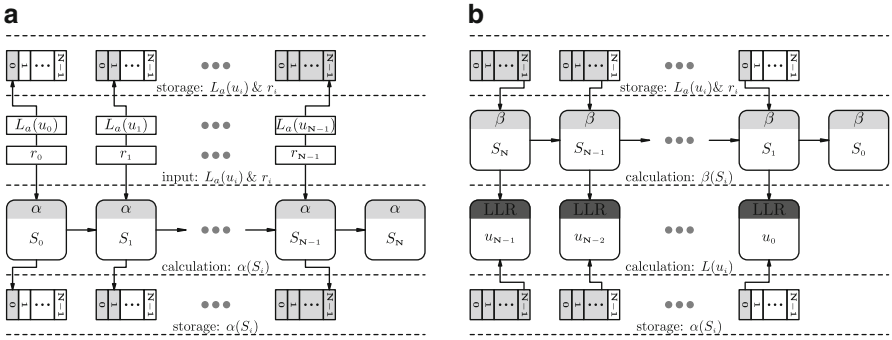

Fig. 1.12 Basic processing schedule of the SISO decoder. (**a**) Data access and execution of $\alpha(S_i)$. (**b**) Execution of $\beta(S_i)$ and $L(u_i)$

keeping the input data and forward metrics in memories before it starts computing the backward metrics and LLRs. Figure 1.12 demonstrates the conventional order of main executions. The execution of $L(u_i)$ follows the generations of $\beta(S_{i+1})$; and all LLRs are outputted in descending order. The period between the first input and the first output is directly proportional to the block size, and so is the requirement of extra memories. As $\mathbf{N}$ is large, the SISO decoder will have long processing latency and use many storage elements; it would be impractical to implement such a design.

The sliding window technique introduced in [28, 29] can avoid considerable overhead of the SISO decoder with large $\mathbf{N}$. It exploits a dummy backward metric calculation, symbolized as $\beta_d(S_i)$ for the states at i-th trellis stage, to establish the initial conditions for the true backward metric. With this technique, the data block is divided into $\lceil \mathbf{N}/\mathbf{L} \rceil$ windows. Here we let $\mathbf{W}_k$ stand for the k-th window, and the length of $\mathbf{W}_k$ obeys

$$\begin{cases} \mathbf{L} & \text{if } k \in \{0, \cdots, \lceil \frac{\mathbf{N}}{\mathbf{L}} \rceil - 2\}; \\ \mathbf{N} - \mathbf{L} \times (\lceil \frac{\mathbf{N}}{\mathbf{L}} \rceil - 1) & \text{if } k = \lceil \frac{\mathbf{N}}{\mathbf{L}} \rceil - 1. \end{cases} \tag{1.69}$$

Except for the last window, the SISO decoder performs the dummy backward metric calculation over each $\mathbf{W}_k$ with the initial value $\beta_d(S_{(k+1)\mathbf{L}} = \mathrm{S}^{(m)}) = 0$ for $m = 0 \sim 7$ when all necessary inputs are ready; and it can get $\beta_d(S_{k\mathbf{L}})$ after $\mathbf{L}$ recursion steps. This result will be assigned to the true backward metric at the $(k\mathbf{L})$-th trellis stage: $\beta(S_{k\mathbf{L}}) = \beta_d(S_{k\mathbf{L}})$; then there is an initialization for calculating $\beta(S_i)$ for $i \leq k\mathbf{L}$. Notice that the process of calculating the initial backward metric of the last window is exactly like the conventional way, but the $\beta(S_{\mathbf{N}})$ in a circular trellis structure will be derived from the dummy backward calculation of $\mathbf{W}_0$. The executions can be scheduled as Fig. 1.13, where the data access and forward metric calculation remain unchanged. The SISO decoder is allowed to calculate the $\beta_d(S_i)$'s of $\mathbf{W}_k$ whenever it collects the data of $\mathbf{W}_k$. At the same time, the executions of $\beta(S_i)$ and $L(u_i)$

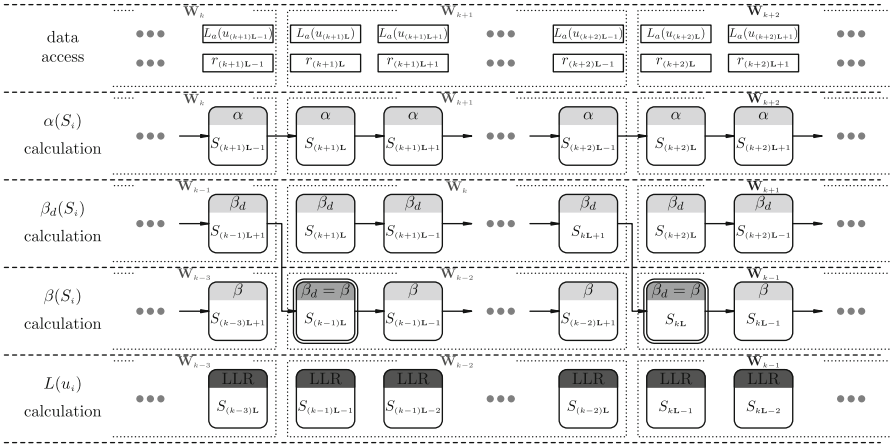

Fig. 1.13 Decoding process with the sliding window technique

relating to $\mathbf{W}_{k-2}$ can start with $\beta(S_{(k-1)\mathbf{L}}) = \beta_d(S_{(k-1)\mathbf{L}})$. Now the processing latency and storage requirement depend on window length $\mathbf{L}$ rather than block size $\mathbf{N}$. As a consequence, the LLRs can be generated earlier than those in the conventional process; besides, the memory usage for r_i, $L_a(u_i)$, and $\alpha(S_i)$ will decrease.

The overhead and performance must be considered together when the sliding window technique is employed. Although this method with short $\mathbf{L}$ can favor the cost reduction of one SISO decoder, the dummy backward metric calculation might fail to provide robust initialization for true backward metrics; that is, too short $\mathbf{L}$ might cause performance loss. The suitable $\mathbf{L}$ will vary with the turbo code specifications such as $\mathbf{N}$ and $\mathbf{R}$. Figure 1.14 presents the performance of turbo codes with different combinations of $\mathbf{N}$ and $\mathbf{L}$ in each standard. For those 3GPP LTE-Advanced turbo codes, choosing $\mathbf{L} = 8$ degrades the performance significantly, especially for $\mathbf{N} = 6144$ and $\mathbf{N} = 512$; whereas choosing $\mathbf{L} > 16$ can make the performance very close to the ideal results. For IEEE 802.16m application, the schemes with $\mathbf{L} \geq 16$ work as successfully as the conventional cases ($\mathbf{L} = \mathbf{N}$) for both $\mathbf{N} = 2400$ and $\mathbf{N} = 256$; and the choices of $\mathbf{L} \geq 8$ are preferable for $\mathbf{N} = 24$. As a result, the practical turbo decoders should support the maximum $\mathbf{L}$ of 32 and of 16 for all $\mathbf{N}$'s in 3GPP LTE-Advanced and IEEE 802.16m standards respectively so that it can approximate the optimal BER at reasonable cost.

1.3.3 Early Stopping Criteria

The iteration number is another dominant factor in the overall decoding time. In the previous discussion, the value of $\mathcal{I}$ is typically determined based on the worst case. Actually, from Figs. 1.9 and 1.10, the decoder can correct some received

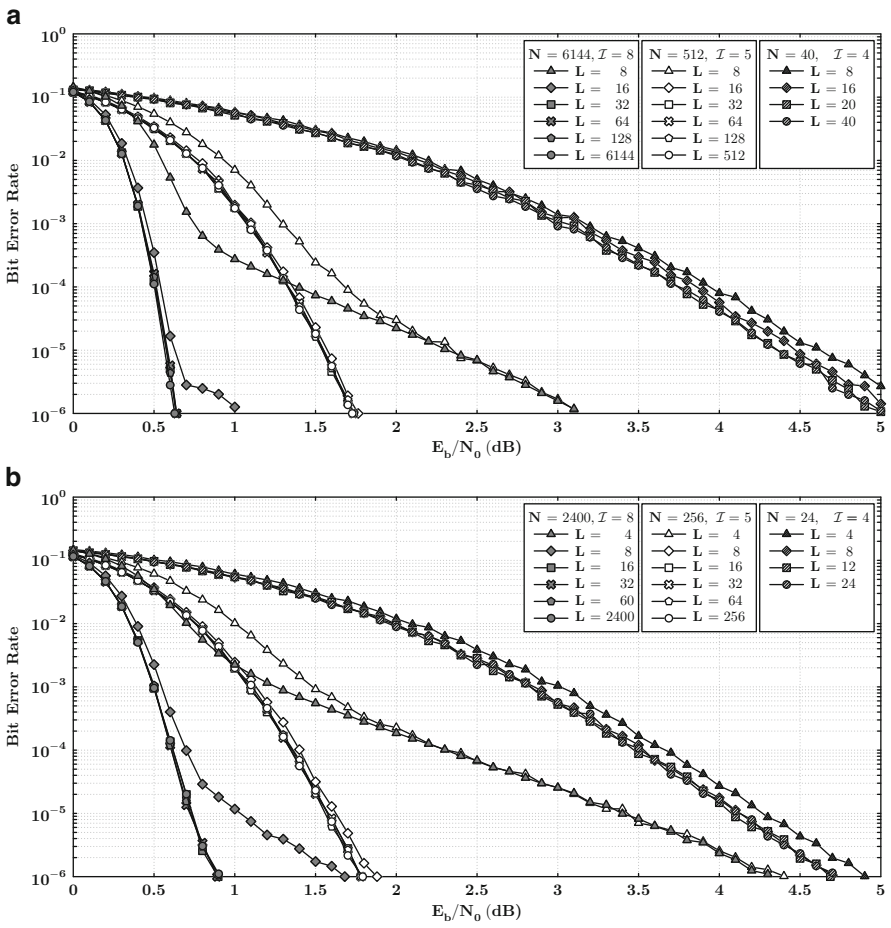

Fig. 1.14 Performance of turbo codes with various block sizes and window lengths. (**a**) 3GPP LTE-Advanced turbo code: $\zeta = 0.75$, and $\mathbf{N} = \{40, 512, 6144\}$. (**b**) IEEE 802.16m turbo code: $\zeta = 0.75$, and $\mathbf{N} = \{24, 256, 2400\}$

data blocks before the maximum iteration. It is unnecessary to spend the same $\mathcal{I}$ iterations in decoding every received data block [22]. During the turbo decoding of most erroneous data blocks, the corresponding soft values will converge after certain iterations; so the performance gain in the subsequent process is very insignificant. If the data suffer from slighter noise, there is a high possibility that the iteration bound about the convergence is smaller than preset $\mathcal{I}$. The process from the lower bound to $\mathcal{I}$ is viewed as a waste of computation and time. To avoid too much redundant process, we will terminate the decoding flow dynamically by early stopping criteria. Most criteria are the statistics for difference between the temporary results at different half-iterations. Among all criteria, the hard-decision-aided (HDA) criterion

in [30] and the sign-difference-ratio (SDR) criterion in [31] are two of the simplest schemes. The HDA criterion is the consistency check between the hard decisions of two consecutive half-iterations; and the SDR criterion is the examination of the sign changes of the extrinsic information between two consecutive half-iterations. Generally, the HDA criterion can result in more reduction of iteration, while the SDR criterion can achieve better BER performance. In either criterion, there will be a threshold indicating the maximum allowable difference. If the number of sign changes is less than the threshold, the decoding process for the current data block will finish; otherwise, it will run for at most $\mathcal{I}$ iterations. A common choice of the threshold is around $0.01\mathbf{N} \sim 0.001\mathbf{N}$ for the SDR criterion and is 0 for the HDA criterion. Loose constraints can lead to more savings in iteration, but undesirable performance degradation may occur.

Figure 1.15 presents the frame error rate (FER) performance and average iterations of turbo codes using different stopping criteria. Here we denote the threshold by ξ. These results about iterations are obtained based on at least 10000 data blocks, except for the trivial case with fixed iterations. For 3GPP LTE-Advanced turbo codes, the criteria with $\xi = 0$ make the decoder take fewer iterations to achieve nearly the same performance; and the saving by the HDA criterion is superior to that by the SDR criterion. The other ξ's are improper threshold values for this application. Although $\xi = 8$ is a relatively small for the SDR criterion, it still causes poor judgment about when to terminate the decoding process, particularly for $\mathbf{E_b}/\mathbf{N_0} \geq 0.7$ dB. The outcomes of IEEE 802.16m turbo codes are analogous with those of the preceding example. This turbo decoder can also get advantage from the criteria with $\xi = 0$. Nevertheless, now it is safe to use the SDR criterion with a small but nonzero threshold ($\xi = 6$) for $\mathbf{E_b}/\mathbf{N_0}$ between 0 and 1 dB. This distinction between these two examples shows that the effectiveness of the stopping rules varies greatly according to applications. We should choose the appropriate criterion in an adaptive way to shorten the decoding time and avoid performance loss.

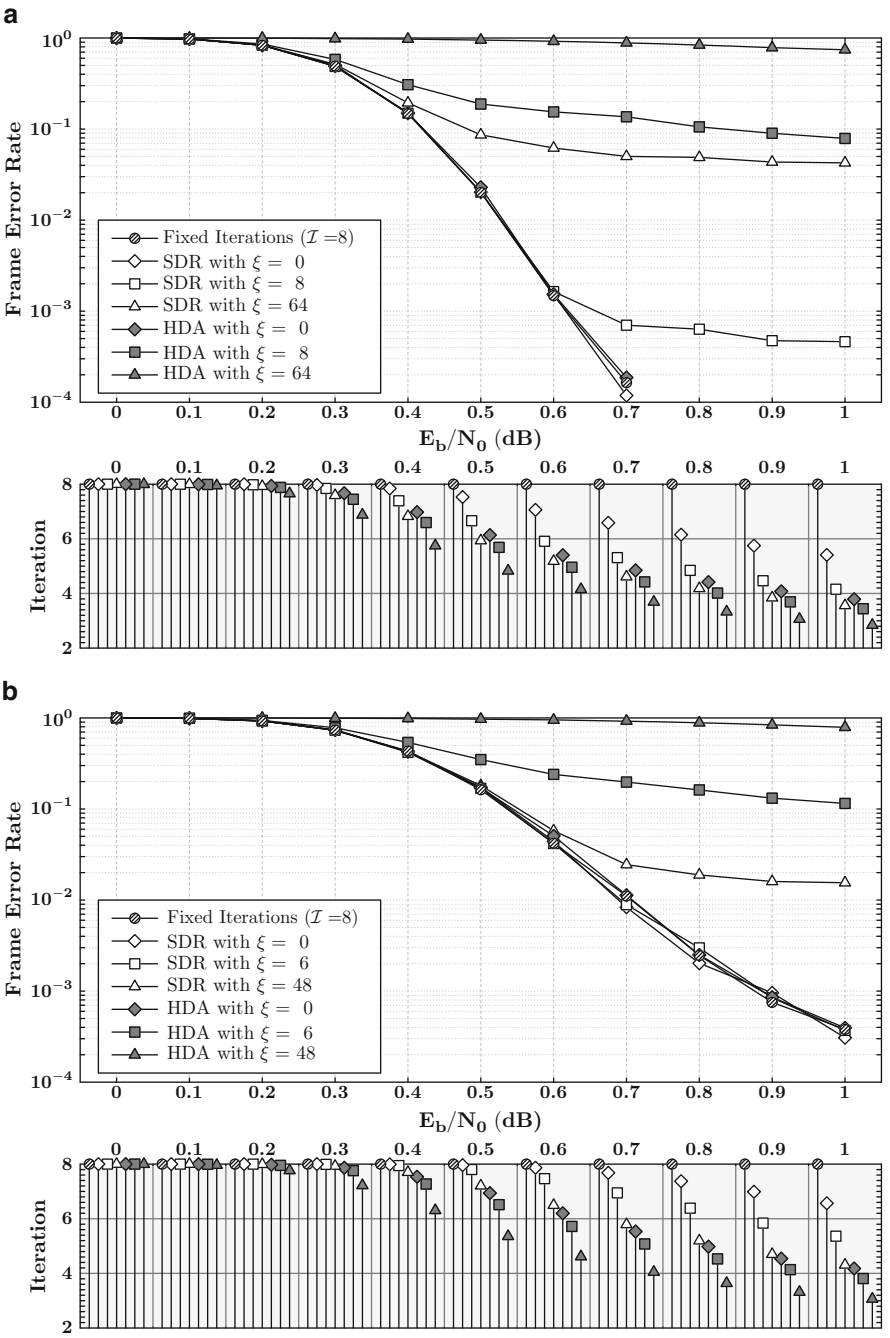

Fig. 1.15 Performance and iterations of turbo codes with various stopping criteria. (**a**) 3GPP LTE-Advanced turbo code: $\mathbf{N} = 6144$, $\mathbf{L} = 32$, and $\zeta = 0.75$. (**b**) IEEE 802.16m turbo code: $\mathbf{N} = 2400$, $\mathbf{L} = 16$, and $\zeta = 0.75$

Chapter 2
Conventional Architecture of Turbo Decoder

When digital circuits and storage elements are exploited to perform the theoretical algorithm, we should pay attention to how the cycle-based process and resource limitation affect the turbo decoding. During the implementation, the hardware cost is of top priority. Our targets are usually fast processing speed and great decoding performance at the least expense of hardware. Of course those techniques toward lower complexity in the previous chapter will be applied. The practical design contains several functional units for the major computations in the simplified algorithm; besides, it needs a few control units to handle the data flow for the sliding window method and the message propagation between two constituent codes. These internal devices will run a variety of arithmetic operations. Among all the operations, the multiplication and division should be avoided because their corresponding circuits take much longer execution time and much larger hardware resource. In this chapter, we introduce the major component circuits of a practical turbo decoder at first. Then we shift the focus to the conventional architecture of the SISO decoder as well as its processing schedule. Based on such a prototype, the design issues about replacing the floating-point data with fixed-point data are highlighted. It is possible that the loss of precision gives rise to performance degradation. On the other hand, the bit number for data expression, also called data width, dominates the hardware cost. Thus, the choice of adequate data width is dependent on both the required performance and acceptable overhead. The subsequent discussions address the potential overflow problem and show the effects of several solutions. At the end of the chapter, two modified SISO decoders that have advantages such as shorter latency or less hardware over the conventional one are presented.

C.-C. Wong and H.-C. Chang, *Turbo Decoder Architecture for Beyond-4G Applications*, DOI 10.1007/978-1-4614-8310-6_2,

2.1 Practical Turbo Decoder Architecture

The practical turbo decoder mainly consists of a few size-**N** memory modules and one SISO decoder. Although the conceptual structure in Fig. 1.8 uses two SISO decoders, either of them remains idle as the other one is in operation. The decoding process of each constituent code requires the *a priori* information from the other constituent code, and the randomness of the interleaver makes it hard to start the half-iteration for this constituent code until all necessary inputs are available. Moreover, the two constituent codes are identical in code structure; so they can share the same SISO decoder. The practical architecture is usually depicted as Fig. 2.1, where these memories will store the received data and temporary decoding results, and a global controller can handle the alternation between the constituent codes. The SISO decoder can process the codes derived from the original data sequence by setting the selection signals of these multiplexers to 0; whereas it can process the codes derived from the permuted data sequence by changing all selection signals to 1. Because of the regular alternation, the process throughout an half-iteration is called a decoding round, too. The interleavers and de-interleavers are implemented in the form of address generators that indicate which data in the memory should be outputted. Note that the special address generators are activated only during the decoding rounds for the permuted data sequence. The SISO decoder can get $L_{a2}(\tilde{u}_i)$ by reading the $\mathcal{Q}(i)$-th (or $\mathcal{A}(i)$-th) word from the extrinsic information memory, and afterwards it will write the corresponding output $L_{e2}(\tilde{u}_i)$ back to the

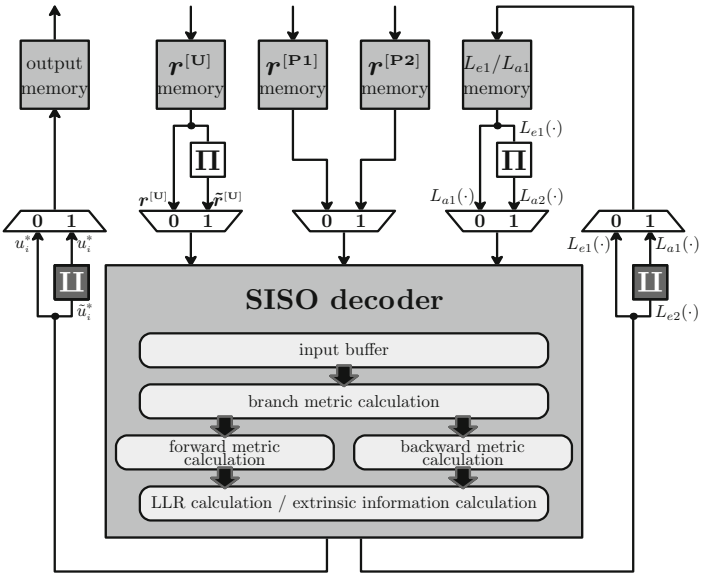

Fig. 2.1 Practical architecture of a turbo decoder

same location as the $\mathcal{Q}(i)$-th (or $\mathcal{A}(i)$-th) *a priori* value estimation for the next round. With such hardware arrangement, single memory module is sufficient for the temporary results of both constituent codes. The output memory is another storage device for the decoding results. It buffers the final decisions and adjusts their order so that the format of output sequence can meet the system requirement. Basically these memories and their controllers greatly influence the area and power of the design.

Compared to the memory modules, the SISO decoder plays a relatively major role in the processing speed. In fact, the speed bottleneck is caused by the arithmetic calculations inside the SISO decoder. When the MAP algorithm is implemented with flip-flops and logic gates, we spend one or more cycles on each kind of calculation. There are an enormous amount of logic paths in this circuit, and the one with the longest delay determines the operating frequency. Unless the clock period is larger than the critical path delay, there is a risk that some flip-flops may fetch unstable and incorrect values from logic gates. This means that shortening this path is one major implementation issue for faster decoding process. In addition to the clock cycle time, the total cycle number is also of great importance. The fact that the SISO decoder must collect $\alpha(S_i)$ and $\beta(S_{i+1})$ to calculate $L(u_i)$ makes every decoding round take some execution time to prepare these metrics before getting any LLR. Here we represent the cycle number per decoding round as $\mathbf{\Gamma_R}$ and assume that the period for outputting the results of all $\mathbf{N}$ symbols equals $\mathbf{\Gamma_N}$ cycles; then the *operating efficiency* of the SISO decoder can be defined as

$$\eta = \frac{\mathbf{\Gamma_N}}{\mathbf{\Gamma_R}}. \tag{2.1}$$

For conventional designs, the general value of $\mathbf{\Gamma_N}$ is $\mathbf{N}$. The relevant design issue is achieving higher η by the minimization of $\mathbf{\Gamma_R}$. It takes $(2 \times \mathcal{I} \times \mathbf{\Gamma_R})$ cycles to derive the decisions of $\mathbf{N}$ symbols. When the turbo decoder operates with the frequency of $\mathcal{F}$ Hz, and one symbol contains ϕ information bits ($\phi = 1$ for 3GPP LTE-Advanced standard and $\phi = 2$ for IEEE 802.16m standard), the corresponding throughput in bits per second will be

$$\Theta = \frac{\phi \times \mathbf{\Gamma_N}}{2 \times \mathcal{I} \times \mathbf{\Gamma_R} \times (1/\mathcal{F})} = \frac{\phi \times \eta \times \mathcal{F}}{2 \times \mathcal{I}}. \tag{2.2}$$

The pursuit of high throughput relies on the improvement of η or $\mathcal{F}$. It might lead to a more complicated data flow. The design challenge is to attain the objective with the least complexity.

2.1.1 Circuits of Address Generators

The most complex component within the controller for memory access is the address generator. It has to compute the interleaving indexes $\mathcal{Q}(i)$ or $\mathcal{A}(i)$ from i when we processing the constituent codes from permuted information blocks. From (1.1) and

(1.3), the direct use of either equation involves multiplication and modulo operation. The address generator should support all block sizes and all parameter sets in each application; that is, the operands for these complicated calculations are variables rather than constants. Their implementation not only costs numerous logic gates but requires many cycles to get the answer. Since the time for accessing memory is extended, the decoding process becomes inefficient, and the throughput decreases. To sidestep this problem, we must find other simpler approaches to calculating $\mathcal{Q}(i)$ and $\mathcal{A}(i)$.

In [32], an indirect way of generating these interleaving indexes is illustrated. It is free from the use of complicated calculations. The key to the simplification of QPP interleaving function is given in (2.3), where the $(i+\delta)$-th interleaving index is derived by the summation of $\mathcal{Q}(i)$ and an auxiliary function $\mathbf{Q'}(i;\delta)$ defined in (2.4).

$$\begin{aligned}\mathcal{Q}(i+\delta) &= f_1 \times (i+\delta) + f_2 \times (i+\delta)^2 && (\text{mod } \mathbf{N}) \\ &= f_1 \times i + f_2 \times i^2 + f_1 \times \delta + f_2 \times \delta^2 + 2f_2 \times i \times \delta && (\text{mod } \mathbf{N}) \\ &= \mathcal{Q}(i) + \mathbf{Q'}(i;\delta) && (\text{mod } \mathbf{N}) \quad (2.3)\end{aligned}$$

$$\mathbf{Q'}(i;\delta) = f_1 \times \delta + f_2 \times \delta^2 + 2f_2 \times i \times \delta \quad (\text{mod } \mathbf{N}) \quad (2.4)$$

Now we examine the correlation between $\mathbf{Q'}(i+\delta;\delta)$ and $\mathbf{Q'}(i;\delta)$ by (2.5) and discover that their difference is independent of i.

$$\begin{aligned}\mathbf{Q'}(i+\delta;\delta) &= f_1 \times \delta + f_2 \times \delta^2 + 2f_2 \times (i+\delta) \times \delta && (\text{mod } \mathbf{N}) \\ &= f_1 \times \delta + f_2 \times \delta^2 + 2f_2 \times i \times \delta + 2f_2 \times \delta^2 && (\text{mod } \mathbf{N}) \\ &= \mathbf{Q'}(i;\delta) + 2f_2 \times \delta^2 && (\text{mod } \mathbf{N}) \quad (2.5)\end{aligned}$$

Assuming that $\mathcal{Q}(i)$ and $\mathbf{Q'}(i;\delta)$ are already known, we can derive $\mathcal{Q}(i+\delta)$ and $\mathbf{Q'}(i+\delta;\delta)$ easily in short time. Then we can substitute $(i+\delta)$ for i in these two equations to find $\mathcal{Q}(i+2\delta)$. The subsequent interleaving indexes can be calculated by renewing the value of i and repeating such procedure. Both $\mathcal{Q}(0)$ and $\mathbf{Q'}(0;\delta)$ are necessary for starting the recursive calculation. The offset $2f_2\delta^2$ modulo $\mathbf{N}$ has to be determined in advance as well. For a fixed δ, these pre-defined terms are constants. After remodeling the interleaving functions, the address generator can accomplish its task with much simpler arithmetic operations.

This concept of recursive calculation is also applicable to the ARP interleaver [32]. At first, we define another auxiliary function $\mathbf{A'}(i)$ as (2.6) and rewrite (1.3) as (2.7).

$$\mathbf{A'}(i) = \varepsilon \times i \quad (\text{mod } \mathbf{N}) \quad (2.6)$$

$$\mathcal{A}(i) = \mathbf{A'}(i) + g_{i \bmod 4} \quad (\text{mod } \mathbf{N}) \quad (2.7)$$

The next step is to modify the expressions of $\mathbf{A'}(i+\delta)$ and $\mathcal{A}(i+\delta)$.

$$\begin{aligned} \mathbf{A'}(i+\delta) &= \varepsilon \times (i+\delta) && (\text{mod } \mathbf{N}) \\ &= \mathbf{A'}(i) + \varepsilon \times \delta && (\text{mod } \mathbf{N}) \quad (2.8) \\ \mathcal{A}(i+\delta) &= \mathbf{A'}(i+\delta) + g_{(i+\delta) \bmod 4} && (\text{mod } \mathbf{N}) \quad (2.9) \end{aligned}$$

These equations allow us to update $\mathcal{A}(i)$ and $\mathbf{A'}(i+\delta)$ from a given $\mathbf{A'}(i)$ at the same time; then the $\mathbf{A'}(i+\delta)$ can be further used for $\mathcal{A}(i+\delta)$ and $\mathbf{A'}(i+2\delta)$. For this recursion, the base case is $\mathbf{A'}(0) = 0$. The other essential parameters include g_0, g_1, g_2, g_3, and $\varepsilon\delta$ modulo $\mathbf{N}$. In spite of the varying offset values for updating $\mathcal{A}(i)$, the selection of the right one merely depends on the least significant 2 bits of i. It is still easy to perform these modified equations.

With the alternative address generation, the hardware consists primarily of adders and subtractors. The addition of each equation has two operands whose values are both less than $\mathbf{N}$, so the upper bound of their summation is $2\mathbf{N}$. For those summations greater than $\mathbf{N}$, the modulo operation is carried out easily by subtracting $\mathbf{N}$ from them. We can complete all calculations of every recursion step within one cycle. Consequently, the address generator is able to output the interleaving indexes on the fly.

2.1.2 Circuits of Main Functional Units

When the Max-Log-MAP algorithm is applied, the major computations of the SISO decoder include $\gamma(S_i, S_{i+1})$ in (1.35) or (1.55), $\alpha(S_i)$ in (1.64), $\beta(S_i)$ in (1.65), and LLR in (1.66) with the approximation in (1.67). Because the value of $x_i^{(j)}$ is ± 1, the circuits for branch metric calculations are common adders. There are 4^ϕ combinations of $\{x_i^{(0)}, \cdots, x_i^{(2\phi-1)}\}$ for a given i and either state transition in Fig. 1.4; we need to calculate at most 4^ϕ branch metrics for all branches between two successive trellis stages. The other three calculations, which all involve the $\max(\cdot)$ functions, will perform the additions followed by comparison and selection. Such a series of executions is the typical add-compare-select (ACS) operation [33]. Figure 2.2a demonstrates the basic hardware for forward metric calculation of a single state. It contains 2^ϕ 2-input adders, one 2^ϕ-input comparator, and one 2^ϕ-to-1 multiplexer. Note that the computation of $\alpha(S_i)$ can be regarded as the accumulation of branch metrics; the circuits for normalization are connected to the multiplexer for fear that the arithmetic overflow occurs. Finally, the flip-flop will capture the normalized results at the active clock edge, and its original content $\alpha(S_{i-1})$ will be overwritten by $\alpha(S_i)$. In either application, the SISO decoder requires eight such functional units for all states. By appropriate substitution of inputs and outputs, the same circuits can be exploited for the backward metric calculation. In Fig. 2.2b, the hardware for calculating the log-likelihood value of one information symbol is presented. Its components are eight 3-input adders, one 8-input comparator, and one 8-to-1 multiplexer. We need two such functional units to calculate the log-likelihood

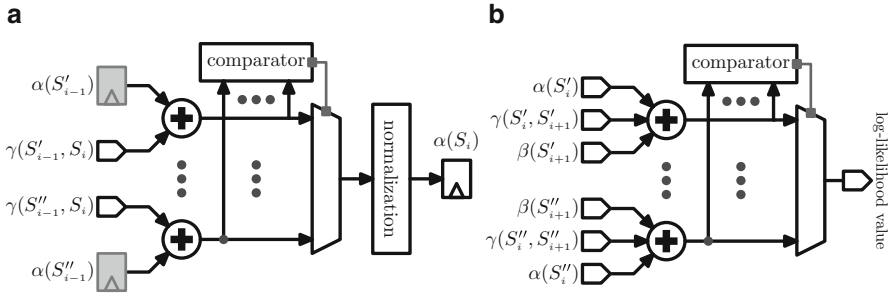

Fig. 2.2 Functional units for calculating forward metric and log-likelihood value. (**a**) Circuits for forward metric. (**b**) Circuits for log-likelihood value

values about $u_i = 0$ and $u_i = 1$ individually for 3GPP LTE-Advanced turbo codes, then we send their results to a subtractor to get $L(u_i)$. For the turbo decoder that supports IEEE 802.16m standard, there are four such functional units for all possible cases of $\{u_i^{(0)}, u_i^{(1)}\}$. Since $L^{[0]}(u_i)$ is always zero, it uses three extra subtractors to find $L^{[1]}(u_i)$, $L^{[2]}(u_i)$, and $L^{[3]}(u_i)$. After computing the LLR, the design takes one more step to produce extrinsic information and make decisions with a smaller number of adders and subtractors. In conclusion, the practical SISO decoder with the simplified algorithm is composed of basic arithmetic and logic circuits.

The ACS operation is the most time-consuming computation of the decoder. For this reason, the critical path of the whole design is generally located in either function unit in Fig. 2.2. The traditional method of shortening the path delay is the pipelining technique. However, it is effective only for the circuits for log-likelihood value. While we insert additional registers inside the circuits for forward or backward metrics, the other functional units will be stalled for some cycles on account of the data dependency of recursive metric calculation. This drawback may lead to inferior throughput. Therefore, most research works develop different methods to improve this data path. For example, the position of normalization circuit is changed in [34], and the double state technique is applied to the design in [35]. The fundamental idea is the modification to the circuit structure. In most cases, the improvement of path delay is accompanied by increases in hardware overhead. If there is a great demand for high operating frequency, we need to put more effort into the minimization of circuit area.

Apart from the hardware for arithmetic calculations, the SISO decoder also utilizes two types of buffers to facilitate its work flow. According to Figs. 1.12 and 1.13, different functional units will need r_i and $L_a(u_i)$ at different time. Hence, the data received from the memories should be stored until the related calculations are completed. Furthermore, these figures imply that $\alpha(S_i)$ is generated earlier than $\beta(S_{i+1})$, so those forward metrics must be retained temporarily for computing the LLR of i-th information symbol. The buffer size relies on the period between when the data are available and when they can be released. We have to make it as short as possible for the less cost of these buffers. As a matter of fact, the operating

efficiency can be enhanced by the reduction of this duration, too. The arrangement of the processing schedule with sliding window technique will be the most essential design issue here.

2.2 Design of Conventional SISO Decoders

2.2.1 Decoder Architecture and Processing Schedule

The SISO decoder architecture correlates strongly with its processing schedule. Figure 2.3 presents the conventional SISO decoder [36]. For simplicity, all the functional units for forward metrics are called α-ACS unit; similarly, β_d-ACS unit and β-ACS unit are the circuits for all dummy backward metrics and all backward metrics respectively; the circuits for LLRs, extrinsic probabilities, and decisions are also grouped together and named as LLR unit. This architecture uses three separate input buffers, each of which can store the data of one window. From the decoding flow in Fig. 2.3b, the data are inputted to the SISO decoder in ascending order. After any of the input buffers is filled with all $\mathbf{L}$ sets of $L_a(u_i)$ and r_i within the k-th window $\mathbf{W}_k$, it will send them to the β_d-ACS, α-ACS, and β-ACS as the processing schedule in order to get the metrics that correspond to $\mathbf{W}_k$. This buffer must hold the i-th data until the calculation of $\beta(S_i)$ finish, so the succeeding data of $\mathbf{W}_{k+1}$ and $\mathbf{W}_{k+2}$ will be forwarded to the other two input buffers, and its contents will be overwritten with the data of $\mathbf{W}_{k+3}$. Because of the circular trellis structure of the IEEE 802.16m turbo codes, we need to do additional executions for metric initialization, including the forward metric of the last window and the dummy backward metric of the first window. For the SISO decoder that supports 3GPP LTE-Advanced standard, these executions will be skipped; instead, the decoder calculates the backward metric of the tail parts, from the $(\mathbf{N}+3)$-th trellis stage to the $\mathbf{N}$-th trellis stage, as soon as it gets the tail bits; then we have the $\beta(S_{\mathbf{N}})$'s to initialize the backward metric of the last windows in both constituent codes. Except for the distinction between their initial calculations, the SISO decoders for these two standards are alike in architecture and schedule.

Each decoding round can be divided as follows: both τ_a and τ_b are pipeline delay time and memory access time; τ_x is the interval for initial metric calculation between receiving the first input and producing the first output; and τ_y is the time to output all LLR and decisions. Note that $\mathbf{\Gamma_R}$ is the summation of the four periods, and $\mathbf{\Gamma_N}$ is equal to τ_y. Thus, the operating efficiency in (2.1) can be expressed as $(\tau_y)/(\tau_a + \tau_b + \tau_x + \tau_y)$. Based on Fig. 2.3b, the values of these execution periods are (2.10), where $\mathbf{\Gamma_L}$ is the cycle number that are required to run every kind of computations for an entire window.

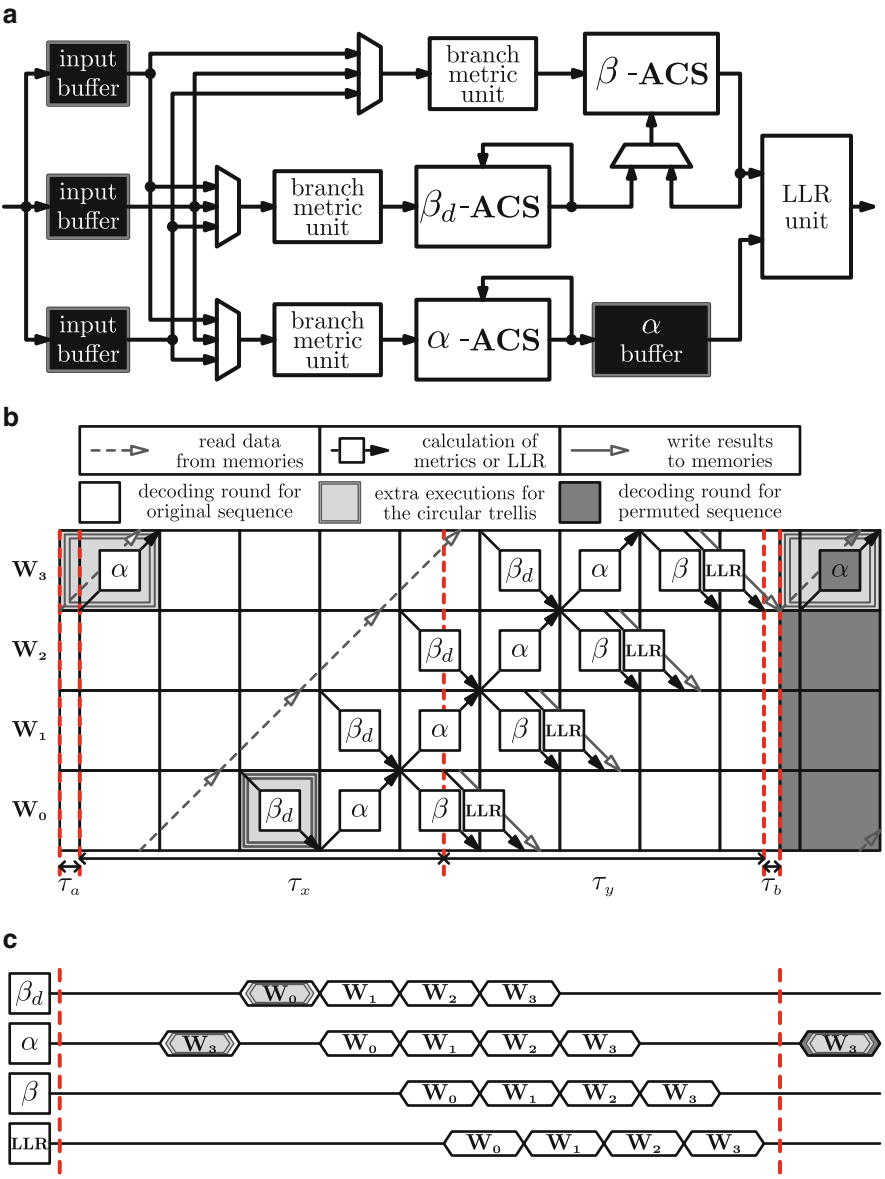

Fig. 2.3 Architecture and schedule of the conventional SISO decoder for the block with four windows. (**a**) SISO decoder architecture. (**b**) Processing schedule with $\eta = 44.4\,\%$. (**c**) Corresponding active periods of main components

$$\begin{cases} 0 \le \tau_a, \tau_b \le \mathbf{\Gamma_L} \\ 4\mathbf{\Gamma_L} \le \tau_x \le 5\mathbf{\Gamma_L} \\ \tau_y = 4\mathbf{\Gamma_L} \end{cases} \tag{2.10}$$

Typically, τ_a, τ_b, and τ_x are constant numbers, but τ_y will be in proportion to the block size. Here we assume that $(\tau_a+\tau_b+\tau_x)$ is $5\mathbf{\Gamma_L}$ and that $\mathbf{N}$ can be divisible by $\mathbf{L}$. While the SISO decoder has to process $\mathbf{N/L}$ windows, τ_y becomes $(\mathbf{N/L}) \times \mathbf{\Gamma_L}$ cycles. Now the operating efficiency can be roughly estimated by

$$\eta = \frac{\mathbf{N}}{\mathbf{N} + 5\mathbf{L}}. \tag{2.11}$$

If the tail-biting technique is not applied, the value of τ_x is between $3\mathbf{\Gamma_L}$ and $4\mathbf{\Gamma_L}$; and $(\tau_a + \tau_b + \tau_x)$ equals $4\mathbf{\Gamma_L}$; so η changes to $\mathbf{N}/(\mathbf{N} + 4\mathbf{L})$. In either case, the decoding process will be inefficient when $\mathbf{N/L}$ is small. Minimizing τ_a, τ_b, and τ_x of the SISO decoder is a trivial solution; however, the improvement is very slight owing to their bounds in (2.10). The negative effect raises concern about the inconsistent throughput of a practical turbo decoder that supports multiple block sizes. We are obliged to ensure the fulfillment of the throughput requirement even for the shortest $\mathbf{N}$.

Another disadvantage of decoding small blocks is the poor utilization of the component circuits. As shown in Fig. 2.3c with $\mathbf{N} = 4\mathbf{L}$, the main units remain idle for around half time of every decoding round. Nevertheless, they still consume certain energy during these inactive periods, and such waste is harmful to the average power. The enhancement of operating efficiency is undoubtedly the best solution to this problem. For minimizing unnecessary energy dissipation, some low-power techniques such as gated clock are often exploited, especially when the SISO decoder is more likely to deal with small blocks.

2.2.2 *Data Width and Normalization*

The issue of data width involves the quantization bits and expression format of the received symbols, *a priori* probabilities, metrics, and LLRs. Both of them are determined by the distribution of the channel noise, the correlation among these data, and the design objective. In the following discussions, we define the total bit number of each sort of data as $\boldsymbol{\varrho}(\cdot)$; then we use the notations $\varrho_{\mathrm{I}}(\cdot)$ and $\varrho_{\mathrm{F}}(\cdot)$ for integer and fractional bits respectively. The two's-complement numeral system is used, and it can represent the value in the range of $-2^{(\varrho_{\mathrm{I}}(\cdot)-1)}$ to $(2^{(\varrho_{\mathrm{I}}(\cdot)-1)} - 2^{(-\varrho_{\mathrm{F}}(\cdot))})$. Since the BPSK modulated coded data pass through the AWGN channel, almost every received symbol needs only single-digit number for its integer part; in other words, $\varrho_{\mathrm{I}}(r_i)$ can be tiny. If the quantity of r_i is beyond the range that $\boldsymbol{\varrho}(r_i)$ can represent, the quantized r_i will be clamped into the largest or the smallest value. Despite the rounding error, the performance degradation can be negligible with

sufficient data width [37]. For the other data, their integer bits will obey the rule in (2.12). The maximum value of quantized $L_a(u_i)$ must be larger than the quantized r_i so that the erroneous data can be corrected by the strong APP estimations. The relations between $L(u_i)$, $\alpha(S_i)$, $\beta(S_i)$, and $\gamma(S_i, S_{i+1})$ come from their respective equations.

$$\varrho_{\mathrm{I}}(L(u_i)) \geq \varrho_{\mathrm{I}}(\alpha(S_i)) = \varrho_{\mathrm{I}}(\beta(S_i)) \geq \varrho_{\mathrm{I}}(\gamma(S_i, S_{i+1})) \geq \varrho_{\mathrm{I}}(L_a(u_i)) \geq \varrho_{\mathrm{I}}(r_i) \quad (2.12)$$

While selecting the number of fractional bits, we first consider how closely the quantized input can approximate to the raw r_i. Larger $\varrho_{\mathrm{F}}(r_i)$ means higher accuracy of the input data. Once we decide the $\varrho_{\mathrm{F}}(r_i)$, the other data will need the same or fewer bits for their fractional parts.

$$\left\{\varrho_{\mathrm{F}}(L(u_i)), \varrho_{\mathrm{F}}(\alpha(S_i)), \varrho_{\mathrm{I}}(\beta(S_i)), \varrho_{\mathrm{F}}(\gamma(S_i, S_{i+1})), \varrho_{\mathrm{F}}(L_a(u_i))\right\} \leq \varrho_{\mathrm{F}}(r_i) \quad (2.13)$$

The fixed-point implementation can function properly only by continuously adjusting path metrics and extrinsic information. Figure 2.2a has stated that the adjustment for path metrics is normalization. Both the normalized $\alpha(S_i)$ and $\beta(S_i)$ must be expressed correctly by $\boldsymbol{\varrho}(\alpha(S_i))$ $(= \boldsymbol{\varrho}(\beta(S_i)))$ bits. The relevant techniques are based on the essential property in [38]: for the metrics of any two states at the i-th trellis stage, the absolute value of their difference is bounded; and the bounds for all i's are identical. For either code that we are interested in, the necessary bit number for the bound is a little larger than $\boldsymbol{\varrho}(r_i)$. The classic normalization is to subtract a constant number from all path metrics at the same trellis stage; then $\alpha(S_i)$ and $\beta(S_i)$ can be represented by a fixed amount of bits throughout the decoding process. The modification of extrinsic information is analogous to the saturation of quantized inputs. According to the arithmetic calculation of $L_e(u_i)$, $\boldsymbol{\varrho}(L_e(u_i))$ should be equal to or larger than $\boldsymbol{\varrho}(L(u_i))$, but converting it directly into the APP for the other constituent code may violate (2.12). To resolve this contradiction, $L_e(u_i)$ will be clamped to the extrema of $(\varrho_{\mathrm{I}}(L_a(u_i)), \varrho_{\mathrm{F}}(L_a(u_i)))$ when it is outside the representable range.

The most appropriate data widths of a practical turbo decoder are decided by checking whether its performance and complexity can reach the objective; and the straightforward method is running simulations. With the help of the above-mentioned rules and tactics, we can conduct a systematic search of some combinations of fixed-point representations. The performance is mostly affected by the quantization format $(\varrho_{\mathrm{I}}(\cdot), \varrho_{\mathrm{F}}(\cdot))$, whereas the hardware cost is dominated by the total width $\boldsymbol{\varrho}(\cdot)$. Hence, we examine the performance of all 16 cases with respect to $\{(\varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i)) \mid 2 \leq \varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i) \leq 5\}$ and compare the circuit area of designs with $\{\boldsymbol{\varrho}(r_i) \mid 4 \leq \boldsymbol{\varrho}(r_i) \leq 10\}$. The required internal precision varies with the code structure. For 3GPP LTE-Advanced turbo codes, the quantized data will satisfy $\varrho_{\mathrm{I}}(L_a(u_i)) = \varrho_{\mathrm{I}}(r_i) + 1$, $\varrho_{\mathrm{I}}(\gamma(S_i, S_{i+1})) = \varrho_{\mathrm{I}}(L_a(u_i)) + 1$, $\varrho_{\mathrm{I}}(\alpha(S_i)) = \varrho_{\mathrm{I}}(\beta(S_i)) = \varrho_{\mathrm{I}}(\gamma(S_i, S_{i+1})) + 1$, and $\varrho_{\mathrm{I}}(L(u_i)) = \varrho_{\mathrm{I}}(\alpha(S_i)) + 1$. Note that the path metrics are normalized by (2.14), where the offset value is the minimum path metric at the i-th trellis stage. The modified metrics are always

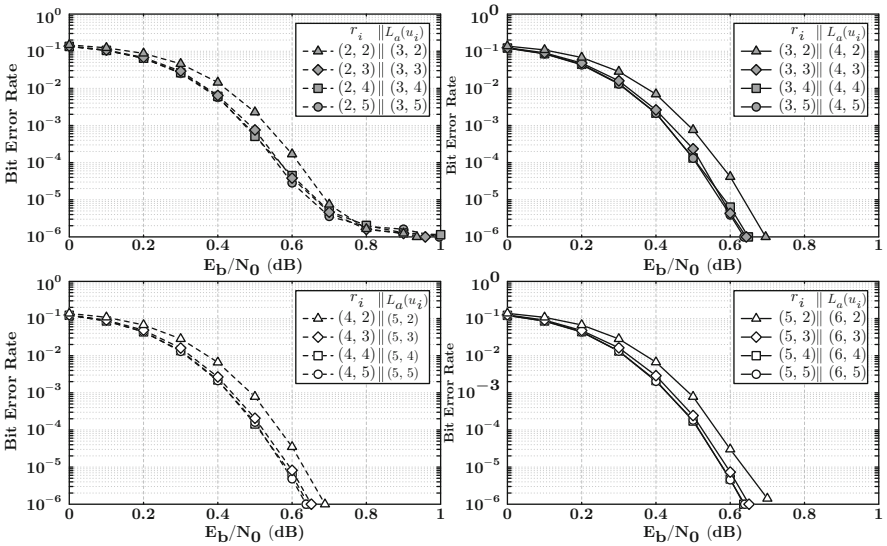

Fig. 2.4 Fixed-point simulation results of 3GPP LTE-Advanced turbo codes with Max-Log-MAP algorithm, $\mathcal{I} = 8$, $\mathbf{N} = 6144$, $\mathbf{L} = 32$, $\zeta = 0.75$, and different data width formats: $(\varrho_{\rm I}(r_i), \varrho_{\rm F}(r_i))$ and $(\varrho_{\rm I}(L_a(u_i)), \varrho_{\rm F}(L_a(u_i)))$

Table 2.1 Area of 3GPP LTE-Advanced turbo decoders with various data widths

		$\boldsymbol{\varrho}(r_i)$						
		4	**5**	**6**	**7**	**8**	**9**	**10**
∘	Turbo decoder	467.3	530.6	593.6	656.6	720.9	786.1	851.0
◇	SISO decoder	108.1	127.2	145.8	164.6	184.8	205.8	226.1
↦	Input buffer	37.3	39.3	46.6	53.4	60.6	67.7	74.7
↦	Branch metric unit	3.2	3.7	4.3	4.8	5.4	5.9	6.5
↦	α, β_d, β-ACS	17.7	20.7	23.6	26.6	31.0	36.4	41.0
↦	α buffer	38.9	45.4	51.8	58.4	64.8	71.2	77.7
↦	LLR unit	8.4	9.7	10.9	12.1	13.3	14.6	15.8
◇	Memory	324.6	368.4	412.3	456.0	499.6	543.3	587.5

⋇ Synthesis results with 10 ns clock period (area unit: $10^3 \times \mu m^2$)

positive or zeros, so it is needless to store their sign bits in the storage elements. The capacity of the forward metric buffer will be $8 \times \mathbf{L} \times (\boldsymbol{\varrho}(\alpha(S_i)) - 1)$ bits.

$$\begin{cases} \alpha(S_i) - \min\limits_{0 \le m \le 7} (\alpha(S_i = \mathrm{S}^{(m)})), \ \forall S_i \in \{\mathrm{S}^{(0)}, \cdots, \mathrm{S}^{(7)}\} \\ \beta(S_i) - \min\limits_{0 \le m \le 7} (\beta(S_i = \mathrm{S}^{(m)})), \ \forall S_i \in \{\mathrm{S}^{(0)}, \cdots, \mathrm{S}^{(7)}\} \end{cases} \tag{2.14}$$

Figure 2.4 provides the performance curves of these practical decoders. At the BER of 10^{-6}, the results of those cases with $\varrho_{\rm I}(r_i) \ge 3$ and $\varrho_{\rm F}(r_i) \ge 3$ are quite close to the floating-point simulation results with $\mathbf{L} = 32$ in Fig. 1.14a. Table 2.1

lists the areas of the main component circuits after the syntheses under loose timing constraints. The area growth is proportional to the increment in data width, but the percentage of each part remains stable. For IEEE 802.16m turbo code, the requisite $\varrho_{\rm I}(L_a(u_i))$ is $\varrho_{\rm I}(r_i)+2$, and the other rules are the same as those in the preceding example. Figure 2.5 and Table 2.2 give the corresponding simulation results. Similarly, the performance degradation can be minimized by selecting $\varrho_{\rm I}(r_i) \geq 3$ and $\varrho_{\rm F}(r_i) \geq 3$. The double binary code structure causes substantial rises in circuit area of most components; while the shorter window length makes the hardware cost for buffering $\alpha(S_i)$ decrease. Considering performance and overhead, $(\varrho_{\rm I}(r_i), \varrho_{\rm F}(r_i)) = (3,3)$ will be our first choice in later implementation examples for both applications.

The next issue of interest is the optimization of decoding speed. We run the syntheses with accuracy of 0.1 ns and look for the shortest clock period that can prevent negative slacks. These results indicate that the shortest periods of our SISO decoders using $(\varrho_{\rm I}(r_i), \varrho_{\rm F}(r_i)) = (3,3)$ are 3.4 ns for 3GPP LTE-Advanced application and 3.9 ns for IEEE 802.16m application. From the timing reports, the critical path locates in the functional unit for path metrics, and most execution time is spent on metric normalization. We need faster normalization methods to improve decoding speed. The functional unit in Fig. 2.2a will be modified, and the circuit area will be affected as well. A common normalization method is given in (2.15), where the offset value is always the path metric at state $\mathrm{S}^{(0)}$. The decoder can skip the comparison among all path metrics and perform subtractions immediately after the ACS operation; it has shorter path delays and costs fewer normalization circuits. The normalized $\alpha(S_i = \mathrm{S}^{(0)})$ is fixed to zero. There is no need to keep its value, so the forward metric buffer will store $7 \times \mathbf{L} \times \boldsymbol{\varrho}(\alpha(S_i))$ bits. In general, the design with (2.15) is smaller than the design with (2.14), provided they are implemented with equal clock periods.

$$\begin{cases} \alpha(S_i) - \alpha(S_i = \mathrm{S}^{(0)}),\ \forall S_i \in \{\mathrm{S}^{(0)}, \cdots, \mathrm{S}^{(7)}\} \\ \beta(S_i) - \beta(S_i = \mathrm{S}^{(0)}),\ \forall S_i \in \{\mathrm{S}^{(0)}, \cdots, \mathrm{S}^{(7)}\} \end{cases} \tag{2.15}$$

The modulo normalization is another way to avoid arithmetic overflow of the path metrics [38–41]. Unlike (2.14) and (2.15), the design with this technique only runs the ACS operations for path metric calculation. The data width will be extended to ensure the correctness of computation, but the increase in area is small. Due to the elimination of extra subtractors, the hardware cost of this functional unit is reduced, and it results in shorter path delay than the unit using (2.14). However, the overhead is transferred to the succeeding circuits. The capacity of the forward metric buffer grows to $8 \times \mathbf{L} \times \boldsymbol{\varrho}(\alpha(S_i))$ bits, and the LLR unit becomes more complicated to deal with those unrefined path metrics. It is probable that the overall cost will be larger than the design with (2.14).

Tables 2.3 and 2.4 list the synthesis results of the SISO decoders with various ways to normalize path metrics. Under the same loose timing constraint (10 ns), the designs with (2.14) has the largest ACS units for the path metrics, whereas the designs with modulo normalization has the largest buffers for $\alpha(S_i)$ and LLR units. These tables also indicate the shortest clock period of each SISO decoder

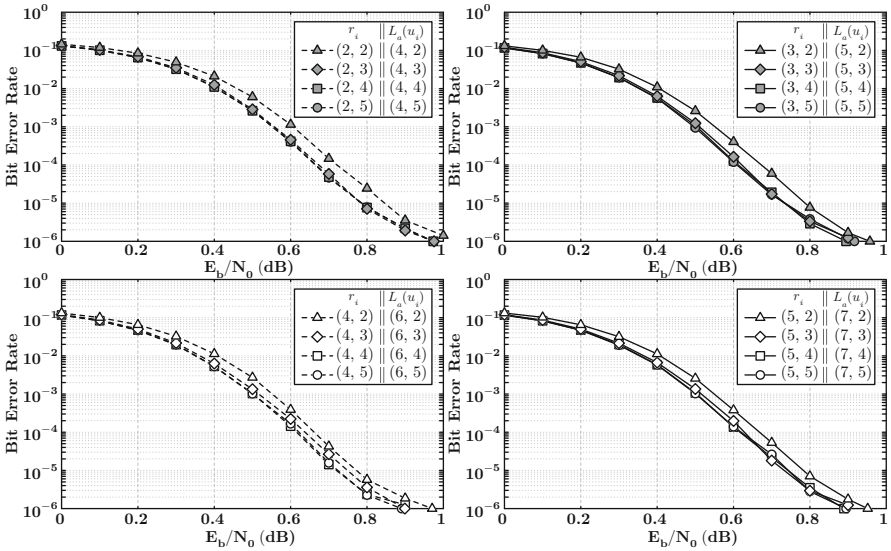

Fig. 2.5 Fixed-point simulation results of IEEE 802.16m turbo codes with Max-Log-MAP algorithm, $\mathcal{I} = 8$, $\mathbf{N} = 2400$, $\mathbf{L} = 16$, $\zeta = 0.75$, and different data width formats: $(\varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i))$ and $(\varrho_{\mathrm{I}}(L_a(u_i)), \varrho_{\mathrm{F}}(L_a(u_i)))$

Table 2.2 Area of IEEE 802.16m turbo decoders with various data widths

		$\boldsymbol{\varrho}(r_i)$						
		4	**5**	**6**	**7**	**8**	**9**	**10**
∘	Turbo decoder	495.6	570.5	645.4	724.1	812.8	891.0	967.7
◇	SISO decoder	142.2	164.3	186.7	212.7	248.9	274.3	298.6
↦	Input buffer	41.2	49.4	57.7	66.1	74.4	82.6	90.9
↦	Branch metric unit	16.2	19.1	22.1	25.1	28.0	31.0	34.0
↦	α, β_d, β-ACS	33.9	38.6	43.5	51.9	70.7	78.5	85.3
↦	α buffer	22.8	26.0	29.3	32.5	35.7	38.9	42.1
↦	LLR unit	20.6	23.3	25.9	28.6	31.3	34.0	36.7
◇	Memory	320.2	372.3	424.0	476.0	527.6	579.6	631.3

⋇ Synthesis results with 10 ns clock period (area unit: $10^3 \times \mu\mathrm{m}^2$).

and their respective overhead. The designs using either modulo normalization or (2.15) can operate 1.4 times faster than the design using (2.14). To meet the tightest constraints, the component circuits of each design require larger logic gates with greater driving capability, and there is a particularly rapid escalation in the area of all ACS units. Although the latter two methods can achieve similar speedup, the distinct code properties of these standards make their costs quite different.

Table 2.3 Area of conventional SISO decoders with various normalization methods for 3GPP LTE-Advanced standard (unit: $10^3 \times \mu m^2$)

Normalization		(2.14)		(2.15)		Modulo	
$\varrho(\alpha(S_i));\varrho(L(u_i))$		9;10		9;10		10;11	
Clock period (ns)		10.0	3.4	10.0	2.3	10.0	2.4
◇	SISO decoder	145.8	168.1	142.8	166.2	161.7	178.7
↦	Input buffer	46.3	46.4	46.3	46.9	46.3	46.6
↦	Branch metric unit	4.3	5.9	4.3	7.8	4.3	6.5
↦	α, β_d, β-ACS	23.6	42.9	20.9	34.6	20.7	31.0
↦	α buffer	51.8	51.8	51.1	51.2	64.8	64.8
↦	LLR unit	10.9	11.9	11.3	16.2	16.8	20.6

Table 2.4 Area of conventional SISO decoders with various normalization methods for IEEE 802.16m standard (unit: $10^3 \times \mu m^2$)

Normalization		(2.14)		(2.15)		Modulo	
$\varrho(\alpha(S_i));\varrho(L(u_i))$		10;11		10;11		11;12	
Clock period (ns)		10.0	3.9	10.0	2.5	10.0	2.6
◇	SISO decoder	186.7	233.1	183.8	274.4	201.0	229.2
↦	Input buffer	57.7	57.8	57.7	59.0	57.7	58.6
↦	Branch metric unit	22.1	26.7	22.1	41.9	22.1	29.4
↦	α, β_d, β-ACS	43.5	81.9	40.4	101.2	41.6	54.6
↦	α buffer	29.3	29.2	28.5	28.4	35.7	35.7
↦	LLR unit	25.9	28.8	26.8	34.9	35.6	42.3

2.3 Design of Modified SISO Decoders

The main drawback of the conventional SISO decoder in Fig. 2.3 is the low operating efficiency when small blocks are processed. From Fig. 2.3b and (2.10), it is clear that τ_x is the primary factor of such inefficiency. Shortening τ_x will be the direct way to alleviate this problem, and it involves several changes in processing schedule as well as in decoder architecture. There are two major types of modifications: reversing data order [42] and utilizing previous metrics [43, 44]. Besides the improvement in η, some functional units that are required in conventional SISO decoder can be eliminated. However, the two techniques may introduce extra overhead, and the modified decoder may requires more area than the conventional one in certain cases. To find out the techniques well suited to the designs for 3GPP LTE-Advanced standard and IEEE 802.16m standard, the corresponding gains in η and variations in area will be discussed in this section.

Figure 2.6 shows how the work in [42] arranges the input data of the SISO decoder. For each window, the received symbols are sent in descending order rather than in ascending order. Since the order matches the backward recursive operations, the β_d-ACS can get its required data immediately. It is needless to wait till the input buffers get all data of every window. This modified schedule can save $\mathbf{\Gamma_L}$ cycles. Furthermore, the design just needs to store the data of two succeeding windows

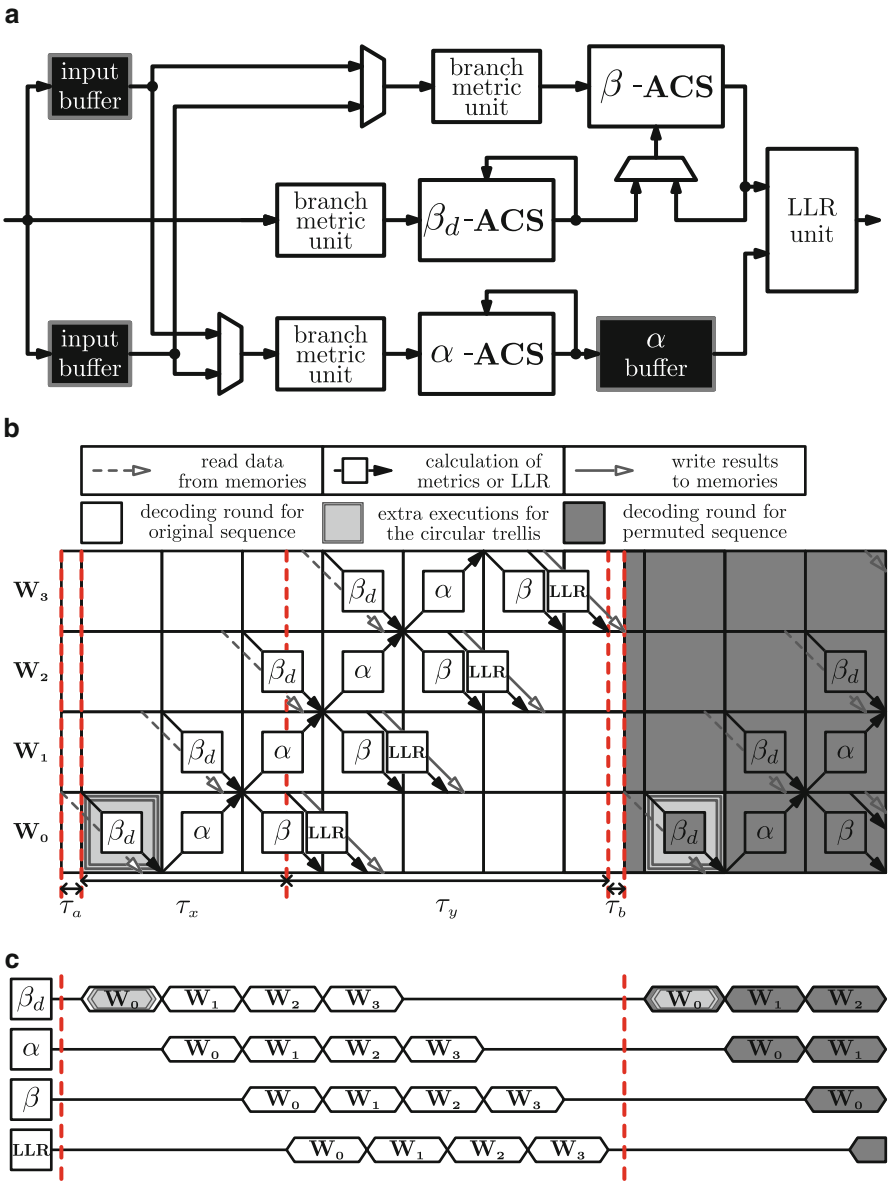

Fig. 2.6 Architecture and schedule of the SISO decoder using reverse input order for the block with four windows. (**a**) SISO decoder architecture. (**b**) Processing schedule with $\eta = 57.1\,\%$. (**c**) Corresponding active periods of main components

for α-ACS and β-ACS, so we can use less input buffers. The size of input buffers will be $2 \times \mathbf{L} \times (2\boldsymbol{\varrho}(r_i) + \boldsymbol{\varrho}(L_a(u_i)))$ bits for 3GPP LTE-Advanced application and $2 \times \mathbf{L} \times (4\boldsymbol{\varrho}(r_i) + 3\boldsymbol{\varrho}(L_a(u_i)))$ bits for IEEE 802.16m application. For turbo codes with circular trellis structure, the dummy forward metric calculation over the last window is also a cause of lengthy τ_x. Here we borrow the idea from [43, 44] to cut the initialization time. The $\alpha(S_0)$ will be updated by the $\alpha(S_{\mathbf{N}})$ at previous iteration, and we use $\alpha(S_0 = \mathrm{S}^{(m)}) = 0$ for $0 \le m \le 7$ at the first iteration. It makes a reduction of another $\boldsymbol{\Gamma}_{\mathbf{L}}$ cycles at the cost of a few storage elements for $16\boldsymbol{\varrho}(\alpha(S_i))$ bits. As a result, the τ_x in either case will decrease to within $2\boldsymbol{\Gamma}_{\mathbf{L}}$ and $3\boldsymbol{\Gamma}_{\mathbf{L}}$ cycles. If we let $(\tau_a + \tau_b + \tau_x)$ equal $3\boldsymbol{\Gamma}_{\mathbf{L}}$, then the η will be improved to $\mathbf{N}/(\mathbf{N} + 3\mathbf{L})$. The only difference between Figs. 2.3 and 2.6 is the order of input data. Both the SISO decoders will generate the same soft outputs; that is, they will get identical performance. Currently, the superior architecture and schedule have taken the place of the conventional SISO decoder.

The second technique can minimize τ_x by getting rid of all dummy metric calculations [43, 44]. Instead of computing $\beta_d(S_i)$ of $\mathbf{W}_{k+1}$, we exploit $\beta(S_{(k+1)\mathbf{L}})$ at the last iteration as the initialization of $\beta(S_i)$ of $\mathbf{W}_k$. Figure 2.7 illustrates the modified architecture and process. Now the SISO decoder consists of α-ACS and β-ACS. Besides the removal of the β_d-ACS and its branch metric unit, the size of total input buffer is only half as that of the design in Fig. 2.6a. On the other hand, we add some registers for the previous metrics. The data will be inputted in ascending order, and the α-ACS can start its work right away. The value of τ_x is between $\boldsymbol{\Gamma}_{\mathbf{L}}$ and $2\boldsymbol{\Gamma}_{\mathbf{L}}$, and the η becomes $\mathbf{N}/(\mathbf{N} + 2\mathbf{L})$. Even with the noticeable improvement in operating efficiency, the performance might degrade because the metrics at previous iterations are usually less reliable than the dummy metric calculation at current iteration. The storage of these previous metrics will be another serious obstacle to implement this design. To keep the initial backward metrics for all windows of both constituent codes, the additional registers must store at most $14 \times (\mathbf{N}/\mathbf{L}) \times \boldsymbol{\varrho}(\alpha(S_i))$ or $16 \times (\mathbf{N}/\mathbf{L}) \times \boldsymbol{\varrho}(\alpha(S_i))$ bits, depending on which normalization method is employed. Like the other decoders that support circular trellis structure, it also requires $16\boldsymbol{\varrho}(\alpha(S_i))$-bit registers for forward metrics. When the designs need to process lots of windows, the area of these registers will far outweigh the savings in other hardware. The concern for performance and overhead will limit the applicability of this technique to 3GPP LTE-Advanced and IEEE 802.16m turbo decoders.

Figure 2.8 shows that, for both applications, the design with the architecture and schedule in Fig. 2.7 will lead to about 0.1 dB performance loss. Basically, the slightly inferior BER is tolerable. Table 2.5 gives the synthesis results of the two modified SISO decoders for 3GPP LTE-Advanced standard. Here we make a comparison with the conventional designs in Table 2.3. With the modification as Fig. 2.6, there is a moderate reduction in area. While we utilize the SISO decoder in Fig. 2.7, the first three component circuits require much less area. However, the storage for the initial metrics of 384 windows ($\mathbf{N} = 6144, \mathbf{L} = 32$) makes the overall hardware cost escalate. We also list the area of SISO decoders for IEEE 802.16m standard in Table 2.6. Similarly, using the SISO decoder in Fig. 2.6 can

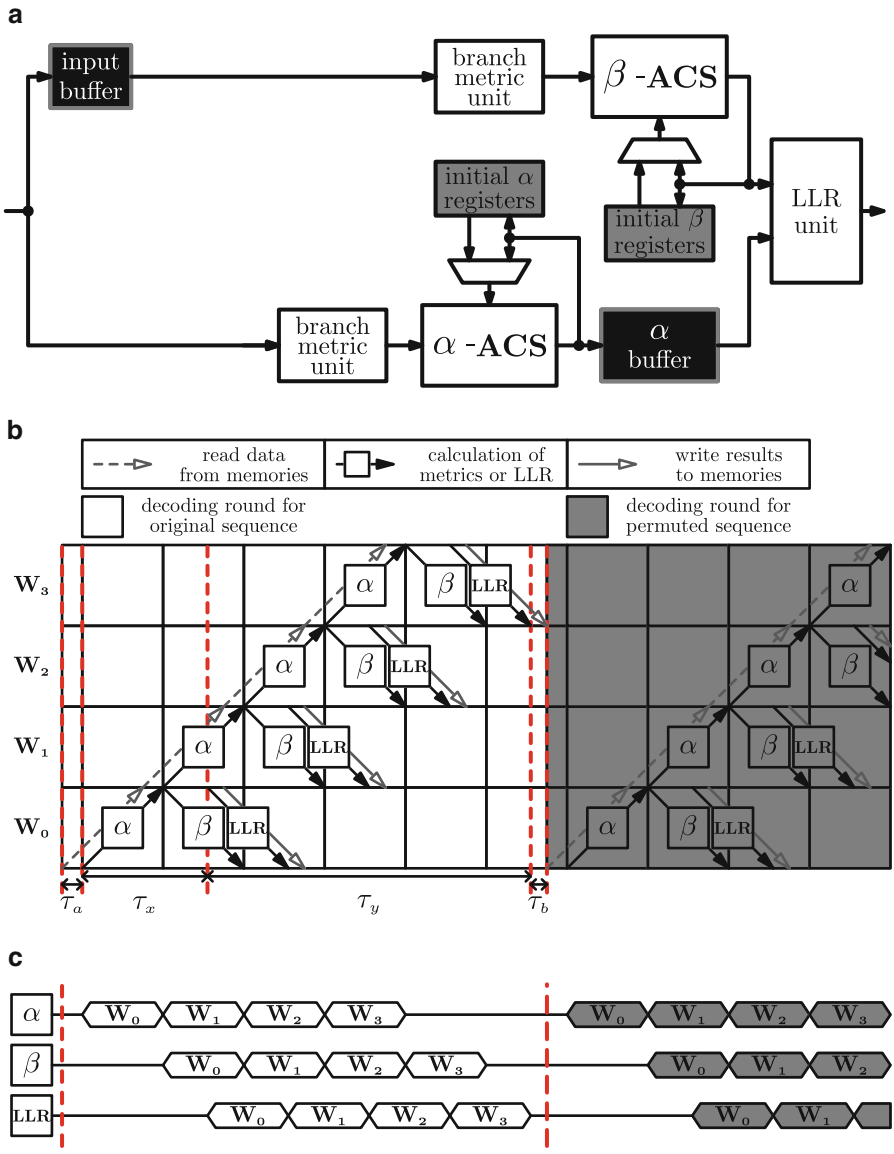

Fig. 2.7 Architecture and schedule of the SISO decoder using previous metrics for the block with four windows. (**a**) SISO decoder architecture. (**b**) Processing schedule with $\eta = 66.7\,\%$. (**c**) Corresponding active periods of main components

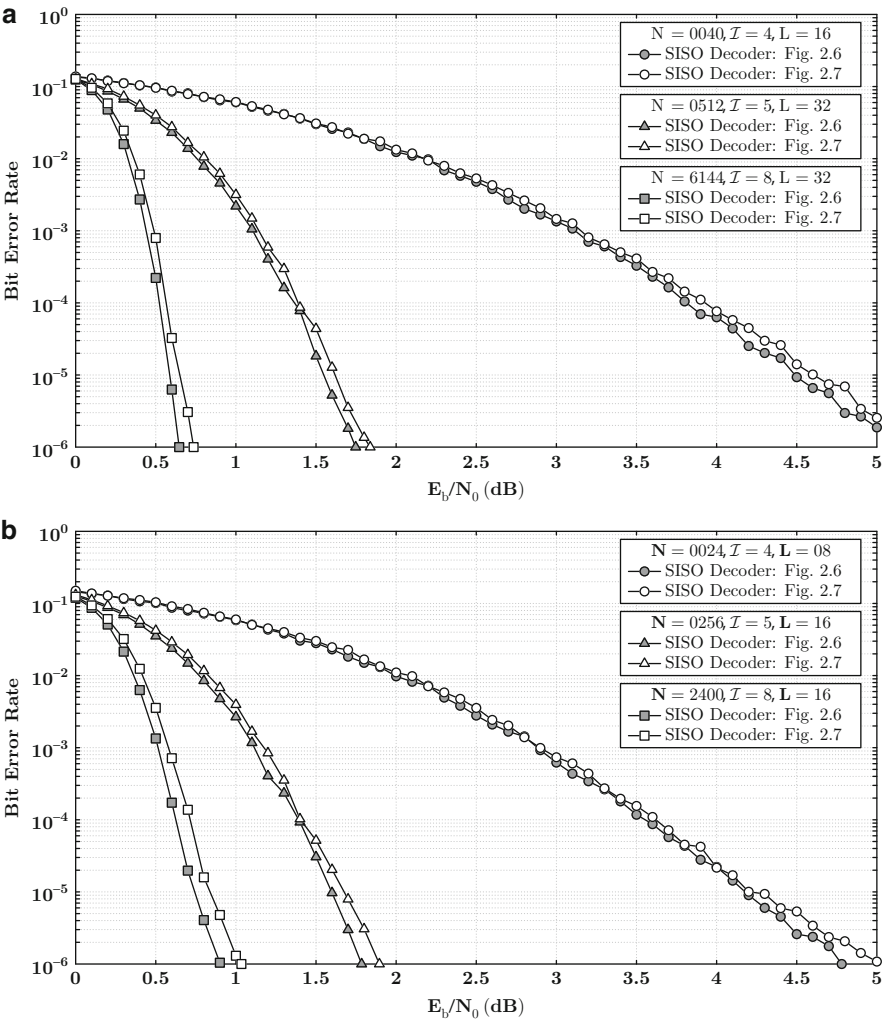

Fig. 2.8 Fixed-point simulation results of 3GPP LTE-Advanced and IEEE 802.16m turbo codes with various processing schedules: Max-Log-MAP algorithm with $\zeta = 0.75$ and $(\varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i)) = (3, 3)$. (**a**) 3GPP LTE-Advanced turbo codes: $\mathbf{N} = \{40, 512, 6144\}$. (**b**) IEEE 802.16m turbo codes: $\mathbf{N} = \{24, 256, 2400\}$

get the minimum overhead. For the SISO decoder in Fig. 2.7, now it has to store the initial metrics of 300 windows ($\mathbf{N} = 2400, \mathbf{L} = 16$), but the difference in overhead between both modified SISO decoders with (2.15) is insignificant. From these results, both 3GPP LTE-Advanced and IEEE 802.16m turbo decoders prefer the design in Fig. 2.6, and they should exploit (2.15) and modulo normalization respectively. Unless the window number ($2 \times \mathbf{N}/\mathbf{L}$) decreases, most practical decoders rarely consider the second type of architecture and schedule. Actually,

Table 2.5 Area of modified SISO decoders with various normalization methods for 3GPP LTE-Advanced standard (unit: $10^3 \times \mu m^2$)

SISO decoder		Figure 2.6		Figure 2.7	
Normalization		(2.15)	Modulo	(2.15)	Modulo
$\varrho(\alpha(S_i));\varrho(L(u_i))$		9;10	10;11	9;10	10;11
Clock period (ns)		2.3	2.4	2.4	2.4
◇	SISO decoder	146.4	162.2	218.1	256.1
↦	Input buffer	30.9	31.1	15.8	15.5
↦	Branch metric unit	6.5	5.9	4.2	3.8
↦	α, β_d, β-ACS	34.6	32.8	27.0	25.0
↦	α buffer	51.2	64.8	51.2	64.8
↦	LLR unit	16.2	20.7	20.0	20.7
↦	Initial β registers	–	–	97.8	119.3

Table 2.6 Area of modified SISO decoders with various normalization methods for IEEE 802.16m standard (unit: $10^3 \times \mu m^2$)

SISO decoder		Figure 2.6		Figure 2.7	
Normalization		(2.15)	Modulo	(2.15)	Modulo
$\varrho(\alpha(S_i));\varrho(L(u_i))$		10;11	11;12	10;11	11;12
Clock period (ns)		2.5	2.6	2.7	2.7
◇	SISO decoder	244.0	207.1	270.0	275.2
↦	Input buffer	39.7	39.7	19.9	19.9
↦	Branch metric unit	28.2	17.1	18.1	16.7
↦	α, β_d, β-ACS	77.2	59.6	74.0	45.2
↦	α buffer	28.4	35.7	28.4	35.7
↦	LLR unit	35.0	42.1	32.2	40.3
↦	Initial β registers	–	–	92.6	112.3

the cost of these extra registers is also determined by the CMOS technology and memory structure. As either the design objective or available resource varies, it is still possible to use the SISO decoder in Fig. 2.7 to get the best operating efficiency at relatively lower cost.

Chapter 3
Turbo Decoder with Parallel Processing

The operating frequency $\mathcal{F}$ is the decisive factor in throughput calculation of conventional turbo decoders. Although the modification to circuits can improve this factor, there is a limit to the critical path delay, and it is difficult to supply a stable clock signal with high frequency. We need other methods to further raise the decoding speed. The general idea is exploiting parallel architecture, which includes the turbo decoder level, the SISO decoder level, and the trellis stage level [45, 46]. In the turbo decoder level, multiple dedicated turbo decoders are used to decode multiple codeword blocks independently. In the SISO decoder level, every codeword block is split into several sub-blocks first, and then these sub-blocks are processed by multiple SISO decoders simultaneously. In the trellis stage level, the functional units inside the SISO decoder are duplicated to complete the computations related to two or more trellis stages within one clock cycle. This chapter will describe the features of each level. The parallel turbo decoder level is an intuitive method, so we only give a brief introduction. Our discussions center on the parallel SISO decoder level in particular because the turbo codes of 3GPP LTE-Advanced standard and IEEE 802.16m standard, or more precisely, the QPP interleaver in (1.1) and the ARP interleaver in (1.3) are designed to support this type of architecture. The available divisions of one codeword block and the largest parallelism in each application are stated first, then the variations of decoding speed and performance are presented. The parallel trellis stage level needs a minor modification to the decoding algorithm, and it also can be employed by the turbo decoders for above-mentioned standards. Even with the same parallelism, the gains of these levels are dissimilar, and so are their respective costs. Thus, the selection of the best parallel architecture will depend on the required throughput and hardware resource.

C.-C. Wong and H.-C. Chang, *Turbo Decoder Architecture for Beyond-4G Applications*, DOI 10.1007/978-1-4614-8310-6_3,

3.1 Multiple Turbo Decoders for Multiple Codewords

The fundamental concept of this parallel level is duplication of the whole turbo decoder. It has several individual sets of memory modules and SISO decoder. A global controller will deliver one received codeword block from input ports to the memory modules of one turbo decoder via a de-multiplexer; then it activates the corresponding SISO decoder to calculate the soft values of information symbols. During the above process, the design can deal with the following blocks by assigning them to other unoccupied turbo decoders. These turbo decoders are connected to a multiplexer so that the decisions of the sequential blocks can be sent to output ports through it. The parallel design still spends $2\mathcal{I}\mathbf{N}/\eta$ cycles decoding every codeword block, but the processes of new blocks can start much earlier. If $\mathcal{P}_c$ turbo decoders are used, the throughput could increase from (2.2) to at most

$$\Theta_c = \frac{\mathcal{P}_c \times \phi \times \mathcal{F} \times \eta}{2 \times \mathcal{I}}. \tag{3.1}$$

The theoretically maximum parallelism is determined by how many codeword blocks are received within the decoding latency. Assuming that the reception of one block takes $\mathbf{N}$ cycles, the upper bound of $\mathcal{P}_c$ will be $2\mathcal{I}/\eta$. The use of parallel turbo decoder level is free from any constraint on interleavers, and it can achieve exactly the same performance as traditional turbo decoders. However, there is a potential problem with the memory modules for extra codeword blocks. The actual value of $\mathcal{P}_c$ can be estimated by dividing the allowable design area by the area of a single turbo decoder. When $\mathbf{N}$ is large, the overhead of memory modules will increase, and the attainable $\mathcal{P}_c$ might decrease. Consequently, the parallel turbo decoder level is usually applied for small blocks or low parallelism to lessen the effect of the drawback.

3.2 Multiple SISO Decoders for One Codeword

3.2.1 Important Characteristics

In this parallel architecture, $\mathcal{P}_s$ SISO decoders work together to decode one size-$\mathbf{N}$ codeword block. According to decoding rounds, each SISO decoder deals with $\mathbf{M}$ ($\mathbf{M} = \mathbf{N}/\mathcal{P}_s$) successive data of the original sequence or permuted sequence. All SISO decoders will submit requests for their respective data at the same time. To support the simultaneous access, we store the codeword block in $\mathcal{P}_s$ separate memory modules, of which the s-th memory ($0 \leq s < \mathcal{P}_s$) usually includes one sub-block: $(r_{s\mathbf{M}}, r_{s\mathbf{M}+1}, \ldots, r_{s\mathbf{M}+(\mathbf{M}-1)})$. With such allocation, the s-th SISO decoder will continue accessing $r_{s\mathbf{M}+j}$ from the s-th sub-block memory for $j = 0$ to $(\mathbf{M}-1)$ during the decoding rounds for original sequence. When the design processes the

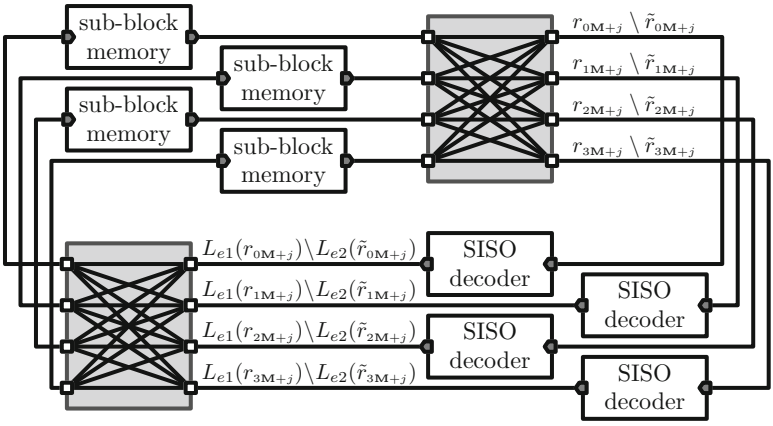

Fig. 3.1 Architecture with parallel SISO decoder level ($\mathcal{P}_S = 4$)

permuted data sequence with a certain interleaving function $\mathbf{\Pi}(\cdot)$, the s-th SISO decoder will request $\tilde{r}_{s\mathbf{M}+j}$ from the $\lfloor \mathbf{\Pi}(s\mathbf{M}+j)/\mathbf{M} \rfloor$-th sub-block memory. The $\mathcal{P}_S$ SISO decoders will send their decoding results to the memory modules in the same manner as they get their respective inputs. Every SISO decoder must be able to access any sub-block memory, so the design needs interconnection networks to support all possible data transmission. Most designs utilize the fully-connected network to link SISO decoders and memory modules for its powerful functionality. A typical example with $\mathcal{P}_S = 4$ is given in Fig. 3.1, where the networks can correctly direct the parallel data to their destinations.

The most important issue of the parallel SISO decoder level is manipulating the apparatus for parallel data transmission. The controller needs to generate the addresses for accessing memory modules and the instructions for arranging interconnection networks. However, one sub-block memory might be accessed by two or more decoders at the same time. When this collision problem occurs, the design will halt the ordinary process and start an interrupt routine [47, 48]. It takes several cycles to handle each of the concurrent requests, one by one. Such increased workload results in extended processing time and inferior throughput improvement. The only advantage is its compatibility with all turbo decoders. Unless the design is unlikely to encounter the collision problem, it is impractical to apply this solution.

Another strategy is preventing the occurrence of this problem, and there are two types of methodologies. The first one is the management of data storage. The algorithm proposed in [49] can obtain a particular function that maps each data to one of the $\mathcal{P}_S$ memories. As the turbo decoder stores data in accordance with this mapping function, it can be free from collision problems throughout all decoding rounds. This solution is suitable for any turbo code, but the decoder needs more memories to keep all special memory addresses and interconnection patterns for every parameter set. If the application involves many block sizes, the overhead might become unaffordable. The second methodology is the adoption

of contention-free interleavers [18, 20, 50–55], of which two representatives are QPP and ARP interleavers. They can ensure instant access to the $\mathcal{P}_S$ memories of the designs employing simple, regular mapping functions. Because of their well-organized permutation rules, the parallel architecture can handle data access at low cost. Additionally, they promise outstanding error correction capability. The benefits make the use of contention-free interleavers supersede the other solutions to the collision problem.

The parameters of contention-free interleavers determine the parallelism and memory mapping. While QPP interleavers are adopted, the permissible number of SISO decoders can be any factor of block size. Since the greatest common divisor of all $\mathbf{N}$'s in 3GPP LTE-Advanced turbo codes is 8, the corresponding design typically consists of 8 SISO decoders and 8 memory modules. It is also possible to use more SISO decoders and alter the parallelism in various cases. For example, a 3GPP LTE-Advanced turbo decoder with $\mathcal{P}_S = 32$ can activate all SISO decoders for $1024 \leq \mathbf{N}$, 16 SISO decoders for $512 \leq \mathbf{N} \leq 1024$, and 8 SISO decoders for $\mathbf{N} \leq 512$. The QPP interleavers are aimed for the division mapping [18, 56]; that is, the data in the s-th memory module must comply with

$$\{r_{s\mathbf{M}+j} \mid 0 \leq j < \mathbf{M}\}. \tag{3.2}$$

The control over these memories needs to locate where $\tilde{r}_{s\mathbf{M}+j}$ is stored during the decoding rounds for the permuted data sequence. For this purpose, the interleaving function is modified to match (3.2). Here we replace i in (1.1) with $(s\mathbf{M} + j)$ and rewrite the interleaving index as (3.3). This equation indicates that $\tilde{r}_{s\mathbf{M}+j}$ is equivalent to $r_{s^{\mathcal{Q}}_{s;j}\mathbf{M}+j^{\mathcal{Q}}_{s;j}}$; the value of $s^{\mathcal{Q}}_{s;j}$ and $j^{\mathcal{Q}}_{s;j}$ can be derived by (3.4) and (3.5) respectively.

$$\begin{aligned}
\mathcal{Q}(s\mathbf{M}+j) &= f_1 \times (s\mathbf{M}+j) + f_2 \times (s\mathbf{M}+j)^2 && (\text{mod } \mathbf{N}) \\
&= s^{\mathcal{Q}}_{s;j}\mathbf{M} + j^{\mathcal{Q}}_{s;j} && (\text{mod } \mathbf{N}) && (3.3) \\
s^{\mathcal{Q}}_{s;j} &= \lfloor \mathcal{Q}(s\mathbf{M}+j)/\mathbf{M} \rfloor && (\text{mod } \mathcal{P}_S) && (3.4) \\
j^{\mathcal{Q}}_{s;j} &= f_1 s\mathbf{M} + f_2 s^2\mathbf{M}^2 + 2f_2 s\mathbf{M}j + f_1 j + f_2 j^2 && (\text{mod } \mathbf{M}) \\
&= f_1 j + f_2 j^2 = \mathcal{Q}(j) && (\text{mod } \mathbf{M}) && (3.5)
\end{aligned}$$

Note that $j^{\mathcal{Q}}_{s;j}$ is independent of s; $\{j^{\mathcal{Q}}_{0;j}, j^{\mathcal{Q}}_{1;j}, \ldots, j^{\mathcal{Q}}_{\mathcal{P}_S-1;j}\}$ are the same in value. The controlling unit can calculate $j^{\mathcal{Q}}_{0;j}$ only and send the result to all memories as the read or write address; and it can use the $\mathcal{P}_S$ pairs of $(s, s^{\mathcal{Q}}_{s;j})$ to establish links between the s-th SISO decoder and the $s^{\mathcal{Q}}_{s;j}$-th memory. The main computations include the original QPP interleaving function and the division by $\mathbf{M}$. By the recursive computations introduced in Chap. 2, $\mathcal{Q}(s\mathbf{M} + j)$ can be generated promptly at every cycle. Thanks to the limited range of $\mathcal{Q}(s\mathbf{M} + j)$ and the predetermined $\mathbf{M}$, the controlling unit can perform the division in (3.4) with simple subtractors rather than complex dividers. Therefore, the transition from $\mathcal{Q}(s\mathbf{M}+j)$ to $s^{\mathcal{Q}}_{s;j}$ and $j^{\mathcal{Q}}_{s;j}$ will be done in short time at reasonable cost.

Based on the design concepts of ARP interleavers [20, 56], the fitting scheme of the corresponding parallel architecture is modulo mapping as (3.6): the $\hat{s}$-th memory stores the data whose indexes are congruent to $\hat{s}$ modulo $\mathcal{P}_\mathcal{S}$.

$$\{r_{\hat{s}+\hat{j}\mathcal{P}_\mathcal{S}} \mid 0 \le \hat{j} < \mathbf{M}\}. \tag{3.6}$$

Under the scheme, both $\mathcal{P}_\mathcal{S} = 2$ and $\mathcal{P}_\mathcal{S} = 4$ are valid for all parameter sets in IEEE 802.16m turbo codes. Higher parallelism can be permitted as long as the parameters satisfy $4 \mid \mathcal{P}_\mathcal{S}$ and $\mathcal{P}_\mathcal{S} \mid \mathbf{N}$. Besides these restrictions, the sequence length of the data sent to each SISO decoder must be relatively prime to $\mathcal{P}_\mathcal{S}$. Thus, the actual sequence length must be adjusted accordingly. We can simplify the adjustment by letting $\mathcal{P}_\mathcal{S}$ be a power of 2. Now the s-th SISO decoder will deal with $[r_{s\mathbf{M}'}, r_{s\mathbf{M}'+1}, \ldots, r_{s\mathbf{M}'+(\mathbf{M}'-1)}]$ and $[\tilde{r}_{s\mathbf{M}'}, \tilde{r}_{s\mathbf{M}'+1}, \ldots, \tilde{r}_{s\mathbf{M}'+(\mathbf{M}'-1)}]$, where $\mathbf{M}'$ is defined as

$$\mathbf{M}' = \begin{cases} \mathbf{N}/\mathcal{P}_\mathcal{S} + 1 & \text{if } (\mathbf{N}/\mathcal{P}_\mathcal{S}) \equiv 0 \pmod 2, \\ \mathbf{N}/\mathcal{P}_\mathcal{S} & \text{if } (\mathbf{N}/\mathcal{P}_\mathcal{S}) \equiv 1 \pmod 2. \end{cases} \tag{3.7}$$

When $(\mathbf{N}/\mathcal{P}_\mathcal{S})$ is an even number and $\mathbf{M}'$ is larger than $\mathcal{P}_\mathcal{S}$, $[r_0, r_1, \ldots, r_{\mathcal{P}_\mathcal{S}-1}]$ and $[\tilde{r}_0, \tilde{r}_1, \ldots, \tilde{r}_{\mathcal{P}_\mathcal{S}-1}]$ will be processed twice, by the first SISO decoder and then by the last SISO decoder, at each of their respective decoding rounds. Although this decoding process takes a few more cycles for the extended sub-block size, the decrease in throughput is slight and tolerable. The utilization of modulo mapping has a great impact on control signal generation. The controller has to translate $(s\mathbf{M}' + j)$ into $(\hat{\boldsymbol{s}}_{s;j} + \hat{\boldsymbol{j}}_{s;j}\mathcal{P}_\mathcal{S})$ by (3.8) and (3.9) to find which memory contains $r_{s\mathbf{M}'+j}$. Similarly, the translation from $\mathcal{A}(s\mathbf{M}' + j)$ into $(\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} + \hat{\boldsymbol{j}}^{\mathcal{A}}_{s;j}\mathcal{P}_\mathcal{S})$ by (3.10) and (3.11) is necessary for accessing $\tilde{r}_{s\mathbf{M}'+j}$.

$$\hat{\boldsymbol{s}}_{s;j} = (s\mathbf{M}' + j) \pmod{\mathcal{P}_\mathcal{S}} \tag{3.8}$$

$$\hat{\boldsymbol{j}}_{s;j} = \left\lfloor (s\mathbf{M}' + j)/\mathcal{P}_\mathcal{S} \right\rfloor \tag{3.9}$$

$$\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} = \mathcal{A}(s\mathbf{M}' + j) \pmod{\mathcal{P}_\mathcal{S}} \tag{3.10}$$

$$\hat{\boldsymbol{j}}^{\mathcal{A}}_{s;j} = \left\lfloor \mathcal{A}(s\mathbf{M}' + j)/\mathcal{P}_\mathcal{S} \right\rfloor \tag{3.11}$$

Both s and j affect the values of $\hat{\boldsymbol{j}}_{s;j}$ and $\hat{\boldsymbol{j}}^{\mathcal{A}}_{s;j}$. For a given j, $\{\hat{\boldsymbol{j}}_{0;j}, \hat{\boldsymbol{j}}_{1;j}, \ldots, \hat{\boldsymbol{j}}_{\mathcal{P}_\mathcal{S}-1;j}\}$ tend to be dissimilar, and so do $\{\hat{\boldsymbol{j}}^{\mathcal{A}}_{0;j}, \hat{\boldsymbol{j}}^{\mathcal{A}}_{1;j}, \ldots, \hat{\boldsymbol{j}}^{\mathcal{A}}_{\mathcal{P}_\mathcal{S}-1;j}\}$. The design needs multiple address generators to get these addresses and another network to send them to right memories. If the premise that $\log_2 \mathcal{P}_\mathcal{S}$ is an integer holds, $\hat{\boldsymbol{s}}_{s;j}$ and $\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$ are the copies of the least significant $\log_2 \mathcal{P}_\mathcal{S}$ bits of $(s\mathbf{M}' + j)$ and $\mathcal{A}(s\mathbf{M}' + j)$ respectively, while $\hat{\boldsymbol{j}}_{s;j}$ and $\hat{\boldsymbol{j}}^{\mathcal{A}}_{s;j}$ are the remaining most significant parts. It will be an effortless conversion from $(s\mathbf{M}' + j)$ and $\mathcal{A}(s\mathbf{M}' + j)$ to control signals.

Actually, as the parameters satisfy (3.12) for all j's, the parallel architecture with ARP interleavers can utilize division mapping in (3.2), too.

$$\{g_{(s\mathbf{M}+j) \bmod 4} \equiv g_{(s'\mathbf{M}+j) \bmod 4} \pmod{\mathbf{M}} \mid 0 \le s < s' < \mathcal{P}_\mathcal{S}\} \tag{3.12}$$

The applicability can be proven easily by comparing the interleaving indexes: $\{\mathcal{A}(j), \mathcal{A}(\mathbf{M}+j), \ldots, \mathcal{A}((\mathcal{P}_S-1)\mathbf{M}+j)\}$. The initial step is the substitution of $(s\mathbf{M}+j)$ for i in (1.3). Then we convert the result to (3.13) with $\boldsymbol{s}^{\mathcal{A}}_{s;j}$ in (3.14) and $\boldsymbol{j}^{\mathcal{A}}_{s;j}$ in (3.15). After applying the constraint in (3.12), $\{\boldsymbol{j}^{\mathcal{A}}_{0;j}, \boldsymbol{j}^{\mathcal{A}}_{1;j}, \ldots, \boldsymbol{j}^{\mathcal{A}}_{\mathcal{P}_S-1;j}\}$ are equal to each other. Because $\mathcal{A}(s\mathbf{M}+j) \neq \mathcal{A}(s'\mathbf{M}+j)$ is always true if s and s' are different, the equivalence $\boldsymbol{j}^{\mathcal{A}}_{s;j} = \boldsymbol{j}^{\mathcal{A}}_{s';j}$ implies $\boldsymbol{s}^{\mathcal{A}}_{s;j} \neq \boldsymbol{s}^{\mathcal{A}}_{s';j}$. It shows that this scheme also possesses the contention-free property.

$$\begin{aligned}
\mathcal{A}(s\mathbf{M}+j) &= \varepsilon \times (s\mathbf{M}+j) + g_{(s\mathbf{M}+j) \bmod 4} && (\text{mod } \mathbf{N}) \\
&= \boldsymbol{s}^{\mathcal{A}}_{s;j}\mathbf{M} + \boldsymbol{j}^{\mathcal{A}}_{s;j} && (\text{mod } \mathbf{N}) && (3.13) \\
\boldsymbol{s}^{\mathcal{A}}_{s;j} &= \lfloor \mathcal{A}(s\mathbf{M}+j)/\mathbf{M} \rfloor && (\text{mod } \mathcal{P}_S) && (3.14) \\
\boldsymbol{j}^{\mathcal{A}}_{s;j} &= \varepsilon s\mathbf{M} + \varepsilon j + g_{(s\mathbf{M}+j) \bmod 4} && (\text{mod } \mathbf{M}) \\
&= \varepsilon j + g_{(s\mathbf{M}+j) \bmod 4} && (\text{mod } \mathbf{M}) && (3.15)
\end{aligned}$$

Obviously, it is unwise to check whether all possible indexes and parameters can meet (3.12). To reduce the computational effort, we exploit the periodicity of $\{g_0, g_1, g_2, g_3\}$ and develop three alternative conditions. The first one requires that $\mathbf{M}$ is divisible by 4, and we can infer $g_{(s\mathbf{M}+j) \bmod 4} = g_{j \bmod 4}$ for any s [56]. The second condition is (3.16); it promises that, regardless of the values of $\mathbf{M}$, s and j, there is only one outcome of $g_{(s\mathbf{M}+j) \bmod 4}$ modulo $\mathbf{M}$.

$$g_0 \equiv g_1 \equiv g_2 \equiv g_3 \pmod{\mathbf{M}} \tag{3.16}$$

The last condition in (3.17) specifies a special case of even-numbered $\mathbf{M}$'s, where $g_{(s\mathbf{M}+j) \bmod 4}$ can be simplified to either $g_{j \bmod 4}$ or $g_{(j+2) \bmod 4}$. With the equivalence relations imposed here, $g_{j \bmod 4}$ and $g_{(j+2) \bmod 4}$ are congruent modulo $\mathbf{M}$.

$$\begin{cases} \mathbf{M} \equiv 0 \pmod 2 \\ g_0 \equiv g_2 \pmod{\mathbf{M}} \\ g_1 \equiv g_3 \pmod{\mathbf{M}} \end{cases} \tag{3.17}$$

These conditions can remove the dependence of $g_{(s\mathbf{M}+j) \bmod 4}$ on s. Any of $4 \mid \mathbf{M}$, (3.16), and (3.17) can lead to the fulfillment of (3.12). Since they just involve the parameters $\{\mathbf{M}, g_0, g_1, g_2, g_3\}$, their verification is much easier than a thorough examination of (3.12). In IEEE 802.16m turbo codes, the numbers of parameter sets that can support division mapping are 25 for $\mathcal{P}_S = 4$, 18 for $\mathcal{P}_S = 8$, and 9 for $\mathcal{P}_S = 16$. The controller for this scheme comprises the address generators for ARP interleaving functions and the converters for (3.14) and (3.15). It is possible to integrate this with the controller for modulo mapping in the same turbo decoder, and the cost can be minimized by sharing the core component circuits. Such flexibility will enable a wider range of applications.

Table 3.1 Speedup of 3GPP LTE-Advanced turbo decoders with $\mathcal{P}_S = \{2, 4, 8\}$

			$\mathcal{P}_S = 2$		$\mathcal{P}_S = 4$		$\mathcal{P}_S = 8$	
N	**L**	η (%)	η_S (%)	$\mathcal{P}_S\eta_S/\eta$	η_S(%)	$\mathcal{P}_S\eta_S/\eta$	η_S(%)	$\mathcal{P}_S\eta_S/\eta$
40	10	57	40	**1.40**	25	**1.75**	20	**2.80**
256	16	86	75	**1.75**	60	**2.81**	43	**4.02**
512	32	87	78	**1.78**	63	**2.90**	46	**4.25**
1024	32	93	87	**1.87**	78	**3.33**	63	**5.43**
3072	32	98	95	**1.95**	91	**3.74**	84	**6.87**
6144	32	99	98	**1.98**	95	**3.86**	91	**7.38**

Table 3.2 Speedup of IEEE 802.16m turbo decoders with $\mathcal{P}_S = \{2, 4, 8\}$

			$\mathcal{P}_S = 2$		$\mathcal{P}_S = 4$		$\mathcal{P}_S = 8$	
N	**L**	η (%)	η_S (%)	$\mathcal{P}_S\eta_S/\eta$	η_S (%)	$\mathcal{P}_S\eta_S/\eta$	η_S (%)	$\mathcal{P}_S\eta_S/\eta$
24	8	48	32	**1.32**	23	**1.88**	18	**3.03**
144	12	81	68	**1.68**	51	**2.54**	35	**3.42**
256	16	86	75	**1.75**	60	**2.81**	43	**4.03**
512	16	92	86	**1.86**	75	**3.26**	60	**5.23**
1184	16	97	93	**1.93**	88	**3.63**	78	**6.45**
2400	16	98	97	**1.97**	93	**3.80**	88	**7.14**

3.2.2 Speedup and Performance

Each SISO decoder in this parallel architecture processes shorter sub-blocks, so the decoding time per half-iteration can be reduced, and the throughput can be increased. However, the operating efficiency is easily affected by the block size, and the speedup must take its variation into account. The value of τ_y is scaled down by a factor of $\mathcal{P}_S$, while τ_a, τ_b, and τ_x are unchanged. Then we redefine the operating efficiency of the parallel SISO decoder level as

$$\eta_S = \frac{\tau_y/\mathcal{P}_S}{\tau_a + \tau_b + \tau_x + \tau_y/\mathcal{P}_S} = \frac{\tau_y}{\mathcal{P}_S(\tau_a + \tau_b + \tau_x) + \tau_y}. \tag{3.18}$$

Owing to larger denominator, η_S is less than the original η. The total decoding time becomes $(2 \times \mathcal{I} \times \mathbf{N})/(\mathcal{P}_S \times \eta_S)$ cycles, and the throughput calculation changes into

$$\Theta_S = \frac{\mathcal{P}_S \times \phi \times \mathcal{F} \times \eta_S}{2 \times \mathcal{I}}. \tag{3.19}$$

The parallel SISO decoder level causes declining operating efficiency [57–59]. Tables 3.1 and 3.2 show the side effects on the parallel architecture for 3GPP LTE-Advanced and IEEE 802.16m turbo codes, whose typical schemes are $\mathcal{P}_S = 8$ and $\mathcal{P}_S = 4$ respectively. These data are estimated based on the implementation examples in Chap. 2. If $\mathbf{L}$ is less than $\mathbf{M}$, the precise value of $(\tau_a + \tau_b + \tau_x)$ is $(2\mathbf{L} + 10)$; otherwise, it is $(2\mathbf{M} + 10)$. The processes for all blocks inevitably

suffer the loss of efficiency. High parallelism will widen the gap between the actual speedup ($\mathcal{P}_S \times \eta_S/\eta$) and the expected speedup $\mathcal{P}_S$. Moreover, this problem becomes more serious as the designs deal with small blocks. For example, the cases with the smallest $\mathbf{N}$ and $\mathcal{P}_S \geq 4$ have very low efficiency in either application. Their $\mathbf{M}$'s are so short that the execution time is dominated by the unimproved part ($(\tau_y/\mathcal{P}_S) < (\tau_a + \tau_b + \tau_x)$). In fact, the situation occurs for most blocks at $\mathcal{P}_S \geq 8$. Only a few number of cases can get the near-ideal speedup from the parallel SISO decoder level. Despite the drawback, it is possible for the parallel designs to achieve the required throughput of any block size because the decoding processes for smaller blocks take fewer iterations.

The use of multiple SISO decoders for every single codeword block also has a negative influence over the error correction capability. Now the computations of one SISO decoder only involves $\mathbf{M}$ trellis stages. For the sub-blocks without known initial conditions, their forward metrics at the first trellis stage are set to zeroes. Because of the shortened trellis and rough initialization, the forward metrics are less accurate than those of a non-parallelized design. To lessen the impact, the designs update the forward metrics with the results at previous iterations [43,44]. The major hardware costs of this method are just a few buffers for additional $16\boldsymbol{\varrho}(\alpha(S_i)) \times \mathcal{P}_S$ bits. The calculation of backward metrics encounters similar problems. Some $\mathbf{M}$'s cannot be divisible by $\mathbf{L}$, and the last window in each sub-block has a length of $\mathbf{M}$ modulo $\mathbf{L}$. It might decrease the effectiveness of dummy backward metric calculation and fail to provide reliable initialization for adjacent windows. We can easily solve the problem by changing $\mathbf{L}$ and assuring sufficient length of the last window in each sub-block. Thus, there will be a need for the SISO decoder that supports configurable window length.

Figure 3.2 illustrates the BER performance of various combinations of $\{\mathcal{P}_S, \mathbf{L}\}$ in practical 3GPP LTE-Advanced and IEEE 802.16m turbo decoders. Each subfigure contains the fixed-point simulation results of six $\mathbf{N}$'s. Note that the typical designs for IEEE 802.16m standard must obey (3.7). Basically, the $\mathbf{L}$'s used in the simulations of $\mathcal{P}_S > 1$ and of $\mathcal{P}_S = 1$ are identical, but some cases, including $\mathbf{N} = \{136, 288\}$ in Fig. 3.2a and $\mathbf{N} = \{48, 144, 256, 512\}$ in Fig. 3.2b, cause significant degradation. After we adjust $\mathbf{L}$ to guarantee sufficient length of the last window, the performance losses of most cases are less than 0.1 dB. There are, however, a few exceptions. Only one choice of $\mathbf{L}$ is available for very small sub-blocks such as the example with $\mathbf{N} = 40$ and $\mathcal{P}_S = 8$ in Fig. 3.2a, and the adjustment of $\mathbf{L}$ is inapplicable here. Excluding the rare extreme cases, the designs with parallel SISO decoder level can accelerate the decoding process while maintaining comparable error correction capability.

3.2.3 *Hardware Cost*

Table 3.3 lists the synthesis results of 3GPP LTE-Advanced and IEEE 802.16m turbo decoders with high parallelism. This table also details the area of the main components of each design. It is easy to notice that the SISO decoders, whose area

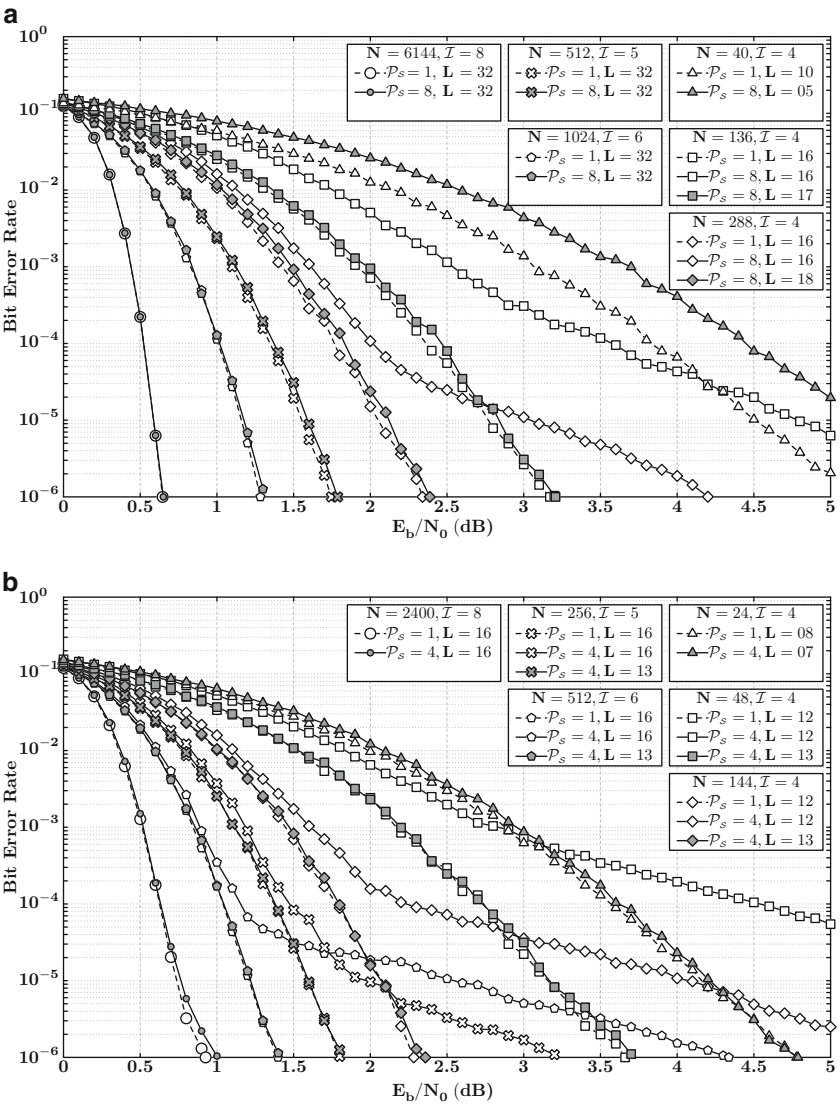

Fig. 3.2 Fixed-point simulation results of 3GPP LTE-Advanced and IEEE 802.16m turbo codes with various cases of $(\mathbf{N}, \mathcal{I}, \mathcal{P}_\mathcal{S}, \mathbf{L})$: Max-Log-MAP algorithm with $\zeta = 0.75$ and $(\varrho_\mathrm{I}(r_i), \varrho_\mathrm{F}(r_i)) = (3, 3)$. (**a**) 3GPP LTE-Advanced turbo codes: $\mathbf{N} = \{40, 136, 288, 512, 1024, 6144\}$, $\mathcal{P}_\mathcal{S} = \{1, 8\}$ with division mapping, and normalization with (2.15). (**b**) IEEE 802.16m turbo codes: $\mathbf{N} = \{24, 48, 144, 256, 512, 2400\}$, $\mathcal{P}_\mathcal{S} = \{1, 4\}$ with modulo mapping, and modulo normalization

Table 3.3 Area of turbo decoders using parallel SISO decoder level (unit: $10^3 \times \mu m^2$)

		3GPP LTE-Advanced			IEEE 802.16m		
		$\mathcal{P}_S = 8$	$\mathcal{P}_S = 16$	$\mathcal{P}_S = 32$	$\mathcal{P}_S = 4$	$\mathcal{P}_S = 8$	$\mathcal{P}_S = 16$
○	Turbo decoder	1774.4	3150.6	5769.2	1348.8	2385.4	4346.0
◇	SISO decoders	1120.6	2240.3	4469.0	813.8	1628.7	3255.1
◇	Memory	555.4	764.9	1039.4	450.3	614.6	799.2
◇	Memory controller	78.8	107.4	158.0	65.2	107.9	200.9
◇	Network	5.4	21.4	81.3	5.1	17.4	69.0

is in direct proportion to $\mathcal{P}_S$, dominate the overall hardware cost. However, the costs of the other three parts are difficult to estimate. Those designs for the same application store **N** sets of received data and extrinsic information, but the area of their memories are quite different. The reason is the use of small memory hard macros, where certain portions are power rings. As $\mathcal{P}_S$ increases, we need more separate memories to support parallel processing, and these power rings become a significant overhead. This problem can be resolved by merging the memories whose read and write addresses are always identical into single one with wider data width. The number of memory hard macros decreases, and so does the overhead of the power rings. Note that such a method will restrict the configurable $\mathcal{P}_S$, and there has to be a trade-off between flexibility and overhead. The area of memory controller depends on the how to get parallel QPP and ARP interleaving indexes. Since we use one and $\mathcal{P}_S$ address generators for calculating $\boldsymbol{j}_{s;j}^{\mathcal{Q}}$ and $\hat{\boldsymbol{j}}_{s;j}^{\mathcal{A}}$ respectively, the IEEE 802.16m turbo decoder usually has a larger memory controller even with lower parallelism. The wires of the interconnection network spread over the whole design, and they corresponds to the global routing. When $\mathcal{P}_S$ doubles, the network complexity will nearly quadruple. That is, the implementation of a highly parallel turbo decoder takes considerable effort.

3.3 Sophisticated Functional Units for Successive Trellis Stages

The design exploiting the parallel trellis stage level spends one cycle doing the computations that originally take $\mathcal{P}_T$ cycles in the conventional turbo decoder. Such parallel architecture is realized by rearranging the decoding algorithm as well as restructuring the circuits. Among all components, the modified functional units for forward and backward metrics receive the most attention for their close connection with the critical path delay [60–64]. In this section, the functional unit for forward metrics of the 3GPP LTE-Advanced turbo decoder with $\mathcal{P}_T = 2$ is used for illustration. With the forward metrics at the $(i-2)$-th trellis stage and the necessary branch metrics, it can calculate any $\alpha(S_i)$ in a clock cycle. There are two possible implementation methodologies. Figure 3.3a shows the functional unit of the first type, which is based on the concatenation of two successive trellis stages. Its executions can be represented as (3.20).

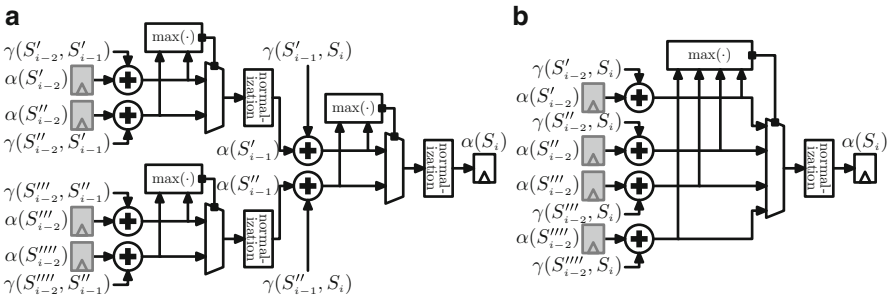

Fig. 3.3 Functional units for calculating $\alpha(S_i)$ in the design with $\mathcal{P}_\mathcal{T} = 2$. (**a**) Circuits for concatenated trellis. (**b**) Circuits for merged trellis

Table 3.4 Hardware cost and path delay of the ACS unit with ϕ and $\mathcal{P}_\mathcal{T}$

	Concatenated trellis		Merged trellis	
Hardware cost	$2^\phi \mathcal{P}_\mathcal{T} \times 8$	adders	$2^{\phi\mathcal{P}_\mathcal{T}} \times 8$	adders
	$\mathcal{P}_\mathcal{T} \times 8$	2^ϕ-input comparators	8	$2^{\phi\mathcal{P}_\mathcal{T}}$-input comparators
	$\mathcal{P}_\mathcal{T} \times 8$	2^ϕ-to-1 multiplexers	8	$2^{\phi\mathcal{P}_\mathcal{T}}$-to-1 multiplexers
	$\mathcal{P}_\mathcal{T} \times 8$	normalization circuits	8	normalization circuits
Path delay	$\mathcal{P}_\mathcal{T}$	addition	1	addition
	$\mathcal{P}_\mathcal{T}$	2^ϕ-input comparison	1	$2^{\phi\mathcal{P}_\mathcal{T}}$-input comparison
	$\mathcal{P}_\mathcal{T}$	2^ϕ-to-1 selection	1	$2^{\phi\mathcal{P}_\mathcal{T}}$-to-1 selection
	$\mathcal{P}_\mathcal{T}$	normalization	1	normalization

$$\alpha(S_i) = \max\Big[\max\big[\alpha(S'_{i-2}) + \gamma(S'_{i-2}, S'_{i-1}), \alpha(S''_{i-2}) + \gamma(S''_{i-2}, S'_{i-1})\big] + \gamma(S'_{i-1}, S_i),$$
$$\max\big[\alpha(S'''_{i-2}) + \gamma(S'''_{i-2}, S''_{i-1}), \alpha(S''''_{i-2}) + \gamma(S''''_{i-2}, S''_{i-1})\big] + \gamma(S''_{i-1}, S_i)\Big] \tag{3.20}$$

Figure 3.3b presents the functional unit of the second type, which is developed by merging two trellis stages into one. We need to redefine the branch metric as

$$\gamma(S_{i-\mathcal{P}_\mathcal{T}}, S_i) = \sum_{\ell=i-\mathcal{P}_\mathcal{T}}^{i-1} \gamma(S_\ell, S_{\ell+1}), \tag{3.21}$$

and complete the summation in advance to perform the calculation in (3.22).

$$\alpha(S_i) = \max\Big[\alpha(S'_{i-2}) + \gamma(S'_{i-2}, S_i), \alpha(S''_{i-2}) + \gamma(S''_{i-2}, S_i),$$
$$\alpha(S'''_{i-2}) + \gamma(S'''_{i-2}, S_i), \alpha(S''''_{i-2}) + \gamma(S''''_{i-2}, S_i)\Big]. \tag{3.22}$$

The circuits in Figs. 3.3a and 3.3b have common functionality, but their required resources and execution time are dissimilar. Table 3.4 gives these data of the functional units for all forward or backward metrics of eight states in an SISO decoder. The ACS unit for concatenated trellis structure needs $\mathcal{P}_\mathcal{T}$ times the hardware of a

conventional ACS unit. Its path delay is increased by $\mathcal{P}_\tau$ times, too. For the ACS unit that supports merged trellis structure, there is an exponential growth of the number of adders; the comparator and multiplexer must handle $2^{\phi\mathcal{P}_\tau}$ inputs; the normalization circuits are unaffected by $\mathcal{P}_\tau > 1$. Compared to the conventional case, this ACS unit uses longer execution time to perform more complex comparison and selection. The overall hardware cost and total path delay are determined by ϕ, $\mathcal{P}_\tau$, and the normalization method. If the modulo normalization is applied, the cost and delay related to normalization circuits can be ignored. While we utilize the cascaded structure, the $2^{\phi\mathcal{P}_\tau}$-input comparator/multiplexer is larger than $\mathcal{P}_\tau$ 2^ϕ-input comparators/multiplexers; on the other hand, the $2^{\phi\mathcal{P}_\tau}$-input comparison/selection takes similar execution time as $\mathcal{P}_\tau$ 2^ϕ-input comparison/selection. In general, the second type of ACS unit needs more area but has faster speed than the other one.

The extended data path lowers the maximum operating frequency, thereby diminishing the benefit of the parallel trellis stage level. The effect is particularly harmful in the ACS unit for concatenated trellis structure because the loss of frequency would negate the gain of $\mathcal{P}_\tau$. This type of ACS unit can bring about throughput improvement only when the obtainable frequency of the design still exceeds the highest clock rate of the clock generator. It is improbable to meet this condition as the parallelism is high. The problem is less severe in the ACS unit for merged trellis structure, but the dramatic increase in hardware cost obstructs the use of large $\mathcal{P}_\tau$. As a consequence, most designs prefer the second type of ACS unit with $\phi = 1$ and $\mathcal{P}_\tau = 2$ [60–64]. The works in [65,66] show the feasibility of SISO decoders with $\phi = 1$ and $\mathcal{P}_\tau = 4$. They exploit the *two-dimensional* technique, combining both types of circuits in Fig. 3.3, to reach a compromise between area and speed of the implementation; then they apply the *relocation* technique, rearranging the position of adders and registers in the functional units, to further shorten the data path. We can regard $\phi\mathcal{P}_\tau \leq 4$ as a restriction of using the parallel trellis stage level in practical designs. Hence, the allowable parallelism of 3GPP LTE-Advanced and IEEE 802.16m turbo decoders are $\mathcal{P}_\tau = 4$ and $\mathcal{P}_\tau = 2$ respectively.

This parallel design needs simultaneous access to $\mathcal{P}_\tau$ successive data every cycle, so the received data and extrinsic information of a codeword block are stored in multiple memory banks with the modulo mapping as (3.23), where the t-th memory bank keeps the data whose indexes are congruent to t modulo $\mathcal{P}_\tau$.

$$\{r_{t+k\mathcal{P}_\tau} \mid 0 \leq k < \mathbf{N}/\mathcal{P}_\tau\}. \tag{3.23}$$

This allocation ensures smooth memory access during the decoding rounds for the original data sequence, but it might create the collision problem during the decoding rounds for the permuted data sequence. Like the parallel SISO decoder level, the favorable solution is an interleaver with contention-free property. The definition of this property for QPP interleavers is that any $(t + k\mathcal{P}_\tau)$ must satisfy

$$\{\mathcal{Q}(t + k\mathcal{P}_\tau) \not\equiv \mathcal{Q}(t + k\mathcal{P}_\tau + \ell) \pmod{\mathcal{P}_\tau} \mid 0 < \ell < \mathcal{P}_\tau\}. \tag{3.24}$$

All interleavers of the 3GPP LTE-Advanced turbo code can pass the verification of (3.24) with $\mathcal{P}_\tau = 4$. The relations between $\mathcal{Q}(t + k\mathcal{P}_\tau)$ and its succeeding three interleaving indexes can be expressed as (3.25) with the help of the following constraints: $2 \nmid f_1$ and $2 \mid f_2$.

$$\begin{cases} \mathcal{Q}(t + k\mathcal{P}_\tau + 1) \equiv \mathcal{Q}(t + k\mathcal{P}_\tau) + f_1 + f_2 & (\text{mod } 4) \\ \mathcal{Q}(t + k\mathcal{P}_\tau + 2) \equiv \mathcal{Q}(t + k\mathcal{P}_\tau) + 2 & (\text{mod } 4) \\ \mathcal{Q}(t + k\mathcal{P}_\tau + 3) \equiv \mathcal{Q}(t + k\mathcal{P}_\tau) + f_1 + f_2 + 2 & (\text{mod } 4) \end{cases} \tag{3.25}$$

Since $(f_1 + f_2)$ modulo 4 is either 1 or 3, any two of the four successive indexes are incongruent modulo 4, and $\{\tilde{r}_{t+k\mathcal{P}_\tau}, \tilde{r}_{t+k\mathcal{P}_\tau+1}, \tilde{r}_{t+k\mathcal{P}_\tau+2}, \tilde{r}_{t+k\mathcal{P}_\tau+3}\}$ are stored in different memory banks. The parallelism can be either 2 or 4. The design needs $\mathcal{P}_\tau$ address generators to get the indexes concurrently. Each of them makes use of (2.3) and (2.5) with $\delta = \mathcal{P}_\tau$. The index $\mathcal{Q}(t + k\mathcal{P}_\tau)$ will be further translated into $(\boldsymbol{t}^{\mathcal{Q}}_{t;k} + \boldsymbol{k}^{\mathcal{Q}}_{t;k}\mathcal{P}_\tau)$ by (3.26) and (3.27), where $\boldsymbol{t}^{\mathcal{Q}}_{t;k}$ is the least significant $\log_2 \mathcal{P}_\tau$ bits of $\mathcal{Q}(t + k\mathcal{P}_\tau)$, and $\boldsymbol{k}^{\mathcal{Q}}_{t;k}$ is the most significant $(\lceil \log_2 \mathbf{N} \rceil - \log_2 \mathcal{P}_\tau)$ bits of $\mathcal{Q}(t + k\mathcal{P}_\tau)$. Then the controller can send the address $\boldsymbol{k}^{\mathcal{Q}}_{t;k}$ to the $\boldsymbol{t}^{\mathcal{Q}}_{t;k}$-th memory bank to access $\tilde{r}_{t+k\mathcal{P}_\tau}$.

$$\boldsymbol{t}^{\mathcal{Q}}_{t;k} = \mathcal{Q}(t + k\mathcal{P}_\tau) \bmod \mathcal{P}_\tau \tag{3.26}$$

$$\boldsymbol{k}^{\mathcal{Q}}_{t;k} = \lfloor \mathcal{Q}(t + k\mathcal{P}_\tau)/\mathcal{P}_\tau \rfloor \tag{3.27}$$

For IEEE 802.16m turbo codes, their ARP interleavers in (1.3) are designed to be compatible with the modulo mapping [20,56]; so (3.28) is always true whenever $\mathcal{P}_\tau$ is equal to or less than 4.

$$\{\mathcal{A}(t + k\mathcal{P}_\tau) \not\equiv \mathcal{A}(t + k\mathcal{P}_\tau + \ell) \pmod{\mathcal{P}_\tau} \mid 0 < \ell < \mathcal{P}_\tau\}. \tag{3.28}$$

Its controller is identical to that of the 3GPP LTE-Advanced turbo decoder using parallel trellis stage level except that the address generators are replaced by the circuits for executing (2.8) and (2.9) with $\delta = \mathcal{P}_\tau$. There is also a transition from $\mathcal{A}(t + k\mathcal{P}_\tau)$ to $(\boldsymbol{t}^{\mathcal{A}}_{t;k} + \boldsymbol{k}^{\mathcal{A}}_{t;k}\mathcal{P}_\tau)$ by (3.29) and (3.30). When $\mathcal{P}_\tau$ is a power of two, the control signals can be derived directly from the binary expression of $\mathcal{A}(t + k\mathcal{P}_\tau)$.

$$\boldsymbol{t}^{\mathcal{A}}_{t;k} = \mathcal{A}(t + k\mathcal{P}_\tau) \bmod \mathcal{P}_\tau \tag{3.29}$$

$$\boldsymbol{k}^{\mathcal{A}}_{t;k} = \lfloor \mathcal{A}(t + k\mathcal{P}_\tau)/\mathcal{P}_\tau \rfloor \tag{3.30}$$

After the turbo decoder exploits this parallel architecture, the execution time (τ_x and τ_y) are scaled down by a factor of $\mathcal{P}_\tau$, whereas the pipeline delay and memory access time (τ_a and τ_b) stay the same. The operating efficiency becomes

$$\eta_\tau = \frac{\tau_y/\mathcal{P}_\tau}{\tau_a + \tau_b + \tau_x/\mathcal{P}_\tau + \tau_y/\mathcal{P}_\tau} = \frac{\tau_y}{\mathcal{P}_\tau(\tau_a + \tau_b) + \tau_x + \tau_y}. \tag{3.31}$$

The shorter execution time decreases the operating efficiency. Now the decoding process takes $(2 \times \mathcal{I} \times \mathbf{N})/(\mathcal{P}_\mathcal{T} \times \eta_\mathcal{T})$ cycles. To avoid overestimating throughput, the variations in the critical path should be considered as well. We represent the maximum clock frequency of this design as $\mathcal{F}_\mathcal{T}$ and rewrite the equation for throughput calculation as

$$\Theta_\mathcal{T} = \frac{\mathcal{P}_\mathcal{T} \times \phi \times \mathcal{F}_\mathcal{T} \times \eta_\mathcal{T}}{2 \times \mathcal{I}}. \tag{3.32}$$

The parallel trellis stage level contributes a speedup of $(\mathcal{P}_\mathcal{T} \times \mathcal{F}_\mathcal{T} \times \eta_\mathcal{T})/(\mathcal{F} \times \eta)$. Because of the limited parallelism, the loss of efficiency is insignificant for large $\mathbf{N}$'s, and the discrepancy between the expected gain $\mathcal{P}_\mathcal{T}$ and the real speedup is dominated by $\mathcal{F}_\mathcal{T}/\mathcal{F}$. For small $\mathbf{N}$'s, even $\mathcal{P}_\mathcal{T} = 2$ could cause a sharp drop in operating efficiency, and the speedup would be far less than $\mathcal{P}_\mathcal{T}$. The potential risk is $(\mathcal{P}_\mathcal{T} \times \mathcal{F}_\mathcal{T} \times \eta_\mathcal{T}) \leq (\mathcal{F} \times \eta)$. This parallel architecture requires extreme caution in use, particularly when the design has to process small blocks.

3.4 Hybrid Parallel Architecture

These three types of parallel architecture have their respective advantages and disadvantages. Some of their features are different but complementary, so they can be combined together and become suitable to a variety of conditions. Such a design with hybrid parallel architecture contains $\mathcal{P}_\mathcal{C}$ separate groups of $\mathcal{P}_\mathcal{S}$ SISO decoders, each of which runs the computations related to $\mathcal{P}_\mathcal{T}$ successive trellis stages per cycle; and every group of SISO decoders deals with one codeword block. The overall throughput is

$$\Theta_\mathcal{H} = \frac{\mathcal{P}_\mathcal{H} \times \phi \times \mathcal{F}_\mathcal{H} \times \eta_\mathcal{H}}{2 \times \mathcal{I}}. \tag{3.33}$$

The values of the essential factors affected by the hybrid parallel architecture are given in (3.34)–(3.36). Its τ_y becomes $(\mathcal{P}_\mathcal{S} \times \mathcal{P}_\mathcal{T})$ smaller than original execution cycles, and the τ_x is reduced by a factor of $\mathcal{P}_\mathcal{T}$. For the cases with $\{\mathcal{P}_\mathcal{C}, \mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{T}\} > 1$, only $\mathcal{P}_\mathcal{H}$ increases, whereas both $\mathcal{F}_\mathcal{H}$ and $\eta_\mathcal{H}$ decrease.

$$\mathcal{P}_\mathcal{H} = \mathcal{P}_\mathcal{C} \times \mathcal{P}_\mathcal{S} \times \mathcal{P}_\mathcal{T} \tag{3.34}$$

$$\mathcal{F}_\mathcal{H} = \mathcal{F}_\mathcal{T} \tag{3.35}$$

$$\eta_\mathcal{H} = \frac{\tau_y/(\mathcal{P}_\mathcal{S} \times \mathcal{P}_\mathcal{T})}{\tau_a + \tau_b + \tau_x/\mathcal{P}_\mathcal{T} + \tau_y/(\mathcal{P}_\mathcal{S} \times \mathcal{P}_\mathcal{T})} \tag{3.36}$$

The product of $\mathcal{P}_\mathcal{C}$ and $\mathcal{P}_\mathcal{S}$ will be a constant number. It indicates how many SISO decoders and memory modules are used. The practical design can adjust $\mathcal{P}_\mathcal{C}$ and $\mathcal{P}_\mathcal{S}$ dynamically according to block size. While processing small blocks, it uses

higher $\mathcal{P}_C$ and lower $\mathcal{P}_S$ because the parallel turbo decoder level is harmless to operating efficiency. Conversely, the decoding flows for large blocks prefer lower $\mathcal{P}_C$ and higher $\mathcal{P}_S$ for the least cost in memory modules and the shorter latency. If there is a shortage of high-frequency clock signal, the parallel trellis stage level can help the design achieve better throughput.

The hybrid parallelism with $\mathcal{P}_S > 1$ and $\mathcal{P}_T > 1$ complicates the memory mapping. Each of the $\mathcal{P}_S$ memory modules consists of $\mathcal{P}_T$ memory banks. The basic constraints are $\mathcal{P}_S \mid \mathbf{N}$ and $\mathcal{P}_T \mid \mathbf{M}$. As long as $\mathcal{P}_T$ is divisible by 2, the $\mathbf{M}'$ in (3.7) is in breach of the second constraint. The parallel trellis stage level is incompatible with the design applying the parallel SISO decoder level with modulo mapping, so we only discuss the mixture of the division mapping in (3.2) and the modulo mapping in (3.23) here. The data r_i $(0 \le i < \mathbf{N})$ will be stored in the t-th memory bank of the s-th memory module under the transformation of (3.37); then the simultaneous access to $r_{s\mathbf{M}+t+k\mathcal{P}_T}$ for all possible pairs of (s,t) and a given k can be done easily.

$$\{r_i \Rightarrow r_{s\mathbf{M}+t+k\mathcal{P}_T} \mid s = \lfloor i/\mathbf{M} \rfloor, t = i \bmod \mathcal{P}_T, k = \lfloor (i \bmod \mathbf{M})/\mathcal{P}_T \rfloor\} \tag{3.37}$$

The hybrid parallel architecture is typically subject to more strict constraints on interleaving rules to circumvent the collision problem during the decoding rounds for the permuted data sequence. The restrictions of the two parallel levels and the prerequisite $\mathcal{P}_T \mid \mathbf{M}$ should be considered jointly. For 3GPP LTE-Advanced and IEEE 802.16m turbo codes, either $s \neq s'$ or $t \neq t'$ will fulfill at least one of (3.38) and (3.39), where $\mathbf{\Pi}(\cdot)$ stands for $\mathcal{Q}(\cdot)$ and $\mathcal{A}(\cdot)$. That is, every memory bank of every memory module always receives exactly one request while the design accesses $\{\tilde{r}_{s\mathbf{M}+t+k\mathcal{P}_T} \mid 0 \le s < \mathcal{P}_S; 0 \le t < \mathcal{P}_T\}$.

$$\lfloor \mathbf{\Pi}(s\mathbf{M} + t + k\mathcal{P}_T)/\mathbf{M} \rfloor \neq \lfloor \mathbf{\Pi}(s'\mathbf{M} + t' + k\mathcal{P}_T)/\mathbf{M} \rfloor \tag{3.38}$$

$$\mathbf{\Pi}(s\mathbf{M} + t + k\mathcal{P}_T) \not\equiv \mathbf{\Pi}(s'\mathbf{M} + t' + k\mathcal{P}_T) \pmod{\mathcal{P}_T} \tag{3.39}$$

The supportable $\mathcal{P}_T$ is either 2 or 4, and it leads to fewer choices of $\mathcal{P}_S$ at each block size. As a result, the objective throughput and the interleaver parameters determine which combination of these parallel levels is best.

3.5 State-of-the-Art Chip Implementation

The published turbo decoder chips with parallel architecture in [67–71] are aimed at 3GPP LTE standard [14] and/or IEEE 802.16e standard [16], whose revisions are 3GPP LTE-Advanced standard and IEEE 802.16m standard respectively. The 3GPP LTE turbo decoder in [67] outperforms the others in decoding speed. With the hybrid of $\mathcal{P}_S = 8$ and $\mathcal{P}_T = 2$, its throughput can reach 390 Mb/s. Both designs in [68] and [69] support all block sizes of 3GPP LTE turbo code. The former can reconfigure

the parallelism ($\mathcal{P}_S = \{1, 2, 4, 8\}$); the latter with $\mathcal{P}_S = 4$ can adaptively adjust the iteration number, clock frequency, and supply voltage. The two designs in [70] and [71] feature dual-mode architecture. Their decoders are regarded as the combination of $\mathcal{P}_S = 8$ and $\mathcal{P}_T = 2$ for 3GPP LTE application and as $\mathcal{P}_S = 8$ for IEEE 802.16e application. High throughput, great flexibility, and variable functionality are the major accomplishments of these works. Most of their techniques are still successful in 3GPP LTE-Advanced and IEEE 802.16m turbo decoders, and higher parallelism ($\mathcal{P}_S \geq 8$) will be the direct method toward faster speed. It implies the increasing complexity and decreasing efficiency. The growing importance of their solutions is especially apparent in the decoder chip implementation for next-generation telecommunication systems.

Chapter 4
Low-Complexity Solution for Highly Parallel Architecture

The complexity of highly parallel architecture depends on the parallelism, the area of a single SISO decoder, and the apparatus for parallel data transmission. Chapter 2 and 3 have given the guidelines of choosing proper processing schedule and normalization method of the SISO decoder. Thus, the focus of this chapter is on the circuits that interconnect SISO decoders and memory modules. The trivial apparatus is the fully-connected network. It can offer arbitrary one-to-one interconnection patterns. However, its area overhead escalates rapidly as $\mathcal{P}_S$ increases, and the routing congestion would be another crucial design issue. Many research works have developed alternative solutions[72–75]. The networks in either [72] or [73] can support any interleaver, while those in [74] and [75] are designed for specific interleavers. In general, using application-specific networks to handle data transmission between multiple sources and multiple destinations requires simpler controllers and takes fewer clock cycles. For 3GPP LTE-Advanced and IEEE 802.16m turbo decoders, such type of interconnection is preferable. Here we will introduce the *multistage* network in [75] for the parallel design with QPP interleavers. Despite some restrictions on the parameters, it is applicable to all block sizes of the 3GPP LTE-Advanced turbo codes. We then present the apparatus, founded on the same principle as [75], for the parallel design with ARP interleavers. Both the cases with modulo mapping and division mapping will be mentioned. For simplicity, our discussions only take the case that $\mathcal{P}_S$ is a power of two into consideration. In addition to the interconnection, this chapter illustrates how to compensate performance in the highly parallel architecture without changing window length. It avoids the search of appropriate $\mathbf{L}$ for each $(\mathbf{N}, \mathcal{P}_S)$ and makes the parallel architecture easier to use.

C.-C. Wong and H.-C. Chang, *Turbo Decoder Architecture for Beyond-4G Applications*, DOI 10.1007/978-1-4614-8310-6_4,

4.1 Interconnection for Parallel Design with QPP Interleavers

The multistage network for the parallel design with QPP interleavers functions like a barrel shifter, so we also call it barrel-shift network. Figure 4.1 shows the networks of interest, including the most powerful apparatus and the ingenious solution. Extensions of these networks to other parallelism is straightforward. The fully-connected network consists of $\mathcal{P}_S \times \mathcal{P}_S$ data links. It needs $\mathcal{P}_S$ $\mathcal{P}_S$-to-1 multiplexers, of which the s-th multiplexer uses $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ in (3.4) as its selection signal. In a barrel-shift network, the total number of data links is $2\mathcal{P}_S \times \log_2 \mathcal{P}_S$. There are $\mathcal{P}_S \times \log_2 \mathcal{P}_S$ basic 2-to-1 multiplexers. At the ℓ-th stage ($0 \le \ell < \log_2 \mathcal{P}_S$), the transmitted data will be shifted upward by either 0 or 2^ℓ positions; besides, the s-th and the $(s + 2^\ell)$-th multiplexers have to share the same 1-bit selection signal to make sure all data can be relayed to the next stage or destinations. The control signals for the whole network add up to $(\mathcal{P}_S - 1)$ bits. Clearly, the simpler barrel-shift network will facilitate the implementation of a highly parallel turbo decoder.

The main task of the barrel-shift network is to establish the interconnection between the $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$-th memory module and the s-th SISO decoder. For each pair of $(s, \boldsymbol{s}^{\mathcal{Q}}_{s;j})$, the difference between the two indexes is indispensable for controlling data transmission. We define this displacement value as

$$\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j} = (\boldsymbol{s}^{\mathcal{Q}}_{s;j} - s) \bmod \mathcal{P}_S. \tag{4.1}$$

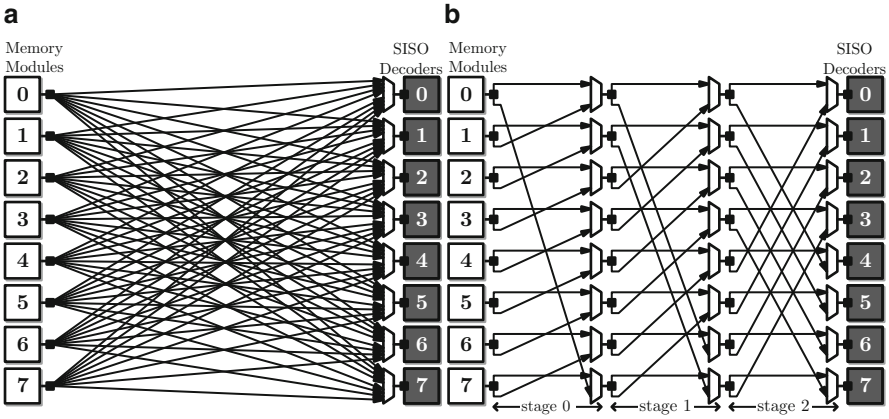

Fig. 4.1 Interconnection networks for the design with $\mathcal{P}_S = 8$. (**a**) Fully-connected network. (**b**) Barrel-shift network

The following proposition that states the relations among all offset values is the basis for the verification of usability of barrel-shift networks and for the generation of selection signals for the component multiplexers.

Proposition 4.1. *If the QPP interleaver parameters satisfy* $\mathcal{P}_S \mid \mathbf{N}$, $2 \nmid f_1$, *and* $2 \mid f_2$, *then* $\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j}$ *is congruent to* $\Delta \boldsymbol{s}^{\mathcal{Q}}_{s+2^\ell;j}$ *modulo* $2^{\ell+1}$ *for* $0 \le \ell < \log_2 \mathcal{P}_S$.

Proof. (a) From (3.5), the interleaved index $\boldsymbol{j}^{\mathcal{Q}}_{s;j}$ can be calculated by subtracting a multiple of $\mathbf{M}$ from $\mathcal{Q}(j)$:

$$\boldsymbol{j}^{\mathcal{Q}}_{s;j} = f_1 j + f_2 j^2 - \boldsymbol{\kappa}\mathbf{M}, \tag{4.2}$$

where the integer $\boldsymbol{\kappa}$ is independent of s.

(b) On replacing $\boldsymbol{j}^{\mathcal{Q}}_{s;j}$ by (4.2) in (3.3), we obtain $\boldsymbol{s}^{\mathcal{Q}}_{s;j}\mathbf{M}$ as (4.3). Then we divide both sides by $\mathbf{M}$ to get (4.4).

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j}\mathbf{M} = f_1 s\mathbf{M} + f_2 s^2\mathbf{M}^2 + 2f_2 s\mathbf{M}j + \boldsymbol{\kappa}\mathbf{M} \pmod{\mathbf{N}} \tag{4.3}$$

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} = f_1 s + f_2 s^2\mathbf{M} + 2f_2 sj + \boldsymbol{\kappa} \pmod{\mathcal{P}_S} \tag{4.4}$$

(c) The subsequent step is rewriting $\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j}$ in (4.1) with $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ in (4.4):

$$\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j} = (f_1 - 1)s + f_2 s^2\mathbf{M} + 2f_2 sj + \boldsymbol{\kappa} \pmod{\mathcal{P}_S}. \tag{4.5}$$

(d) After the substitution of $(s + 2^\ell)$ for s in (4.5) and some adjustments, $\Delta \boldsymbol{s}^{\mathcal{Q}}_{s+2^\ell;j}$ can be simplified to (4.6), where $\boldsymbol{\lambda}$ is an integer.

$$\begin{aligned}\Delta \boldsymbol{s}^{\mathcal{Q}}_{s+2^\ell;j} &= (f_1 - 1)(s + 2^\ell) + f_2(s + 2^\ell)^2\mathbf{M} + 2f_2(s + 2^\ell)j + \boldsymbol{\kappa} \\ &= \Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j} + 2^\ell \underbrace{[(f_1 - 1) + f_2(2^\ell + 2s)\mathbf{M} + 2f_2 j]}_{\boldsymbol{\lambda}} \\ &= \Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j} + 2^\ell \boldsymbol{\lambda} \pmod{\mathcal{P}_S}\end{aligned} \tag{4.6}$$

(e) Because of $2 \mid (f_1 - 1)$ and $2 \mid f_2$, the $\boldsymbol{\lambda}$ is always an even number, and $2^\ell\boldsymbol{\lambda}$ is divisible by $2^{\ell+1}$. With the precondition of $\mathcal{P}_S$, $2^{\ell+1} \mid \mathcal{P}_S$ is true for $0 \le \ell < \log_2 \mathcal{P}_S$. Consequently, we have

$$\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j} \equiv \Delta \boldsymbol{s}^{\mathcal{Q}}_{s+2^\ell;j} \pmod{2^{\ell+1}}. \tag{4.7}$$

■

In fact, the $\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j}$ modulo $2^{\ell+1}$ means the displacement of data in the first ℓ stages of a barrel-shift network. The relation in (4.7) completely fits the behavior of the proposed apparatus. We let $\Delta \boldsymbol{s}^{[\ell]}_{s;j}$ be the ℓ-th least significant bit of $\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j}$. It can be directly used to control the s-th multiplexer at the ℓ-th stage. Due to (4.7), $\{\Delta \boldsymbol{s}^{\mathcal{Q}}_{s;j} \mid 0 \le s < \mathcal{P}_S/2\}$ is sufficient to control all multiplexers. The 188 sets of (f_1, f_2) of 3GPP LTE-Advanced standard can fulfill the constraints in the

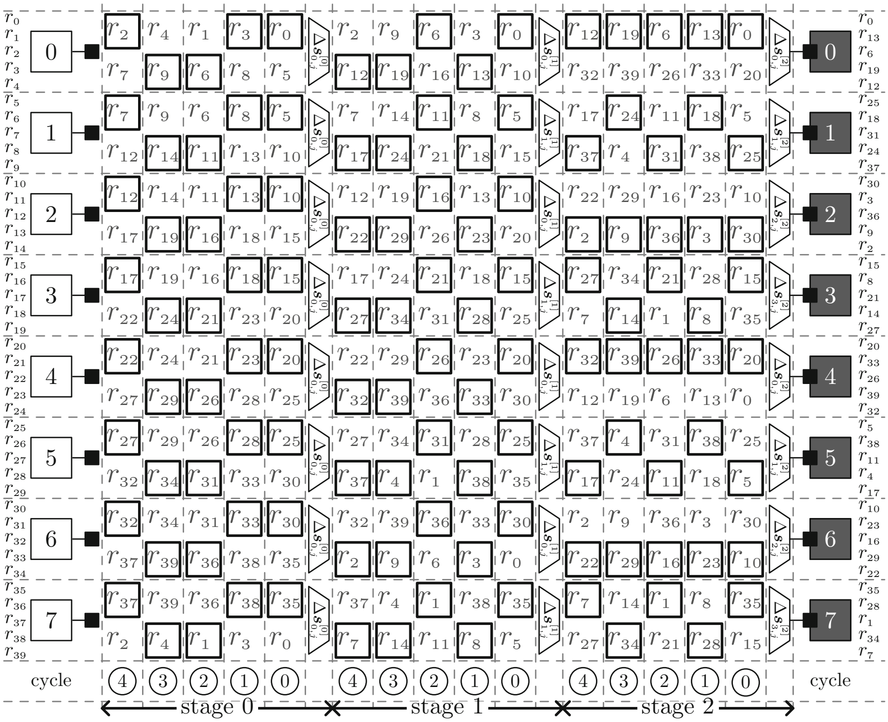

Fig. 4.2 Parallel data transmission at the decoding rounds for permuted sequence: the 3GPP LTE-Advanced turbo code with $\mathbf{N} = 40$, $\mathcal{P}_{\mathcal{S}} = 8$, and division mapping $((f_1, f_2) = (3, 10))$

Table 4.1 Interleaving indexes and displacement values of the 3GPP LTE-Advanced turbo code with $\mathbf{N} = 40$, $\mathcal{P}_{\mathcal{S}} = 8$, and division mapping

	$j = 0$			$j = 1$			$j = 2$			$j = 3$			$j = 4$		
	$\boldsymbol{s}^{\mathcal{Q}}_{s:j}$,	$\boldsymbol{j}^{\mathcal{Q}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{Q}}_{s:j}$	$\boldsymbol{s}^{\mathcal{Q}}_{s:j}$,	$\boldsymbol{j}^{\mathcal{Q}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{Q}}_{s:j}$	$\boldsymbol{s}^{\mathcal{Q}}_{s:j}$,	$\boldsymbol{j}^{\mathcal{Q}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{Q}}_{s:j}$	$\boldsymbol{s}^{\mathcal{Q}}_{s:j}$,	$\boldsymbol{j}^{\mathcal{Q}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{Q}}_{s:j}$	$\boldsymbol{s}^{\mathcal{Q}}_{s:j}$,	$\boldsymbol{j}^{\mathcal{Q}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{Q}}_{s:j}$
$s = 0$	0 ,	0	0	2 ,	3	2	1 ,	1	1	3 ,	4	3	2 ,	2	2
$s = 1$	5 ,	0	4	3 ,	3	2	6 ,	1	5	4 ,	4	3	7 ,	2	6
$s = 2$	6 ,	0	4	0 ,	3	6	7 ,	1	5	1 ,	4	7	0 ,	2	6
$s = 3$	3 ,	0	0	1 ,	3	6	4 ,	1	1	2 ,	4	7	5 ,	2	2
$s = 4$	4 ,	0	0	6 ,	3	2	5 ,	1	1	7 ,	4	3	6 ,	2	2
$s = 5$	1 ,	0	4	7 ,	3	2	2 ,	1	5	0 ,	4	3	3 ,	2	6
$s = 6$	2 ,	0	4	4 ,	3	6	3 ,	1	5	5 ,	4	7	4 ,	2	6
$s = 7$	7 ,	0	0	5 ,	3	6	0 ,	1	1	6 ,	4	7	1 ,	2	2

above proposition, so the turbo decoder can utilize this barrel-shift network for its parallel data transmission. Figure 4.2 gives an example of parallel data transmission in the 3GPP LTE-Advanced turbo decoder using division mapping. This figure shows which input of each multiplexer is selected at each cycle. For the design with $\mathcal{P}_{\mathcal{S}} = 8$, the selection signals include $\{\Delta\boldsymbol{s}^{[0]}_{0;j}\}$ at the first stage, $\{\Delta\boldsymbol{s}^{[1]}_{0;j}, \Delta\boldsymbol{s}^{[1]}_{1;j}\}$ at the second stage, and $\{\Delta\boldsymbol{s}^{[2]}_{0;j}, \Delta\boldsymbol{s}^{[2]}_{1;j}, \Delta\boldsymbol{s}^{[2]}_{2;j}, \Delta\boldsymbol{s}^{[2]}_{3;j}\}$ at the third stage. Table 4.1 lists

$\boldsymbol{s}^{\circlearrowleft}_{s;j}$, $\boldsymbol{j}^{\circlearrowleft}_{s;j}$, and $\Delta\boldsymbol{s}^{\circlearrowleft}_{s;j}$ of all combinations of (s, j) for easy reference. In the proof of Proposition 4.1, the equation (4.6) reveals another way to obtain all selection signals. Rather than calculating $\{\Delta\boldsymbol{s}^{\circlearrowleft}_{s;j} \mid 0 \le s < \mathcal{P}_S/2\}$ by (3.4), the controller first finds $\Delta\boldsymbol{s}^{\circlearrowleft}_{0;j}$ and then adds an offset value to it. The corresponding calculation involves merely $\log_2 \mathcal{P}_S$ bits and can be done quickly.

4.2 Interconnection for Parallel Design with ARP Interleavers

4.2.1 Parallel Architecture Using Modulo Mapping

The property like (4.7) is the key to the usability of barrel-shift networks. When the turbo decoder with ARP interleavers stores data by the modulo mapping as (3.6), the s-th SISO decoder will access data from the $\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$-th memory module during the decoding rounds for permuted data sequence, and the definition of displacement becomes

$$\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} = (\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} - s) \bmod \mathcal{P}_S. \tag{4.8}$$

For this case, its parallelism must comply with $4 \mid \mathcal{P}_S$ and $\mathcal{P}_S \mid \mathbf{N}$; and the sub-block size $\mathbf{M}'$ in (3.7) is always an odd number. The following proposition introduces the required parameters for the desired property.

Proposition 4.2. *If the ARP interleaver parameters satisfy*

$$\begin{cases} 2 \nmid \varepsilon & \text{(4.9a)} \\ 2 \mid (g_{(s\mathbf{M}'+j\pm1) \bmod 4} - g_{(s\mathbf{M}'+j) \bmod 4}) & \text{(4.9b)} \\ 4 \mid (g_{(s\mathbf{M}'+j\pm2) \bmod 4} - g_{(s\mathbf{M}'+j) \bmod 4}), & \text{(4.9c)} \end{cases}$$

then $\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$ *is congruent to* $\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s+2^\ell;j}$ *modulo* $2^{\ell+1}$ *for* $0 \le \ell < \log_2 \mathcal{P}_S$.

Proof. (a) After the expansion of (3.10), $\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$ can be rewritten as

$$\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} = \varepsilon s\mathbf{M}' + \varepsilon j + g_{(s\mathbf{M}'+j) \bmod 4} \pmod{\mathcal{P}_S}. \tag{4.10}$$

(b) On replacing $\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$ with (4.10) in (4.8), we get

$$\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} = (\varepsilon\mathbf{M}' - 1)s + \varepsilon j + g_{(s\mathbf{M}'+j) \bmod 4} \pmod{\mathcal{P}_S}. \tag{4.11}$$

(c) We substitute $(s + 2^\ell)$ for s in (4.11), modify the equation, and express $\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s+2^\ell;j}$ as the summation of $\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$ and other terms.

$$\begin{aligned}\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s+2^{\ell};j} &= (s+2^{\ell})(\varepsilon\mathbf{M}'-1)+\varepsilon j+g_{(s\mathbf{M}'+2^{\ell}\mathbf{M}'+j) \bmod 4}\\ &= \Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}+2^{\ell}(\varepsilon\mathbf{M}'-1)+\underbrace{g_{(s\mathbf{M}'+2^{\ell}\mathbf{M}'+j) \bmod 4}-g_{(s\mathbf{M}'+j) \bmod 4}}_{\boldsymbol{\lambda}'}\\ &= \Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}+2^{\ell}(\varepsilon\mathbf{M}'-1)+\boldsymbol{\lambda}' \pmod{\mathcal{P}_{\mathcal{S}}} \end{aligned} \tag{4.12}$$

(d) The constraints $2 \nmid \varepsilon$ and $2 \nmid \mathbf{M}'$ lead to $2 \nmid (\varepsilon\mathbf{M}')$; that is, $(\varepsilon\mathbf{M}'-1)$ is a multiple of 2. Hence, $2^{\ell}(\varepsilon\mathbf{M}'-1)$ is divisible by $2^{\ell+1}$.

(e) Each constituent term in $\boldsymbol{\lambda}'$ is one of $\{g_0, g_1, g_2, g_3\}$. The result of $\boldsymbol{\lambda}'$ relies on $2^{\ell}\mathbf{M}'$ modulo 4. Because of $2 \nmid \mathbf{M}'$, $\mathbf{M}'$ is congruent to either 1 or -1 modulo 4, and we have

$$2^{\ell}\mathbf{M}' \equiv \pm 2^{\ell} \pmod 4.$$

There are three possible results of $\boldsymbol{\lambda}'$:

$$\boldsymbol{\lambda}' = \begin{cases} g_{(s\mathbf{M}'+j\pm 1) \bmod 4}-g_{(s\mathbf{M}'+j) \bmod 4} & \text{if } \ell = 0,\\ g_{(s\mathbf{M}'+j\pm 2) \bmod 4}-g_{(s\mathbf{M}'+j) \bmod 4} & \text{if } \ell = 1,\\ 0 & \text{if } \ell \geq 2. \end{cases}$$

For $\ell \geq 2$, the value of $2^{\ell}\mathbf{M}'$ modulo 4 is 0; therefore, $\boldsymbol{\lambda}'$ is 0 and can be omitted from (4.12). The constraints (4.9b) and (4.9c) promise that $2^{\ell+1} \mid \boldsymbol{\lambda}'$ can hold true for $\ell = 0$ and $\ell = 1$ respectively.

(f) With $2^{\ell+1} \mid (2^{\ell}(\varepsilon\mathbf{M}'-1)+\boldsymbol{\lambda}')$ and $2^{\ell+1} \mid \mathcal{P}_{\mathcal{S}}$, the relation in (4.13) can be derived from (4.12).

$$\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j} \equiv \Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s+2^{\ell};j} \pmod{2^{\ell+1}} \tag{4.13}$$

■

For IEEE 802.16m turbo codes, all 39 sets of $(\varepsilon, g_0, g_1, g_2, g_3)$ can fulfill these constraints in Proposition 4.2. Notice that (4.9b) requires the examination of $\{|g_0-g_1|, |g_1-g_2|, |g_2-g_3|, |g_3-g_0|\}$; whereas (4.9c) necessitates the check on $\{|g_0-g_2|, |g_1-g_3|\}$. Figure 4.3 illustrates how to transmit data from multiple memory modules with module mapping to multiple SISO decoders for $\mathbf{N} = 24$ and $\mathcal{P}_{\mathcal{S}} = 4$. Here each multiplexer is labeled with its selection signal $\Delta\hat{\boldsymbol{s}}^{[\ell]}_{s;j}$, the ℓ-th bit (from right) of $\Delta\hat{\boldsymbol{s}}^{\mathcal{A}}_{s;j}$. Table 4.2 details the indexes and displacement of this example. Unlike division mapping, the modulo mapping also complicates the parallel data transmission of original data sequence. However, we can regard it as a special case of Proposition 4.2, whose ε is 1 and all of $\{g_0, g_1, g_2, g_3\}$ are zeroes. The barrel-shift network is therefore sufficient to support such parallel architecture at every decoding round.

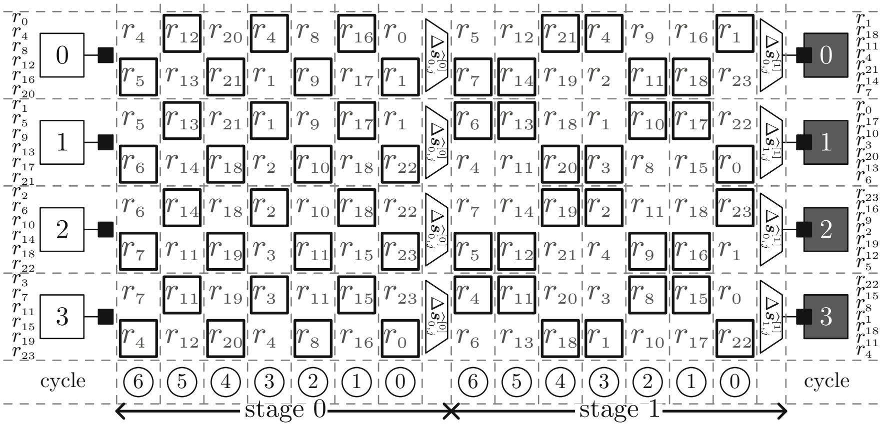

Fig. 4.3 Parallel data transmission at the decoding rounds for permuted sequence: the IEEE 802.16m turbo code with $\mathbf{N} = 24$, $\mathcal{P}_{\mathcal{S}} = 4$, and modulo mapping $((\varepsilon, g_0, g_1, g_2, g_3) = (5, 1, 13, 1, 13))$

Table 4.2 Interleaving indexes and displacement values of the IEEE 802.16m turbo code with $\mathbf{N} = 24$, $\mathcal{P}_{\mathcal{S}} = 4$, and modulo mapping

	$j=0$		$j=1$		$j=2$		$j=3$		$j=4$		$j=5$		$j=6$	
	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$	$\hat{s}^{\mathcal{A}}_{s;j}, \hat{j}^{\mathcal{A}}_{s;j}$	$\Delta\hat{s}^{\mathcal{A}}_{s;j}$
$s=0$	1,0	1	2,4	2	3,2	3	0,1	0	1,5	1	2,3	2	3,1	3
$s=1$	0,0	3	1,4	0	2,2	1	3,0	2	0,5	3	1,3	0	2,1	1
$s=2$	3,5	1	0,4	2	1,2	3	2,0	0	3,4	1	0,3	2	1,1	3
$s=3$	2,5	3	3,3	0	0,2	1	1,0	2	2,4	3	3,2	0	0,1	1

4.2.2 Parallel Architecture Using Division Mapping

With the same procedure, we can verify whether the barrel-shift network is applicable to the design with ARP interleavers and division mapping. Now the displacement is redefined as

$$\Delta \boldsymbol{s}^{\mathcal{A}}_{s;j} = (\boldsymbol{s}^{\mathcal{A}}_{s;j} - s) \bmod \mathcal{P}_{\mathcal{S}}. \tag{4.14}$$

The corresponding requirements is given in the following proposition.

Proposition 4.3. *If the ARP interleaver parameters satisfy*

$$\begin{cases} 2 \nmid \varepsilon & \text{(4.15a)} \\ g_{(s\mathbf{M}+j) \bmod 4} \equiv g_{j \bmod 4} \pmod{\mathbf{M}} & \text{(4.15b)} \end{cases}$$

then $\Delta \boldsymbol{s}^{\mathcal{A}}_{s;j}$ *is congruent to* $\Delta \boldsymbol{s}^{\mathcal{A}}_{s+2^{\ell};j}$ *modulo* $2^{\ell+1}$ *for* $0 \le \ell < \log_2 \mathcal{P}_{\mathcal{S}}$.

Proof. (a) From (3.15), the interleaved index $\boldsymbol{j}^{\mathcal{A}}_{s;j}$ can be calculated by subtracting a multiple of $\mathbf{M}$ from $\mathcal{A}(j)$:

$$\boldsymbol{j}^{\mathcal{A}}_{s;j} = \varepsilon j + g_{(s\mathbf{M}+j) \bmod 4} - \boldsymbol{\kappa}'\mathbf{M}. \tag{4.16}$$

Because of (4.15b), $\{\boldsymbol{j}^{\mathcal{A}}_{0;j}, \boldsymbol{j}^{\mathcal{A}}_{1;j}, \ldots, \boldsymbol{j}^{\mathcal{A}}_{\mathcal{P}_S-1;j}\}$ are the same, and the integer $\boldsymbol{\kappa}'$ is independent of s.

(b) On replacing $\boldsymbol{j}^{\mathcal{A}}_{s;j}$ by (4.16) in (3.13), we obtain $\boldsymbol{s}^{\mathcal{A}}_{s;j}\mathbf{M}$ as (4.17). Then we get a new expression of $\boldsymbol{s}^{\mathcal{A}}_{s;j}$ by dividing both sides of (4.17) by $\mathbf{M}$.

$$\boldsymbol{s}^{\mathcal{A}}_{s;j}\mathbf{M} = \varepsilon s\mathbf{M} + \boldsymbol{\kappa}'\mathbf{M} \pmod{\mathbf{N}} \tag{4.17}$$

$$\boldsymbol{s}^{\mathcal{A}}_{s;j} = \varepsilon s + \boldsymbol{\kappa}' \pmod{\mathcal{P}_S} \tag{4.18}$$

(c) The subsequent step is rewriting $\Delta\boldsymbol{s}^{\mathcal{A}}_{s;j}$ in (4.14) with $\boldsymbol{s}^{\mathcal{A}}_{s;j}$ in (4.18):

$$\Delta\boldsymbol{s}^{\mathcal{A}}_{s;j} = (\varepsilon - 1)s + \boldsymbol{\kappa}' \pmod{\mathcal{P}_S}. \tag{4.19}$$

(d) We substitute $(s + 2^\ell)$ for s in (4.19) and then simplify it to (4.20).

$$\begin{aligned}\Delta\boldsymbol{s}^{\mathcal{A}}_{s+2^\ell;j} &= (\varepsilon - 1)(s + 2^\ell) + \boldsymbol{\kappa}' \\ &= \Delta\boldsymbol{s}^{\mathcal{A}}_{s;j} + 2^\ell(\varepsilon - 1) \pmod{\mathcal{P}_S}\end{aligned} \tag{4.20}$$

(e) The constraint (4.15a) implies that 2 is a factor of $(\varepsilon - 1)$, and $2^\ell(\varepsilon - 1)$ is divisible by $2^{\ell+1}$. Moreover, $\mathcal{P}_S$ can be divided by $2^{\ell+1}$ for $0 \le \ell < \log_2 \mathcal{P}_S$. Consequently, we have.

$$\Delta\boldsymbol{s}^{\mathcal{A}}_{s;j} \equiv \Delta\boldsymbol{s}^{\mathcal{A}}_{s+2^\ell;j} \pmod{2^{\ell+1}}. \tag{4.21}$$

■

The constraint (4.15b) is identical with (3.12). Hence, the parameter sets of IEEE 802.16m turbo codes that support division mapping can definitely allow for the use of barrel-shift networks. Figure 4.4 provides an example of the case with $\mathbf{N} = 24$ and $\mathcal{P}_S = 4$, and Table 4.3 gives the essential information about this parallel data transmission. The ℓ-th bit of $\Delta\boldsymbol{s}^{\mathcal{A}}_{s;j}$ is symbolized as $\Delta\check{\boldsymbol{s}}^{[\ell]}_{s;j}$ here. Since (4.21) indicates that both the s-th and the $(s + 2^\ell)$-th multiplexers at the ℓ-th stage are controlled by $\Delta\check{\boldsymbol{s}}^{[\ell]}_{s;j}$ for $0 \le \ell < \log_2 \mathcal{P}_S$, the design in this example uses $\{\Delta\check{\boldsymbol{s}}^{[0]}_{0;j}, \Delta\check{\boldsymbol{s}}^{[1]}_{0;j}, \Delta\check{\boldsymbol{s}}^{[1]}_{1;j}\}$ for all multiplexers. After calculating $\Delta\boldsymbol{s}_{0;j}$ with (3.14), we can employ (4.20) to get the other displacement values. The offset $2^\ell(\varepsilon - 1)$ in this equation is a constant. Thanks to this feature, the generation of selection signals becomes much simpler.

Figures 4.3 and 4.4 consider the shortest block of IEEE 802.16m turbo codes under the same parallelism but distinct memory mapping. Although they store and transmit data in different manners, there are lots of similarities in their

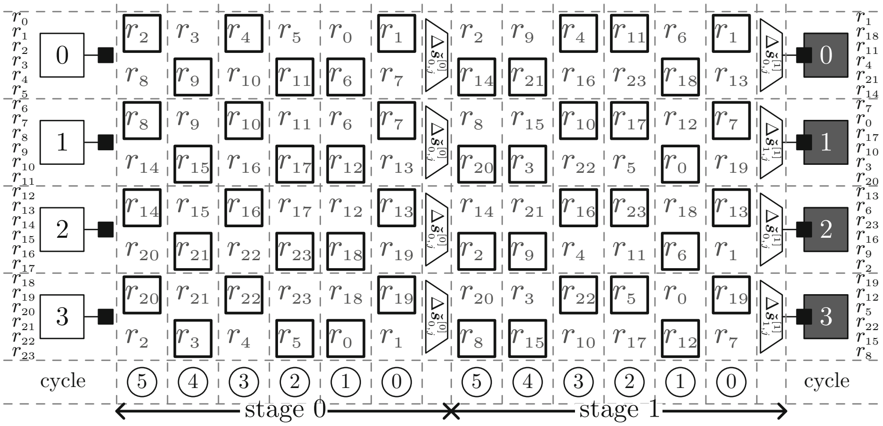

Fig. 4.4 Parallel data transmission at the decoding rounds for permuted sequence: the IEEE 802.16m turbo code with $\mathbf{N} = 24$, $\mathcal{P}_{\mathcal{S}} = 4$, and division mapping $((\varepsilon, g_0, g_1, g_2, g_3) = (5, 1, 13, 1, 13))$

Table 4.3 Interleaving indexes and displacement values of the IEEE 802.16m turbo code with $\mathbf{N} = 24$, $\mathcal{P}_{\mathcal{S}} = 4$, and division mapping

	$j = 0$		$j = 1$		$j = 2$		$j = 3$		$j = 4$		$j = 5$	
	$\boldsymbol{s}^{\mathcal{A}}_{s:j}, \boldsymbol{j}^{\mathcal{A}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{A}}_{s:j}$	$\boldsymbol{s}^{\mathcal{A}}_{s:j}, \boldsymbol{j}^{\mathcal{A}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{A}}_{s:j}$	$\boldsymbol{s}^{\mathcal{A}}_{s:j}, \boldsymbol{j}^{\mathcal{A}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{A}}_{s:j}$	$\boldsymbol{s}^{\mathcal{A}}_{s:j}, \boldsymbol{j}^{\mathcal{A}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{A}}_{s:j}$	$\boldsymbol{s}^{\mathcal{A}}_{s:j}, \boldsymbol{j}^{\mathcal{A}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{A}}_{s:j}$	$\boldsymbol{s}^{\mathcal{A}}_{s:j}, \boldsymbol{j}^{\mathcal{A}}_{s:j}$	$\Delta\boldsymbol{s}^{\mathcal{A}}_{s:j}$
$s = 0$	0 , 1	0	3 , 0	3	1 , 5	1	0 , 4	0	3 , 3	3	2 , 2	2
$s = 1$	1 , 1	0	0 , 0	3	2 , 5	1	1 , 4	0	0 , 3	3	3 , 2	2
$s = 2$	2 , 1	0	1 , 0	3	3 , 5	1	2 , 4	0	1 , 3	3	0 , 2	2
$s = 3$	3 , 1	0	2 , 0	3	0 , 5	1	3 , 4	0	2 , 3	3	1 , 2	2

core component circuits, including the barrel-shift network. By integration of the controllers over memory modules and barrel-shift networks, the IEEE 802.16m turbo decoder that supports both types of memory mappings is feasible.

4.3 Performance Compensation for Parallel Design

The less reliable path metric initialization of smaller sub-blocks is the major cause of inferior performance of the turbo decoder with $\mathcal{P}_{\mathcal{S}} > 1$. In Chap. 3, we utilize the $\alpha(S_{s\mathbf{M}})$ at previous iterations to reinforce forward metric calculation and change $\mathbf{L}$ to enhance backward metric calculation. This method is successful for almost all cases. However, the proper $\mathbf{L}$ varies with different $\mathbf{M}$'s, and its value must be found in advance. To minimize such effort, this section presents another methodical approach which permits consistent window length in any parallelism. It could be viewed as a full hybrid of the dummy metric calculation in [42] and the compensation from previous metrics in [43, 44] because the parallel design makes use of both forward and backward metrics at the boundary of every sub-block. Figure 4.5 shows how the

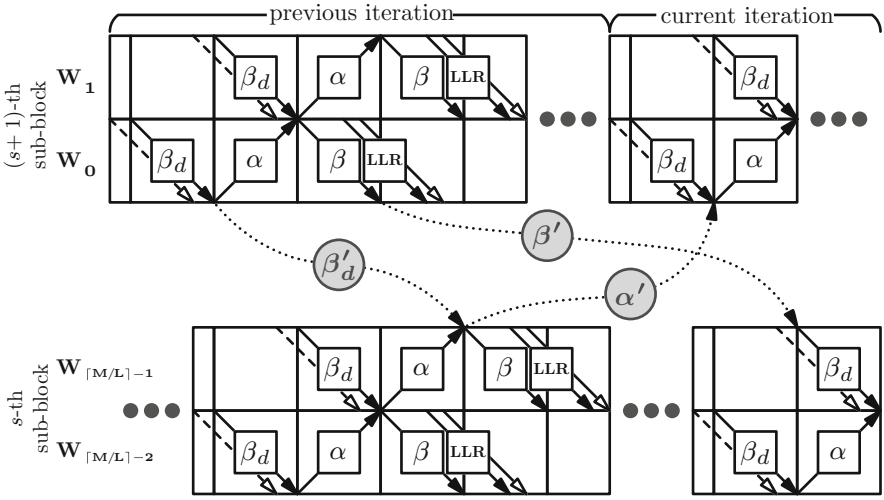

Fig. 4.5 Path metric transmission between two parallel SISO decoders

SISO decoder updates its path metrics from the adjacent one. The $\boldsymbol{\beta_d}'$ and $\boldsymbol{\alpha'}$ are normal initialization procedures, while $\boldsymbol{\beta}'$ acts as a supplementary compensation. As depicted in the figure, the $\beta_d(S_{s\mathbf{M}})$ of the s-th sub-block at the current iteration is initialized with the $\beta(S_{s\mathbf{M}})$ of the $(s+1)$-th sub-block at the previous iteration instead of equal probability. The design will need extra buffers with a capacity of $16\boldsymbol{\varrho}(\alpha(S_i))\times\mathcal{P}_S$ bits to support this $\boldsymbol{\beta}'$ compensation. Since the reliability of dummy backward metric calculation is raised, the problem caused by insufficient window length is relieved.

Figure 4.6 shows the fixed-point simulation results of practical decoders that adopt the above approach. The $\mathbf{L}$ is fixed in the examples of the same $\mathbf{N}$ except for too small sub-blocks. From these results, the performance of the designs with $\mathcal{P}_S > 1$ is close to that of non-parallelized turbo decoders. Even though the last window of every sub-block has a length of merely 1 or 2, the worst degradation is less than 0.3 dB at the BER of 10^{-6}. Compared to the results in Fig. 3.2, the $\boldsymbol{\beta}'$ compensation can achieve similar improvements to the adjustment of $\mathbf{L}$. This approach also works well for the extreme case that has only one available $\mathbf{L}$. For instance, it can greatly reduce the performance loss of the 3GPP LTE-Advanced turbo code with $\mathbf{N} = 40$ and $\mathcal{P}_S = 8$ ($\mathbf{M} = \mathbf{L} = 5$). In summary, its distinct advantages include the consistency in $\mathbf{L}$ and the suitability for any $\{\mathbf{N}, \mathcal{P}_S\}$.

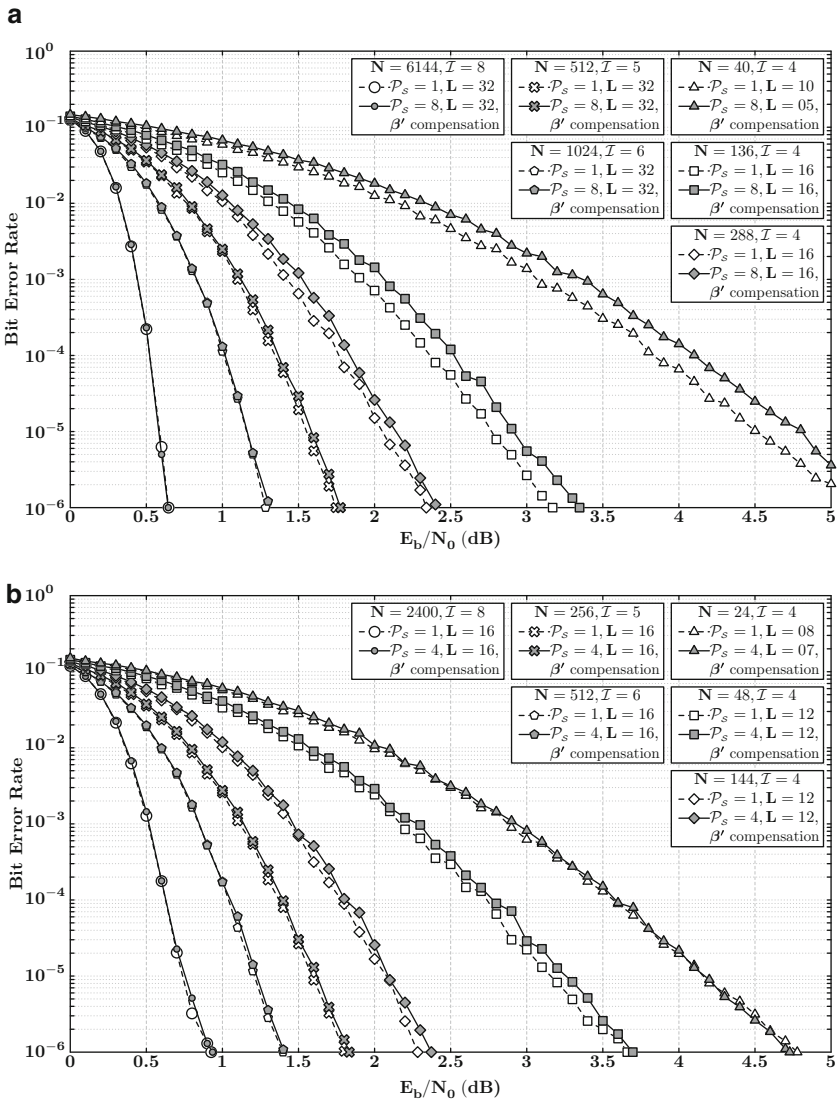

Fig. 4.6 Fixed-point simulation results of 3GPP LTE-Advanced and IEEE 802.16m turbo codes with various cases of $(\mathbf{N}, \mathcal{I}, \mathcal{P}_S, \mathbf{L})$ and $\boldsymbol{\beta}'$ compensation: Max-Log-MAP algorithm with $\zeta = 0.75$ and $(\varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i)) = (3, 3)$. (**a**) 3GPP LTE-Advanced turbo codes: $\mathbf{N} = \{40, 136, 288, 512, 1024, 6144\}$, $\mathcal{P}_S = \{1, 8\}$ with division mapping, and normalization with (2.15). (**b**) IEEE 802.16m turbo codes: $\mathbf{N} = \{24, 48, 144, 256, 512, 2400\}$, $\mathcal{P}_S = \{1, 4\}$ with modulo mapping, and modulo normalization

Chapter 5
High-Efficiency Solution for Highly Parallel Architecture

The data dependency between the constituent codes is the reason for imperfect operating efficiency. Before the beginning of any decoding round (half-iteration), the turbo decoder must renew all extrinsic information with the latest results to get the maximum benefit of message-passing algorithm. Such a basic rule makes the functional units idle for many clock cycles during the transition between two consecutive decoding rounds. For the highly parallel architecture whose each component SISO decoder frequently deals with small sub-blocks, these inactive periods can be very detrimental to its throughput. In this chapter, we introduce two methodologies able to fill the processing schedule with the tasks relating to independent data. The first one is interlacing the decoding rounds for multiple codeword blocks [74]. It could be applied to any turbo decoder; however, it might cost more storage elements and take longer processing time. Our discussions will point out in which cases the operating efficiency can be optimized with the least overhead. The second solution is overlapping the decoding rounds for the original and permuted data sequences [59, 76]. It is a clear violation of the rule mentioned above and quite likely to create problems with decoding performance and memory access. The former is due to the lack of reliable APP estimation for succeeding decoding rounds; the latter occurs when the memory address for write operation of unfinished decoding round coincides with that for read operation of new decoding round. Based on the central idea in [59, 76], these problems can be avoided by designing interleaving rules and arranging data execution order. Here we propose the specialized QPP and ARP interleavers and show the modified processing schedule. Moreover, the best suitable case will be highlighted. As the interleavers match the propositions, the turbo decoder can maximize the usage of all functional units and raise its throughput significantly.

C.-C. Wong and H.-C. Chang, *Turbo Decoder Architecture for Beyond-4G Applications*, DOI 10.1007/978-1-4614-8310-6_5,

5.1 Processing Schedule with Interlaced Decoding Rounds

The concept of interlaced decoding rounds is derived from the parallel turbo decoder level. Instead of adding physical SISO decoders, we utilize the inactive periods of functional units in ordinary schedule to decode multiple codewords. When any functional unit finishes the operations of current decoding round for one sub-block, it soon starts the decoding round for the other equal-sized sub-block at the next clock cycle. Figure 5.1 shows how to switch the processes of two independent sub-blocks, each of which follows the typical schedule in Fig. 2.6. In this example with $\mathbf{M} = 4\mathbf{L}$, the methodology increases the operating efficiency to 100%; however, it produces a delay between two successive decoding rounds for the same codeword because the process for the other codeword occupies the functional units. The interlaced decoding rounds might extend the latency of a single codeword. On the other hand, the overlapping interval can save much time for every two codewords. Actually, the advantage will outweigh the timing overhead, and the decoder can get the decisions of most codewords in a data stream much earlier.

For the design with interlaced decoding rounds, its operating efficiency η'_S and cycle number per round $\mathbf{\Gamma}'_{\mathbf{R}}$ depends on the sub-block size. Every functional unit spends τ_y or $\mathbf{M}$ cycles on one sub-block per decoding round. The inactive period

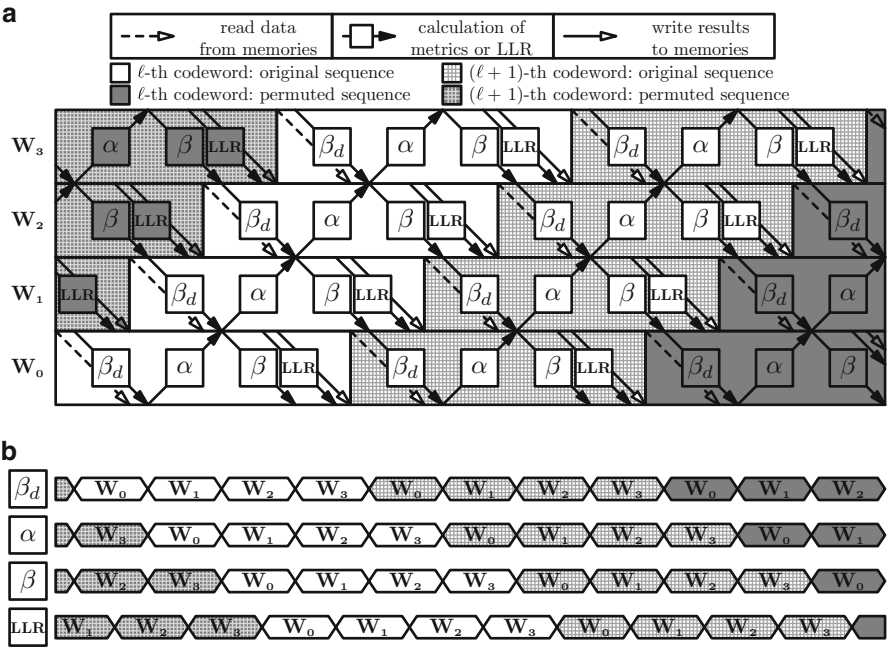

Fig. 5.1 Interlaced decoding rounds of two 4-windowed sub-blocks. (**a**) Processing schedule with 100% operating efficiency. (**b**) Corresponding active periods of main components

in ordinary schedule is $(\tau_a + \tau_b + \tau_x)$, abbreviated to $\tau_{a,b,x}$ for convenience. It is also the maximum interval in which the processes of two independent sub-blocks overlap. Note that τ_y is a variable determined by $\mathbf{N}$ and $\mathcal{P}_\mathcal{S}$; whereas $\tau_{a,b,x}$ is a constant for $\mathbf{M} \geq \mathbf{L}$. If τ_y is less than $\tau_{a,b,x}$, or equivalently $\eta_\mathcal{S} < 50\%$, all functional units will have double operating efficiency but still become inactive for a short while. Otherwise, $\eta'_\mathcal{S}$ will rise to 100%, but the next decoding round for the same codeword will be postponed by $(\tau_y - \tau_{a,b,x})$ cycles. Therefore we can summarize $\eta'_\mathcal{S}$ in (5.1) and $\mathbf{\Gamma}'_\mathbf{R}$ in (5.2). Generally, $(\eta'_\mathcal{S}/\eta_\mathcal{S})$ is the upper bound of throughput improvement. Because the decoding rounds for these two codewords are joined together, total processing time is the necessary information for exact throughput calculation. We denote the overall clock cycles for the designs with ordinary schedule and interlaced rounds to finish decoding the ℓ-th codeword by $\mathbf{\Gamma}_{\mathbf{C}_\ell}$ and $\mathbf{\Gamma}'_{\mathbf{C}_\ell}$ respectively. Like $\eta'_\mathcal{S}$ and $\mathbf{\Gamma}'_\mathbf{R}$, the value of $\mathbf{\Gamma}'_{\mathbf{C}_\ell}$ is also affected by $\eta_\mathcal{S}$. The variations from $\mathbf{\Gamma}_{\mathbf{C}_\ell}$ to $\mathbf{\Gamma}'_{\mathbf{C}_\ell}$ are shown in (5.3) for $\ell = 0$ and (5.4) for $\ell = 1$. Although the increasing latency for the first codeword $(\mathbf{\Gamma}'_{\mathbf{C}_0} - \mathbf{\Gamma}_{\mathbf{C}_0})$ is at most $(2\mathcal{I} - 1) \times (\tau_y - \tau_{a,b,x})$ cycles, the reducing time to get the initial two codewords $(\mathbf{\Gamma}_{\mathbf{C}_1} - \mathbf{\Gamma}'_{\mathbf{C}_1})$ is $(2\mathcal{I} - 1) \times (\tau_y - \tau_{a,b,x})$ and $(4\mathcal{I} - 2) \times \tau_{a,b,x}$ cycles for $\eta_\mathcal{S} < 50\%$ and $\eta_\mathcal{S} \geq 50\%$ respectively.

$$\eta_\mathcal{S} = \tau_y/(\tau_y + \tau_{a,b,x}) \quad \Rightarrow \eta'_\mathcal{S} = \begin{cases} 2\tau_y/(\tau_y + \tau_{a,b,x}) & \text{if } \eta_\mathcal{S} < 50\% \\ 100\% & \text{if } \eta_\mathcal{S} \geq 50\% \end{cases} \tag{5.1}$$

$$\mathbf{\Gamma}_\mathbf{R} = \tau_y + \tau_{a,b,x} \quad \Rightarrow \mathbf{\Gamma}'_\mathbf{R} = \begin{cases} \tau_y + \tau_{a,b,x} & \text{if } \eta_\mathcal{S} < 50\% \\ 2\tau_y & \text{if } \eta_\mathcal{S} \geq 50\% \end{cases} \tag{5.2}$$

$$\mathbf{\Gamma}_{\mathbf{C}_0} = 2\mathcal{I} \times (\tau_y + \tau_{a,b,x}) \quad \Rightarrow \mathbf{\Gamma}'_{\mathbf{C}_0} = \begin{cases} 2\mathcal{I} \times (\tau_y + \tau_{a,b,x}) & \text{if } \eta_\mathcal{S} < 50\% \\ 2\mathcal{I} \times 2\tau_y - \tau_y + \tau_{a,b,x} & \text{if } \eta_\mathcal{S} \geq 50\% \end{cases} \tag{5.3}$$

$$\mathbf{\Gamma}_{\mathbf{C}_1} = 4\mathcal{I} \times (\tau_y + \tau_{a,b,x}) - \tau_{a,b,x} \Rightarrow \mathbf{\Gamma}'_{\mathbf{C}_1} = \begin{cases} 2\mathcal{I} \times (\tau_y + \tau_{a,b,x}) + \tau_y & \text{if } \eta_\mathcal{S} < 50\% \\ 2\mathcal{I} \times 2\tau_y + \tau_{a,b,x} & \text{if } \eta_\mathcal{S} \geq 50\% \end{cases} \tag{5.4}$$

We can use (5.4) to further derive the general forms of $\mathbf{\Gamma}_{\mathbf{C}_\ell}$ in (5.5) and $\mathbf{\Gamma}'_{\mathbf{C}_\ell}$ in (5.6) for those ℓ's satisfying $2 \mid (\ell + 1)$. That is, only even number of codewords are considered. The actual speedup is $\mathbf{\Gamma}_{\mathbf{C}_\ell}/\mathbf{\Gamma}'_{\mathbf{C}_\ell}$. It will more closely approximate to $\eta'_\mathcal{S}/\eta_\mathcal{S}$ as the design processes more codewords.

$$\mathbf{\Gamma}_{\mathbf{C}_\ell} = (\lfloor \ell/2 \rfloor + 1) \times \mathbf{\Gamma}_{\mathbf{C}_1} - \lfloor \ell/2 \rfloor \times \tau_{a,b,x} \tag{5.5}$$

$$\mathbf{\Gamma}'_{\mathbf{C}_\ell} = (\lfloor \ell/2 \rfloor + 1) \times \mathbf{\Gamma}'_{\mathbf{C}_1} - \lfloor \ell/2 \rfloor \times \tau_{a,b,x} \tag{5.6}$$

Table 5.1 lists the gain and loss in efficiency and latency of this schedule as the design processes six different small sub-blocks. All these calculations are based on the following assumptions: $\mathcal{I} = 8$, $\mathbf{L} = 32$, and $\tau_{a,b,x} = 74$. The examples with

Table 5.1 Comparison between ordinary schedule and interlaced decoding rounds

	Ordinary schedule				Interlaced rounds			
M	η_S (%)	$\mathbf{\Gamma_R}$	$\mathbf{\Gamma_{C_0}}$	$\mathbf{\Gamma_{C_1}}$	η'_S (%)	$\mathbf{\Gamma'_R}$	$\mathbf{\Gamma'_{C_0}}$	$\mathbf{\Gamma'_{C_1}}$
32	30	106	1696	3318	60	106	1696	1728
64	46	138	2208	4342	92	138	2208	2272
96	56	170	2720	5366	100	192	3050	3146
128	63	202	3232	6390	100	256	4042	4170
160	68	234	3744	7414	100	320	5034	5194
192	72	266	4256	8438	100	384	6026	6218

$\tau_y < \tau_{a,b,x}$ can achieve $\eta'_S = 2\eta_S$ without sacrificing $\mathbf{\Gamma'_{C_0}}$, while those with $\tau_y \geq \tau_{a,b,x}$ can get the maximum $(\mathbf{\Gamma_{C_1}} - \mathbf{\Gamma'_{C_1}})$ equal to 2220. It is obvious that the extra delay of the first codeword becomes much longer for larger sub-blocks. Consequently, we prefer applying this modified schedule to those cases whose **M**'s are close to $\tau_{a,b,x}$ so that the design can enjoy more benefits.

5.2 Processing Schedule with Overlapping Decoding Rounds

For successful overlapping decoding rounds, we must develop the interleaving function and the processing schedule jointly. The window-wise strategy in [76] is adopted here. All original data are divided equally into two window groups $\{\mathcal{W}_{\rm I}, \mathcal{W}_{\rm II}\}$ according to their window indexes. Similarly, all permuted data are classified into $\{\tilde{\mathcal{W}}_{\rm I}, \tilde{\mathcal{W}}_{\rm II}\}$. Note that the total window number $(\mathbf{N}/\mathbf{L})$ should be an even number. The interleaver needs to map $\mathcal{W}_{\rm I}$ exactly onto either one of $\tilde{\mathcal{W}}_{\rm I}$ or $\tilde{\mathcal{W}}_{\rm II}$. It implies that $\mathcal{W}_{\rm II}$ and the remaining window group of permuted data will be correlated with each other. Then we take the advantage of the schedule in [43, 44], where all windows can be processed in arbitrary order by utilizing previous path metrics. If the mapping is $\mathcal{W}_{\rm I} \mapsto \tilde{\mathcal{W}}_{\rm I}$ and $\mathcal{W}_{\rm II} \mapsto \tilde{\mathcal{W}}_{\rm II}$, after proper arrangement, the processes of $\mathcal{W}_{\rm I}$ and of $\mathcal{W}_{\rm II}$ can overlap with the processes of $\tilde{\mathcal{W}}_{\rm II}$ and of $\tilde{\mathcal{W}}_{\rm I}$ respectively. Thus, the design using overlapping decoding rounds can prevent memory hazards and great performance loss.

Both the classification methods and permutation rules are main topics of interleaver design for overlapping decoding rounds. To support our discussion, the i-th data, r_i and $\tilde{r}_i$, will be denoted by, respectively, $r_{s\mathbf{L}+j}$ and $\tilde{r}_{s\mathbf{L}+j}$, indicating the j-th data of the s-th window in either sequence. The expression is analogous to (3.2). By replacing **M** with **L**, we can get $\mathcal{Q}(s\mathbf{L}+j)$ from (3.3) and $\mathcal{A}(s\mathbf{L}+j)$ from (3.13). Since we are interested in the relation between window indexes, it is necessary to find out $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ by (3.4) and $\boldsymbol{s}^{\mathcal{A}}_{s;j}$ by (3.14). Furthermore, the circular trellis structure is treated as a prerequisite for a wider choice of QPP and ARP interleavers. It allows the turbo decoder to process the rotated data sequence with an offset ρ: $[\tilde{r}_\rho, \tilde{r}_{\rho+1}, \ldots, \tilde{r}_{\mathbf{N}-1}, \tilde{r}_0, \ldots, \tilde{r}_{\rho-1}]$. Without loss of generality, the range of ρ is set to

$0 \leq \rho < \mathbf{L}$. We redefine this sequence as $[\tilde{r}'_0, \tilde{r}'_1, \ldots, \tilde{r}'_{\mathbf{N}-1}]$ and express their data indexes with the alternative form: $\boldsymbol{s}^{(\rho)}_{s;j}\mathbf{L} + \boldsymbol{j}^{(\rho)}_{s;j}$. The connection between the normal and new indexes is

$$s\mathbf{L} + j = \boldsymbol{s}^{(\rho)}_{s;j}\mathbf{L} + \boldsymbol{j}^{(\rho)}_{s;j} + \rho \pmod{\mathbf{N}}; \tag{5.7}$$

so we can derive the conversions in (5.8) and (5.9).

$$j = \boldsymbol{j}^{(\rho)}_{s;j} + \rho \pmod{\mathbf{L}} \tag{5.8}$$

$$s = \boldsymbol{s}^{(\rho)}_{s;j} + \left\lfloor \frac{\boldsymbol{j}^{(\rho)}_{s;j} + \rho}{\mathbf{L}} \right\rfloor \pmod{\frac{\mathbf{N}}{\mathbf{L}}} \tag{5.9}$$

In this section, the windows are classified by their indexes ($\boldsymbol{s}^{(\rho)}_{s;j}$, $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$, and $\boldsymbol{s}^{\mathcal{A}}_{s;j}$) modulo 2. The determination of ρ is one essential part of classification method. We will present some constraints on interleavers and specify ρ to guarantee that certain relation between $\boldsymbol{s}^{(\rho)}_{s;j}$ and $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ or $\boldsymbol{s}^{\mathcal{A}}_{s;j}$ can hold true for $0 \leq \boldsymbol{s}^{(\rho)}_{s;j} < (\mathbf{N}/\mathbf{L})$ and $0 \leq \boldsymbol{j}^{(\rho)}_{s;j} < \mathbf{L}$. Based on these propositions, we can further plan the schedule with overlapping decoding rounds and examine its speedup. The corresponding error correction capability is the last issue. The comparison between the unusual interleavers with restricted parameters and the normal interleavers defined in standards will show to what extent the constraints affect performance.

5.2.1 *QPP Interleaver Design for Overlapping Decoding Rounds*

The QPP interleaver design involves the parameters (f_1, f_2), the window length $\mathbf{L}$, and the indexes $(\boldsymbol{s}^{(\rho)}_{s;j}, \boldsymbol{j}^{(\rho)}_{s;j})$. Our introduction starts with (5.10), the expansion of (3.4) with $\mathbf{M} = \mathbf{L}$ and $\mathcal{P}_{\mathcal{S}} = \mathbf{N}/\mathbf{L}$.

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} = f_1 s + f_2 s\mathbf{L} + 2f_2 s\mathbf{L}j + \left\lfloor \frac{f_1 j + f_2 j^2}{\mathbf{L}} \right\rfloor \pmod{\frac{\mathbf{N}}{\mathbf{L}}} \tag{5.10}$$

Under the precondition $2 \mid (\mathbf{N}/\mathbf{L})$, both sides in this equation are congruent modulo 2. We impose two basic constraints, $2 \nmid f_1$ and $2 \mid f_2$, to simplify it. The first one implies that $f_1 s$ is congruent to s modulo 2; while the second one makes both $f_2 s\mathbf{L}$ and $2f_2 s\mathbf{L}j$ modulo 2 equal 0. Thanks to these properties, (5.10) can be rewritten as

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} \equiv s + \left\lfloor \frac{f_1 j + f_2 j^2}{\mathbf{L}} \right\rfloor \pmod{2}. \tag{5.11}$$

In the following two propositions, we state the major constraints on (f_1, f_2) and their respective ρ's that can transform the summation in the right side of (5.11) into $\boldsymbol{s}^{(\rho)}_{s;j}$ plus a constant number.

Proposition 5.1. *If the QPP interleaver parameters satisfy*

$$\begin{cases} \mathbf{L} \mid (f_1 - 1) & \text{(5.12a)} \\ \mathbf{L} \mid f_2 & \text{(5.12b)} \\ (f_1 - 1)/\mathbf{L} \equiv f_2/\mathbf{L} \pmod 2 & \text{(5.12c)} \end{cases}$$

and $\rho = 0$*, then* $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ *is congruent to* $\boldsymbol{s}^{(\rho)}_{s;j}$ *modulo* 2.

Proof. (a) We can factorize $\lfloor (f_1 j + f_2 j^2)/\mathbf{L} \rfloor$ into

$$\left\lfloor \frac{(f_1 - 1)j}{\mathbf{L}} + \frac{f_2 j^2}{\mathbf{L}} + \frac{j}{\mathbf{L}} \right\rfloor. \tag{5.13}$$

(b) Because of (5.12a) and (5.12b), both $(f_1 - 1)j/\mathbf{L}$ and $f_2 j^2/\mathbf{L}$ are integers, and they can be moved out from the floor function as

$$\frac{(f_1 - 1)j}{\mathbf{L}} + \frac{f_2 j^2}{\mathbf{L}} + \left\lfloor \frac{j}{\mathbf{L}} \right\rfloor. \tag{5.14}$$

(c) If j is an odd number, $(f_1 - 1)j/\mathbf{L}$ and $f_2 j^2/\mathbf{L}$ are congruent modulo 2 after applying (5.12c); otherwise, they both contain a factor of 2. In either case, these constraints make their summation always be an even integer; so we can view the first two terms in (5.14) as 0 in modulo-2 arithmetic and only take account of $\lfloor j/\mathbf{L} \rfloor$.

(d) The range of j $(0 \le j < \mathbf{L})$ implies $0 \le j/\mathbf{L} < 1$ and $\lfloor j/\mathbf{L} \rfloor = 0$, that is, the last term in (5.14) can be eliminated. As a result, $\lfloor (f_1 j + f_2 j^2)/\mathbf{L} \rfloor$ is congruent to 0 modulo 2. On employing this property in (5.11), $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ is congruent to s modulo 2.

(e) The conversion in (5.9) with $\rho = 0$ means that s is exactly equal to $\boldsymbol{s}^{(\rho)}_{s;j}$, so we have

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} \equiv \boldsymbol{s}^{(\rho)}_{s;j} \pmod 2. \tag{5.15}$$

■

Proposition 5.2. *If the QPP interleaver parameters satisfy*

$$\begin{cases} \mathbf{L} \mid (f_1 + 1) & \text{(5.16a)} \\ \mathbf{L} \mid f_2 & \text{(5.16b)} \\ (f_1 + 1)/\mathbf{L} \equiv f_2/\mathbf{L} \pmod 2 & \text{(5.16c)} \end{cases}$$

and $\rho = 1$*, then* $\boldsymbol{s}^{\mathcal{Q}}_{s;j}$ *is congruent to* $(\boldsymbol{s}^{(\rho)}_{s;j} + 1)$ *modulo* 2.

Proof. (a) First, $\lfloor (f_1 j + f_2 j^2)/\mathbf{L} \rfloor$ is factorized into

$$\left\lfloor \frac{(f_1+1)j}{\mathbf{L}} + \frac{f_2 j^2}{\mathbf{L}} - \frac{j}{\mathbf{L}} \right\rfloor. \tag{5.17}$$

(b) Next, we substitute (5.17) for $\lfloor (f_1 j + f_2 j^2)/\mathbf{L} \rfloor$ in (5.11). The constraints (5.16a)–(5.16c) imply that the summation of the first two terms in (5.17) is an even integer, so we have the following equality:

$$\begin{aligned} \boldsymbol{s}^{\mathcal{Q}}_{s;j} &\equiv s + \frac{(f_1+1)j}{\mathbf{L}} + \frac{f_2 j^2}{\mathbf{L}} + \left\lfloor -\frac{j}{\mathbf{L}} \right\rfloor \pmod 2 \\ &\equiv s + \left\lfloor -\frac{j}{\mathbf{L}} \right\rfloor \pmod 2. \end{aligned} \tag{5.18}$$

(c) The conversions in (5.8) and (5.9) with $\rho = 1$ can change (5.18) into

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} \equiv \boldsymbol{s}^{(\rho)}_{s;j} + \left\lfloor \frac{\boldsymbol{j}^{(\rho)}_{s;j} + 1}{\mathbf{L}} \right\rfloor + \left\lfloor -\frac{(\boldsymbol{j}^{(\rho)}_{s;j} + 1) \bmod \mathbf{L}}{\mathbf{L}} \right\rfloor \pmod 2, \tag{5.19}$$

where the value of each floor function is dependent on $\boldsymbol{j}^{(\rho)}_{s;j}$. The first floor function generates 0 for $0 \le \boldsymbol{j}^{(\rho)}_{s;j} < (\mathbf{L}-1)$ and 1 for $\boldsymbol{j}^{(\rho)}_{s;j} = (\mathbf{L}-1)$; while the second floor function equals -1 for $0 \le \boldsymbol{j}^{(\rho)}_{s;j} < (\mathbf{L}-1)$ and 0 for $\boldsymbol{j}^{(\rho)}_{s;j} = (\mathbf{L}-1)$. Then we list all possible cases of (5.19):

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} \equiv \begin{cases} \boldsymbol{s}^{(\rho)}_{s;j} + 0 - 1 \pmod 2 & \text{if } \boldsymbol{j}^{(\rho)}_{s;j} \neq (\mathbf{L}-1), \\ \boldsymbol{s}^{(\rho)}_{s;j} + 1 + 0 \pmod 2 & \text{if } \boldsymbol{j}^{(\rho)}_{s;j} = (\mathbf{L}-1). \end{cases} \tag{5.20}$$

(d) Because 1 and (-1) are congruent modulo 2, for every $\boldsymbol{j}^{(\rho)}_{s;j}$, the summation of the two floor functions in (5.19) modulo 2 is always 1. By merging both cases in (5.20), we obtain

$$\boldsymbol{s}^{\mathcal{Q}}_{s;j} \equiv \boldsymbol{s}^{(\rho)}_{s;j} + 1 \pmod 2 \tag{5.21}$$

■

The decomposition of $\lfloor (f_1 j + f_2 j^2)/\mathbf{L} \rfloor$ is the most critical step in proving these propositions. Each set of constraints imposed on (f_1, f_2) and $\mathbf{L}$ minimizes the possible outcomes of this floor function. Although the number of outcomes might be still more than one, we can use index transformation with a nonzero ρ to resolve it. Since (5.15) and (5.21) are valid for $\boldsymbol{s}^{(\rho)}_{s;j}$ rather than s, the practical design should adjust the way of accessing memory to support the schedule with overlapping decoding rounds.

5.2.2 ARP Interleaver Design for Overlapping Decoding Rounds

Analogous to the QPP interleaver design, we correlates the parameters of ARP interleaver, $(\varepsilon, g_0, g_1, g_2, g_3)$, with $\mathbf{L}$ to accomplish the design objective. The first step is deriving (5.22) by expanding (3.14) and then replacing $\mathbf{M}$ with $\mathbf{L}$ and $\mathcal{P}_S$ with $\mathbf{N}/\mathbf{L}$.

$$\boldsymbol{s}^{\mathcal{A}}_{s;j} = \varepsilon s + \left\lfloor \frac{\varepsilon j + g_{(s\mathbf{L}+j) \bmod 4}}{\mathbf{L}} \right\rfloor \pmod{\frac{\mathbf{N}}{\mathbf{L}}}. \tag{5.22}$$

This equality is still valid under modulo-2 arithmetic. There are also two basic constraints: $2 \nmid \varepsilon$ and $4 \mid \mathbf{L}$; the first one indicates that εs and s are congruent modulo 2, and the second one means $g_{(s\mathbf{L}+j) \bmod 4} = g_{j \bmod 4}$. With these properties, (5.22) can be modified into

$$\boldsymbol{s}^{\mathcal{A}}_{s;j} \equiv s + \left\lfloor \frac{\varepsilon j + g_{j \bmod 4}}{\mathbf{L}} \right\rfloor \pmod 2. \tag{5.23}$$

The following two propositions give the major constraints on $(\varepsilon, g_0, g_1, g_2, g_3)$ and their respective ρ's to transform the summation in the right side of (5.23) into $\boldsymbol{s}^{(\rho)}_{s;j}$ plus a constant number.

Proposition 5.3. *If the ARP interleaver parameters satisfy*

$$\begin{cases} \mathbf{L} \mid (\varepsilon - 1) & \text{(5.24a)} \\ g_0 \equiv g_1 \equiv g_2 \equiv g_3 \pmod{\mathbf{L}} & \text{(5.24b)} \\ \lfloor g_0/\mathbf{L} \rfloor \equiv \lfloor g_2/\mathbf{L} \rfloor \pmod 2 & \text{(5.24c)} \\ \lfloor g_1/\mathbf{L} \rfloor \equiv \lfloor g_3/\mathbf{L} \rfloor \pmod 2 & \text{(5.24d)} \\ (\varepsilon - 1)/\mathbf{L} \equiv \lfloor g_0/\mathbf{L} \rfloor + \lfloor g_1/\mathbf{L} \rfloor \pmod 2 & \text{(5.24e)} \end{cases}$$

and

$$\rho = \mathbf{L} - (g_0 \bmod \mathbf{L}), \tag{5.25}$$

then $\boldsymbol{s}^{\mathcal{A}}_{s;j}$ *is congruent to* $(\boldsymbol{s}^{(\rho)}_{s;j} + \lfloor g_0/\mathbf{L} \rfloor + 1)$ *modulo* 2.

Proof. (a) We first factorize εj and $g_{j \bmod 4}$ separately. It is straightforward to replace εj with the summation of $(\varepsilon - 1)j$ and j. For the decomposition of $g_{j \bmod 4}$, the division of $g_{j \bmod 4}$ by $\mathbf{L}$, whose quotient is $\lfloor g_{j \bmod 4}/\mathbf{L} \rfloor$ and remainder is $g_{j \bmod 4}$ modulo $\mathbf{L}$, is exploited to get

$$g_{j \bmod 4} = \left\lfloor \frac{g_{j \bmod 4}}{\mathbf{L}} \right\rfloor \times \mathbf{L} + ((g_{j \bmod 4}) \bmod \mathbf{L}). \tag{5.26}$$

The constraint (5.24a) implies that $(\varepsilon-1)j/\mathbf{L}$ is an integer, and $\lfloor g_{j \bmod 4}/\mathbf{L} \rfloor$ is trivially an integer. After substituting these results back to (5.23), we can therefore obtain

$$\boldsymbol{s}_{s;j}^{\mathcal{A}} \equiv s + \frac{(\varepsilon-1)j}{\mathbf{L}} + \left\lfloor \frac{g_{j \bmod 4}}{\mathbf{L}} \right\rfloor + \left\lfloor \frac{j}{\mathbf{L}} + \frac{(g_{j \bmod 4}) \bmod \mathbf{L}}{\mathbf{L}} \right\rfloor \pmod 2. \quad (5.27)$$

(b) Then we examine all cases of (5.27). If j is divisible by 2, $(\varepsilon-1)j/\mathbf{L}$ also contains a factor of 2. Since $g_{j \bmod 4}$ is either g_0 or g_2 now, the constraints (5.24b) and (5.24c) promise that, for any even j, (5.27) can be rewritten as (5.28), where the terms in the right side are simplified to $(s + \boldsymbol{\lambda_0} + \boldsymbol{\kappa_0})$.

$$\boldsymbol{s}_{s;j}^{\mathcal{A}} \equiv s + \underbrace{0 + \left\lfloor \frac{g_0}{\mathbf{L}} \right\rfloor}_{\boldsymbol{\lambda_0}} + \underbrace{\left\lfloor \frac{j}{\mathbf{L}} + \frac{g_0 \bmod \mathbf{L}}{\mathbf{L}} \right\rfloor}_{\boldsymbol{\kappa_0}} \pmod 2 \quad (5.28)$$

If j modulo 2 is 1, $(\varepsilon-1)j/\mathbf{L}$ modulo 2 become independent of j, and the possible values of $g_{j \bmod 4}$ include g_1 and g_3. Under this condition, we apply (5.24b) and (5.24d) so that, for any odd j, (5.29) is an alternative to (5.27). This equation is also expressed as $(s + \boldsymbol{\lambda_1} + \boldsymbol{\kappa_1})$ here.

$$\boldsymbol{s}_{s;j}^{\mathcal{A}} \equiv s + \underbrace{\frac{(\varepsilon-1)}{\mathbf{L}} + \left\lfloor \frac{g_1}{\mathbf{L}} \right\rfloor}_{\boldsymbol{\lambda_1}} + \underbrace{\left\lfloor \frac{j}{\mathbf{L}} + \frac{g_1 \bmod \mathbf{L}}{\mathbf{L}} \right\rfloor}_{\boldsymbol{\kappa_1}} \pmod 2 \quad (5.29)$$

In fact, we can infer from (5.24e) that $\boldsymbol{\lambda_0}$ and $\boldsymbol{\lambda_1}$ are congruent modulo 2. Moreover, (5.24b) suggests that the constant terms inside $\boldsymbol{\kappa_0}$ and $\boldsymbol{\kappa_1}$ are the same. Hence, (5.28) and (5.29) both can be used to find the relation between $\boldsymbol{s}_{s;j}^{\mathcal{A}}$ and s for all j's.

(c) On substituting (5.8) for j, (5.9) for s, and (5.25) for ρ, the right side of (5.28) can be converted into

$$\boldsymbol{s}_{s;j}^{(\rho)} + \left\lfloor \frac{g_0}{\mathbf{L}} \right\rfloor + \underbrace{\left\lfloor \frac{\boldsymbol{j}_{s;j}^{(\rho)} + \rho}{\mathbf{L}} \right\rfloor}_{\boldsymbol{\lambda_2}} + \underbrace{\left\lfloor \frac{(\boldsymbol{j}_{s;j}^{(\rho)} + \rho) \bmod \mathbf{L}}{\mathbf{L}} + \frac{\mathbf{L}-\rho}{\mathbf{L}} \right\rfloor}_{\boldsymbol{\kappa_2}}, \quad (5.30)$$

where we express g_0 modulo $\mathbf{L}$ by $(\mathbf{L}-\rho)$ and denote the last two floor functions by $\boldsymbol{\lambda_2}$ and $\boldsymbol{\kappa_2}$ respectively. The variable $\boldsymbol{j}_{s;j}^{(\rho)}$ affects the outcomes of $\boldsymbol{\lambda_2}$ and $\boldsymbol{\kappa_2}$. If $\boldsymbol{j}_{s;j}^{(\rho)}$ is less than $(\mathbf{L}-\rho)$, we can get upper and lower bounds of the terms involving $\boldsymbol{j}_{s;j}^{(\rho)}$ as (5.31); otherwise, we derive (5.32).

$$0 \le \boldsymbol{j}_{s;j}^{(\rho)} < \mathbf{L}-\rho \Rightarrow \left\{ \begin{array}{ll} \rho \le \boldsymbol{j}_{s;j}^{(\rho)} + \rho & < \mathbf{L} \\ \rho \le (\boldsymbol{j}_{s;j}^{(\rho)} + \rho) \bmod \mathbf{L} & < \mathbf{L} \end{array} \right\} \quad (5.31)$$

$$\mathbf{L} > \boldsymbol{j}_{s;j}^{(\rho)} \geq \mathbf{L} - \rho \Rightarrow \left\{ \begin{array}{ll} \mathbf{L} \leq \boldsymbol{j}_{s;j}^{(\rho)} + \rho & < \mathbf{L} + \rho \\ 0 \leq (\boldsymbol{j}_{s;j}^{(\rho)} + \rho) \bmod \mathbf{L} & < \rho \end{array} \right\} \tag{5.32}$$

Based on these derivations, $\boldsymbol{\lambda}_2$ and $\boldsymbol{\kappa}_2$ can be determined as

$$\left\{ \begin{array}{l} \boldsymbol{\lambda}_2 = 0 \text{ and } \boldsymbol{\kappa}_2 = 1 \text{ for } 0 \leq \boldsymbol{j}_{s;j}^{(\rho)} < \mathbf{L} - \rho, \\ \boldsymbol{\lambda}_2 = 1 \text{ and } \boldsymbol{\kappa}_2 = 0 \text{ for } \mathbf{L} > \boldsymbol{j}_{s;j}^{(\rho)} \geq \mathbf{L} - \rho. \end{array} \right. \tag{5.33}$$

In either case, the summation of $\boldsymbol{\lambda}_2$ and $\boldsymbol{\kappa}_2$ is 1. Consequently, we have

$$\boldsymbol{s}_{s;j}^{\mathcal{A}} \equiv \boldsymbol{s}_{s;j}^{(\rho)} + \left\lfloor \frac{g_0}{\mathbf{L}} \right\rfloor + 1 \pmod 2. \tag{5.34}$$

■

Proposition 5.4. *If the ARP interleaver parameters satisfy*

$$\left\{ \begin{array}{ll} \mathbf{L} \mid (\varepsilon + 1) & \text{(5.35a)} \\ g_0 \equiv g_1 \equiv g_2 \equiv g_3 \not\equiv (\mathbf{L} - 1) \pmod{\mathbf{L}} & \text{(5.35b)} \\ \lfloor g_0/\mathbf{L} \rfloor \equiv \lfloor g_2/\mathbf{L} \rfloor \pmod 2 & \text{(5.35c)} \\ \lfloor g_1/\mathbf{L} \rfloor \equiv \lfloor g_3/\mathbf{L} \rfloor \pmod 2 & \text{(5.35d)} \\ (\varepsilon + 1)/\mathbf{L} \equiv \lfloor g_0/\mathbf{L} \rfloor + \lfloor g_1/\mathbf{L} \rfloor \pmod 2 & \text{(5.35e)} \end{array} \right.$$

and

$$\rho = ((g_0 + 1) \bmod \mathbf{L}), \tag{5.36}$$

then $\boldsymbol{s}_{s;j}^{\mathcal{A}}$ *is congruent to* $(\boldsymbol{s}_{s;j}^{(\rho)} + \lfloor g_0/\mathbf{L} \rfloor + 1)$ *modulo* 2.

Proof. (a) The initial step is factorizing εj into the summation of $(\varepsilon + 1)j$ and $(-j)$. Because of (5.35a), $(\varepsilon+1)j/\mathbf{L}$ is an integer. Furthermore, the substitution of (5.26) for $g_{j \bmod 4}$ results in another integer $\lfloor g_{j \bmod 4}/\mathbf{L} \rfloor$. By moving these integers out from the floor function in (5.23), we have

$$\boldsymbol{s}_{s;j}^{\mathcal{A}} \equiv s + \frac{(\varepsilon + 1)j}{\mathbf{L}} + \left\lfloor \frac{g_{j \bmod 4}}{\mathbf{L}} \right\rfloor + \left\lfloor \frac{(g_{j \bmod 4}) \bmod \mathbf{L}}{\mathbf{L}} - \frac{j}{\mathbf{L}} \right\rfloor \pmod 2. \tag{5.37}$$

(b) The subsequent step is examining (5.37) for all j's. If j modulo 2 is 0, we treat $(\varepsilon + 1)j/\mathbf{L}$ as 0 under modulo-2 arithmetic; otherwise, we replace the variable j within $(\varepsilon+1)j/\mathbf{L}$ by 1. In addition, the three constraints (5.35b), (5.35c), and (5.35d) mean that $g_{j \bmod 4}$ and $g_{(j+2) \bmod 4}$ are interchangeable in (5.37). Hence, we can rewrite (5.37) as $(s + \boldsymbol{\lambda}_0' + \boldsymbol{\kappa}_0')$ in (5.38) for any even j and as $(s + \boldsymbol{\lambda}_1' + \boldsymbol{\kappa}_1')$ in (5.39) for any odd j.

$$s^{\mathcal{A}}_{s;j} \equiv s + \underbrace{0 + \left\lfloor \frac{g_0}{\mathbf{L}} \right\rfloor}_{\lambda'_0} + \underbrace{\left\lfloor \frac{g_0 \bmod \mathbf{L}}{\mathbf{L}} - \frac{j}{\mathbf{L}} \right\rfloor}_{\kappa'_0} \pmod 2 \tag{5.38}$$

$$s^{\mathcal{A}}_{s;j} \equiv s + \underbrace{\frac{(\varepsilon+1)}{\mathbf{L}} + \left\lfloor \frac{g_1}{\mathbf{L}} \right\rfloor}_{\lambda'_1} + \underbrace{\left\lfloor \frac{g_1 \bmod \mathbf{L}}{\mathbf{L}} - \frac{j}{\mathbf{L}} \right\rfloor}_{\kappa'_1} \pmod 2 \tag{5.39}$$

With (5.35e), λ'_0 modulo 2 equals λ'_1 modulo 2. Besides, (5.35b) implies κ'_0 and κ'_1 have identical constant terms. Thus, (5.38) and (5.39) are both valid for describing the relation between $s^{\mathcal{A}}_{s;j}$ and s for any j.

(c) The last step is making substitutions with (5.8), (5.9), and (5.36); then we modify the right side of (5.38) into

$$s^{(\rho)}_{s;j} + \left\lfloor \frac{g_0}{\mathbf{L}} \right\rfloor + \underbrace{\left\lfloor \frac{j^{(\rho)}_{s;j} + \rho}{\mathbf{L}} \right\rfloor}_{\lambda'_2} + \underbrace{\left\lfloor \frac{\rho - 1}{\mathbf{L}} - \frac{(j^{(\rho)}_{s;j} + \rho) \bmod \mathbf{L}}{\mathbf{L}} \right\rfloor}_{\kappa'_2} \pmod 2, \tag{5.40}$$

where g_0 modulo $\mathbf{L}$ is represented as $(\rho-1)$, and the last two floor functions are denoted by λ'_2 and κ'_2 respectively. From the derivations in (5.31) and (5.32), we can further develop (5.41) and (5.42). Note that the inequality in (5.35b), that leads to $1 \le \rho \le (\mathbf{L}-1)$, can prevent $(\rho-1)$ from being $(\mathbf{L}-1)$ and contradicting (5.41) and prevent $(\rho-1)$ from being -1 and contradicting (5.42).

$$0 \le j^{(\rho)}_{s;j} < \mathbf{L} - \rho \Rightarrow \{-1 \ge (\rho-1) - ((j^{(\rho)}_{s;j} + \rho) \bmod \mathbf{L}) > \rho - 1 - \mathbf{L}\} \tag{5.41}$$

$$\mathbf{L} > j^{(\rho)}_{s;j} \ge \mathbf{L} - \rho \Rightarrow \{-1 < (\rho-1) - ((j^{(\rho)}_{s;j} + \rho) \bmod \mathbf{L}) \le \rho - 1\} \tag{5.42}$$

As a result, we can get λ'_2 and κ'_2 as

$$\begin{cases} \lambda'_2 = 0 \text{ and } \kappa'_2 = -1 & \text{for } 0 \le j^{(\rho)}_{s;j} < \mathbf{L} - \rho, \\ \lambda'_2 = 1 \text{ and } \kappa'_2 = 0 & \text{for } \mathbf{L} > j^{(\rho)}_{s;j} \ge \mathbf{L} - \rho. \end{cases} \tag{5.43}$$

Since the summation of λ'_2 and κ'_2 is always 1, (5.40) is equivalent to $(s^{(\rho)}_{s;j} + \lfloor g_0/\mathbf{L} \rfloor + 1)$, and (5.34) holds true for $0 \le j^{(\rho)}_{s;j} < \mathbf{L}$.

■

Propositions 5.3 and 5.4 mainly differ in the constraints on ε, which affect the factorization in their proofs; but they impose almost the same constraints on (g_0, g_1, g_2, g_3). In either proposition, the value of ρ and the relation between $s^{\mathcal{A}}_{s;j}$ and $s^{(\rho)}_{s;j}$ varies with g_0 and $\mathbf{L}$, so we need a more flexible controller that can handle memory access according to input parameters. If we set g_0 as a constant such as the

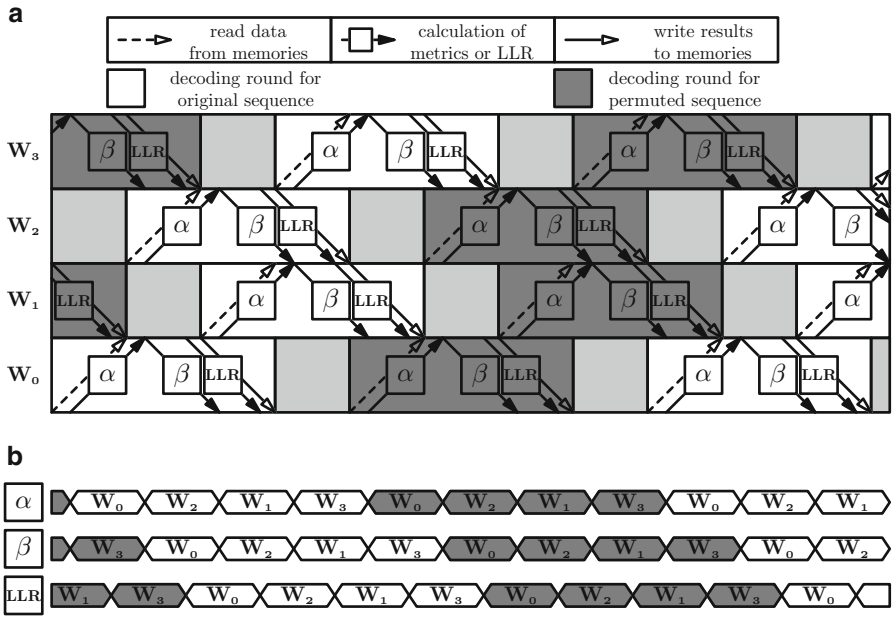

Fig. 5.2 Overlapping decoding rounds of one 4-windowed sub-blocks. (**a**) Processing schedule with 100% operating efficiency. (**b**) Corresponding active periods of main components

standard parameter in (1.3), there will be fixed outcomes of ρ and (5.34), and the memory controller will become less complicated.

5.2.3 *Application of Overlapping Decoding Rounds*

The attainable improvement in throughput by overlapping decoding rounds is determined by τ_y and $\tau_{a,b,x}$. Here we assume that $\boldsymbol{s}_{s;j}^{\mathcal{Q}}$ or $\boldsymbol{s}_{s;j}^{\mathcal{A}}$ is congruent to $\boldsymbol{s}_{s;j}^{(\rho)}$ modulo 2 and ρ is 0. Besides, we let the SISO decoder first deal with the even-indexed windows and then the odd-indexed windows at every decoding round. Figure 5.2 exemplifies how to rearrange and overlap the processes of 4 windows. The SISO decoder can decode the uncorrelated parts of these constituent codes simultaneously and achieve 100% operating efficiency. Since the execution time for one window group takes $(\tau_y/2 + \tau_{a,b,x})$ cycles, the subsequent decoding round can start $\tau_y/2$ cycles earlier than the completion of current decoding round. However, every function unit will be occupied for τ_y cycles during each round. The finite hardware resources make the period in which two successive decoding rounds

Table 5.2 Comparison between ordinary schedule and overlapping decoding rounds

$L = 32$				$L = 16$			
M	$\eta_{\mathcal{S}}(\%)$	$\eta''_{\mathcal{S}}(\%)$	$\eta''_{\mathcal{S}}/\eta_{\mathcal{S}}$	M	$\eta_{\mathcal{S}}(\%)$	$\eta''_{\mathcal{S}}(\%)$	$\eta''_{\mathcal{S}}/\eta_{\mathcal{S}}$
64	60	86	1.43	32	55	76	1.38
128	75	100	1.33	64	71	100	1.40
192	82	100	1.22	96	79	100	1.27

overlap bounded by $\tau_{a,b,x}$ cycles. Because of this limitation, we have to consider the cases of $\tau_y/2 < \tau_{a,b,x}$ and $\tau_y/2 \geq \tau_{a,b,x}$. If $\tau_y/2$ is less than $\tau_{a,b,x}$, or equivalently $\eta_{\mathcal{S}}$ is less than $2/3$, the cycle number of one decoding round can be regarded as $(\tau_y/2 + \tau_{a,b,x})$; otherwise, it can be regarded as τ_y. The operating efficiency of this design is $\eta''_{\mathcal{S}}$ in (5.44), and the total decoding time for the first $(\ell + 1)$ codewords is $\mathbf{\Gamma}''_{\mathbf{C}_\ell}$ in (5.45) for $\ell \geq 0$.

$$\eta''_{\mathcal{S}} = \begin{cases} \tau_y/(\tau_y/2+\tau_{a,b,x}) & \text{if } \eta_{\mathcal{S}} < 2/3 \\ 100\% & \text{if } \eta_{\mathcal{S}} \geq 2/3 \end{cases} \tag{5.44}$$

$$\mathbf{\Gamma}''_{\mathbf{C}_\ell} = \begin{cases} (\ell+1)\times 2\mathcal{I}\times(\tau_y/2+\tau_{a,b,x})+\tau_y/2 & \text{if } \eta_{\mathcal{S}} < 2/3 \\ (\ell+1)\times 2\mathcal{I}\times\tau_y+\tau_{a,b,x} & \text{if } \eta_{\mathcal{S}} \geq 2/3 \end{cases} \tag{5.45}$$

Both the throughput and latency are improved, and the speedup $(\eta''_{\mathcal{S}}/\eta_{\mathcal{S}})$ is maximized at $\tau_y/2 = \tau_{a,b,x}$. Considering the range of $\tau_{a,b,x}$, when each sub-block contains 2 or 4 windows, the design can approximate the optimal speedup. Based on the examples relating to Fig. 2.7 where $\tau_{a,b,x}$ is $(\mathbf{L} + 10)$, Table 5.2 lists both $\eta_{\mathcal{S}}$ and $\eta''_{\mathcal{S}}$ for $\mathbf{M} = \{2\mathbf{L}, 4\mathbf{L}, 6\mathbf{L}\}$. It is obvious that, for $\mathbf{M} \geq 4\mathbf{L}$, the gain in efficiency decreases, so the schedule with overlapping decoding rounds is more suitable for the SISO decoders that mainly process small sub-blocks.

In addition to the interleaver constraints, each SISO decoder should follow two basic rules, fixed window length and even window number, to allow for this high-efficiency schedule. These rules imply that $\mathbf{M}$ must be an even number and cause the choice of $\mathcal{P}_{\mathcal{S}}$ for each $\mathbf{N}$ narrow. They also prohibit the modification of $\mathbf{M}$ as (3.7). Hence, the schedule with overlapping decoding rounds is inapplicable to the parallel design with modulo mapping. By contrast, the preconditions of these specific QPP and ARP interleavers are consistent with the requirements of both division mapping and barrel-shift network. In consequence, except for the above-mentioned application, these techniques for parallel processing can be compatible with each other.

5.2.4 Performance of Overlapping Decoding Rounds

The overlapping decoding rounds can be put into practice only if the turbo codes with these specific QPP and ARP interleavers could provide satisfactory performance. For both interleavers, there will be numerous parameters fulfilling the proposed constraints, and we must look for those parameters with better error correction capability. The first step is listing the possible case. During the search of QPP interleaver, (f_1, f_2) will be preset to one of ($\mathbf{L}+1, \mathbf{L}$), ($2\mathbf{L}+1, 2\mathbf{L}$), ($\mathbf{L}-1, \mathbf{L}$), and ($2\mathbf{L}-1, 2\mathbf{L}$). It is easy to derive the other parameter sets by adding any multiples of $2\mathbf{L}$ to f_1 or f_2. We follow the same procedure to search valid ARP interleaver parameters and only consider $g_0 = 1$. The initial values of $(\varepsilon, g_1, g_2, g_3)$ can be either $(\mathbf{L} \pm 1, \mathbf{L}+1, 1, \mathbf{L}+1)$ or $(2\mathbf{L} \pm 1, 1, 1, 1)$. Similarly, we can get more sets of $(\varepsilon, g_1, g_2, g_3)$ by adding arbitrary multiples of $2\mathbf{L}$ to them. To reduce the search time, the subsequent step is filtering out some inferior interleavers. We can make an estimate of performance by the spread factor (SF) and minimum distance (MD) [18, 77–79]. The two properties depends on code structure and interleaver, and their values can be computed quickly. If they are too small, the iterative message propagation is likely to be inefficient and lead to insignificant coding gain. Therefore, we can skip these cases and only examine the error rate of the turbo codes with higher SF and MD.

Figures 5.3 and 5.4 show the BER performance of turbo codes with various interleaver parameters. In these simulations, all windows are processed in parallel, and the metrics of each window are initialized with the metrics of its adjacent windows at previous iterations. Tables 5.3 and 5.4 list the corresponding SF and MD. For the examples with the same $\mathbf{N}$, the first set of parameters comes from 3GPP LTE-Advanced or IEEE 802.16 m specifications, and their simulation results are a useful benchmark for judging the quality of other parameters; while the second and third sets can comply with one of the propositions in this section. Compared with the standard cases, some of these constrained parameters can result in better SF and equal MD, and they provide similar performance. However, even with an exhaustive search, several examples still have lower SF or MD, and the worst performance loss is around 0.3 dB at the BER of 10^{-6}. Basically, as $\mathbf{N}$ is larger, it is more possible to find proper interleavers for overlapping decoding rounds. From Fig. 5.3 and Table 5.3, the presented sets of (f_1, f_2) at $\mathbf{N} = 4096$ and $\mathbf{N} = 6144$ are good choices. For ARP interleaver design, Fig. 5.4 and Table 5.4 indicate that the selected sets of $(\varepsilon, g_0, g_1, g_2, g_3)$ at $\mathbf{N} = 2400$ are acceptable.

In addition to error correction capability and contention-free property, the major concerns of the interleavers in 3GPP LTE-Advanced and IEEE 802.16 m standards, the proposed interleaver also takes operating efficiency into account. Since the interleaver design only involves the changes in parameters, it ensures the compatibility with current standards. According to above discussions, the favorable conditions for comparable performance and faster decoding process are large $\mathbf{N}$ and small $\mathbf{M}$ respectively, and they imply the applicability of overlapping decoding

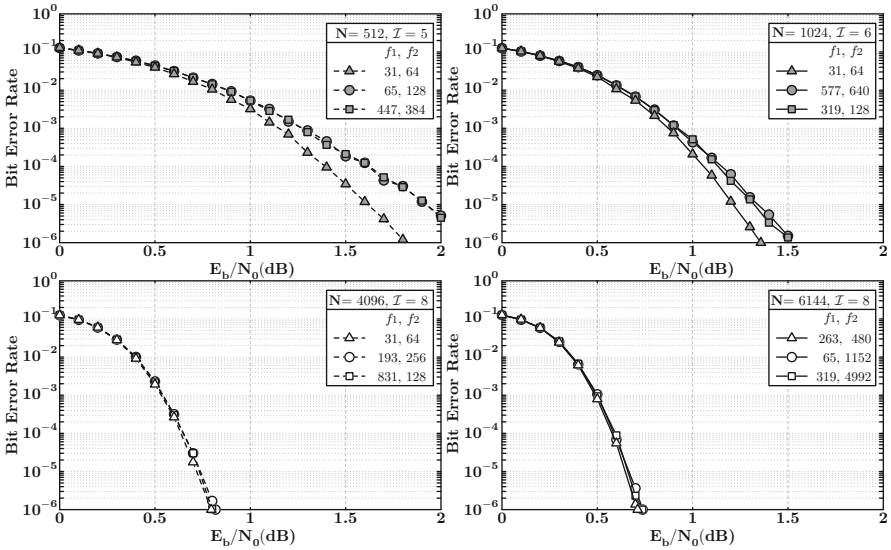

Fig. 5.3 Fixed-point simulation results of turbo codes (tail-biting trellis structure with Fig. 1.4a) using QPP interleavers with different sets of parameters at $\mathbf{N} = \{512, 1024, 4096, 6144\}$: Max-Log-MAP algorithm with $\zeta = 0.75$, $\mathbf{L} = 32$, and $(\varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i)) = (3, 3)$

Table 5.3 Properties of various sets of **N**'s and QPP interleaver parameters

N	f_1	f_2	SF	MD	**N**	f_1	f_2	SF	MD
	31	64	32	33		31	64	32	43
512	65	128	16	31	1024	577	640	32	38
	447	384	16	36		319	128	32	38
	31	64	32	44		263	480	24	50
4096	193	256	64	50	6144	65	1152	64	50
	831	128	64	44		319	4992	64	50

rounds to the turbo decoder with $\mathcal{P}_{\mathcal{S}}$. Moreover, the implementation results in [76] prove the feasibility. Such an effective methodology will be useful for the development of telecommunication systems in next generation.

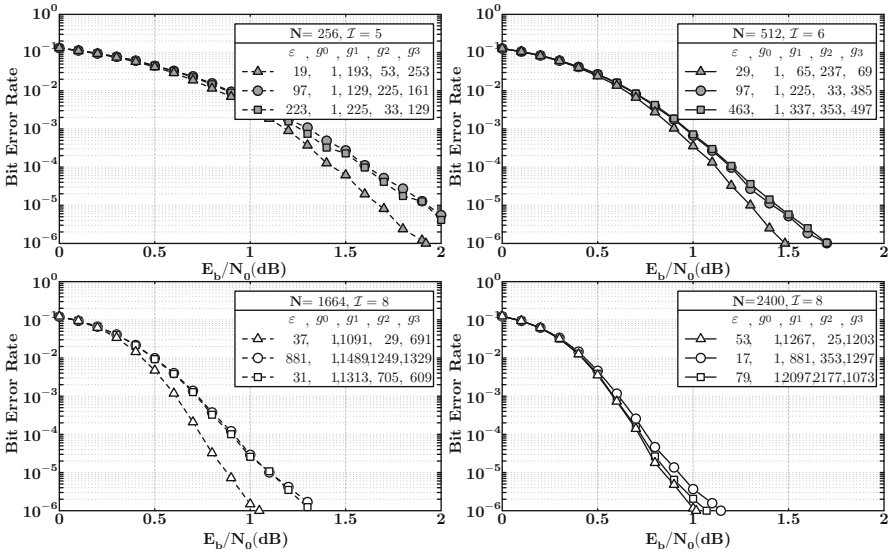

Fig. 5.4 Fixed-point simulation results of turbo codes (tail-biting trellis structure with Fig. 1.4b) using ARP interleavers with different sets of parameters at $\mathbf{N} = \{256, 512, 1664, 2400\}$: Max-Log-MAP algorithm with $\zeta = 0.75$, $\mathbf{L} = 16$, and $(\varrho_{\mathrm{I}}(r_i), \varrho_{\mathrm{F}}(r_i)) = (3, 3)$

Table 5.4 Properties of various sets of **N**'s and ARP interleaver parameters

N	ε	g_0	g_1	g_2	g_3	SF	MD	**N**	ε	g_0	g_1	g_2	g_3	SF	MD
	19	1	193	53	253	16	30		29	1	65	237	69	24	32
256	97	1	129	225	161	16	25	512	97	1	225	33	385	32	26
	223	1	225	33	129	16	30		463	1	337	353	497	28	24
	37	1	1091	29	691	40	36		53	1	1267	25	1203	42	36
1664	881	1	1489	1249	1329	38	36	2400	17	1	881	353	1297	44	36
	31	1	1313	705	609	44	32		79	1	2097	2177	1073	58	36

Bibliography

1. C. E. Shannon, "A mathematical theory of communication (Part I)," *Bell Syst. Tech. J.*, vol. 27, pp. 379–428, Jul. 1948.
2. C. E. Shannon, "A mathematical theory of communication (Part II)," *Bell Syst. Tech.J.*, vol. 27, pp. 623–656, Oct. 1948.
3. R. J. McEliece, *The theory of information and coding*, 2nd ed. Cambridge, UK: Cambridge University Press, 2004.
4. S. Lin and D. J. Costello, Jr., *Error control coding: fundamentals and applications*, 2nd ed. Englewood Cliffs, NJ: Pearson-Hall, 2004.
5. C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *IEEE Proc. Int. Conf. on Communications*, May 1993, pp. 1064–1070.
6. D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar. 1997.
7. D. J. Costello, Jr. and G. D. Forney, Jr., "Channel coding: The road to channel capacity," *Proc. IEEE*, vol. 95, no. 6, pp. 1150–1177, Jun. 2007.
8. B. Vucetic, Y. Li, L. C. Pérez, and F. Jiang, "Recent advances in turbo code design and theory," *Proc. IEEE*, vol. 95, no. 6, pp. 1323–1344, Jun. 2007.
9. K. Gracie and M.-H. Hamon, "Turbo and turbo-like codes: Principles and applications in telecommunications," *Proc. IEEE*, vol. 95, no. 6, pp. 1228–1254, Jun. 2007.
10. *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, European Telecommunications Standard Institute Std. ETSI EN 302 307 v1.2.1, 2009.
11. *Recommendation for Space Data System Standards: TM Synchronization and Channel Coding*, The Consultative Committee for Space Data Systems Std. CCSDS 131.0-B-2, 2011.
12. *Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)*, 3rd Generation Partnership Project Std. TS 25.212 v11.3.0, 2012.
13. *Physical Layer Standard for CDMA2000 Spread Spectrum Systems*, 3rd Generation Partnership Project 2 Std. C.S0002-E, 2011.
14. *Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding*, 3rd Generation Partnership Project Std. TS 36.212 v8.7.0, 2009.
15. *Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding*, 3rd Generation Partnership Project Std. TS 36.212 v11.0.0, 2012.

C.-C. Wong and H.-C. Chang, *Turbo Decoder Architecture for Beyond-4G Applications*, DOI 10.1007/978-1-4614-8310-6,

16. *IEEE Standards for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, Inst. Electrical and Electronics Engineers (IEEE) Std. IEEE 802.16e-2005, 2005.
17. *IEEE Standards for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, Inst. Electrical and Electronics Engineers (IEEE) Std. IEEE 802.16m-2009, 2009.
18. O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.
19. C. Weiß, C. Bettstetter, S. Riedel, and D. J. Costello, Jr., "Turbo decoding with tail-biting trellises," in *IEEE Proc. URSI Int. Symp. on Signals, Systems, and Electronics*, Oct. 1998, pp. 343–348.
20. C. Berrou, Y. Saouter, C. Douillard, S. Kerouédan *et al.*, "Desiging good permutations for turbo codes: toward a single model," in *IEEE Int. Conf. on Communications*, Jun. 2004, pp. 341–345.
21. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.
22. J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.
23. W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data disturbed by time-varying intersymbol interference," in *IEEE Global Telecommunications Conf.*
24. J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1261–1271, Feb./Mar./Apr. 1994.
25. P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE Proc. Int. Conf. on Communications*, Jun. 1995, pp. 1009–1013.
26. L. Papke and P. Robertson, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," in *IEEE Proc. Int. Conf. on Communications*, Jun. 1996, pp. 102–106.
27. J. Vogt and A. Finger, "Improving the Max-Log-MAP turbo decoder," *IET Electronics Letters*, vol. 36, no. 23, pp. 1937–1937, Nov. 2000.
28. S. A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, University of South Australia, 1996.
29. A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.
30. R. Y. Shao, S. Lin, and P. C. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117–1120, Aug. 1999.
31. Y. Wu, B. D. Woener, and W. J. Ebel, "A simple stopping criteria for turbo decoding," *IEEE Commun. Lett.*, vol. 4, no. 8, pp. 258–260, Aug. 2000.
32. R. Asghar, D. Wu, J. Eilert, and D. Liu, "Memory conflict analysis and implementation of a re-configurable interleaver architecture supporting unified parallel turbo decoding," *Journal of Signal Processing Systems*, vol. 60, no. 1, pp. 15–29, Jul. 2010.
33. A. J. Viterbi, "Error bounds for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.
34. J. H. Han, A. T. Erdogan, and T. Arslan, "High speed Max-Log-MAP turbo SISO decoder implementation using branch metric normalization," in *IEEE Computer Society Annual Symp. on VLSI*, 2005, pp. 173–178.
35. I. Lee and J. L. Sonntag, "A new architecture for the fast Viterbi algorithm," *IEEE Trans. Commun.*, vol. 51, no. 10, pp. 1624–1628, Oct. 2003.
36. H. Dawid and H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," in *Sixth IEEE Int. Symp. on Personal, Indoor and Mobile Radio Communications*, Sep. 1996, pp. 193–197.
37. G. Jeong and D. Hsia, "Optimal quantization for soft-decision turbo decoder," in *IEEE Vehicular Tech. Conf.*, Sep. 1999, pp. 1620–1624.

38. A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1220–1222, Nov. 1989.
39. C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *IEEE Proc. Int. Conf. on Communications*, vol. 4, Apr. 1990, pp. 1723–1728.
40. G. Masra, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architecture for turbo codes," *IEEE Trans. VLSI Syst.*, vol. 7, no. 3, pp. 369–379, Sep. 1999.
41. Y. Wu, B. D. Woener, and T. K. Blankenship, "Data width requirements in SISO decoding with modulo normalization," *IEEE Trans. Commun.*, vol. 49, no. 11, pp. 1861–1868, Nov. 2001.
42. G. Masera, M. Mazza, G. Piccinini, F. Viglione *et al.*, "Architectural strategies for low-power VLSI turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 10, no. 3, pp. 279–285, Jun. 2002.
43. S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Commun. Lett.*, vol. 6, no. 7, pp. 288–290, Jul. 2002.
44. Z. He, P. Fortier, and S. Roy, "Highly-parallel decoding architecture for convolutional turbo codes," *IEEE Trans. VLSI Syst.*, vol. 14, no. 10, pp. 1147–1151, Oct. 2006.
45. O. Muller, A. Baghdadi, and M. Jézéquel, "Exploring parallel processing levels for convolutional turbo decoding," in *2nd Information and Communication Technologies*, Apr. 2006, pp. 2353–2358.
46. E. Boutillon, C. Douillard, and G. Montorsi, "Iterative decoding of concatenated convolutional codes: Implementation issues," *Proc. IEEE*, vol. 95, no. 6, pp. 1201–1227, Jun. 2007.
47. M. J. Thul, N. Wehn, and L. P. Rao, "Enabling high-speed turbo decoding through concurrent interleaving," in *IEEE Proc. Int. Symp. on Circuits and Systems*, May 2002, pp. 26–29.
48. M. J. Thul, F. Gilbert, and N. Wehn, "Concurrent interleaving architecture for high-throughput channel coding," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Apr. 2003, pp. 613–616.
49. A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architecture," *IEEE Trans. Inf. Theory*, vol. 50, no. 9, pp. 2002–2009, Sep. 2004.
50. A. Giulietti, L. V. der Perre, and M. Strum, "Parallel turbo coding interleavers: Avoiding collisions in accesses to storage elements," *Elec. Lett.*, vol. 38, no. 5, pp. 232–234, Feb. 2002.
51. Y.-X. Zheng and Y.-T. Su, "A new interleaver design and its application to turbo codes," in *Proc. IEEE Vehicular Technology Conf.*, vol. 3, Sep. 2002, pp. 1437–1441.
52. D. Gnaedig, E. Boutillon, M. Jezequel, V. C. Gaudet *et al.*, "On multiple slice turbo code," in *Proc. 3rd Int. Symp. on Turbo Codes and Related Topics*, Sep. 2003, pp. 343–346.
53. A. Abbasfar and K. Yao, "Interleaver design for high speed turbo decoders," in *IEEE Wireless Communications and Networking Conf.*, Mar. 2004, pp. 1611–1615.
54. L. Dinoi and S. Benedetto, "Variable-size interleaver design for parallel turbo decoder architectures," in *IEEE Global Telecommunications Conf.*, Nov. 2004, pp. 3108–3112.
55. Z. He, S. Roy, and P. Fortier, "High speed and low power design of parallel turbo decoder," in *IEEE Proc. Int. Symp. on Circuits and Systems*, 2005, pp. 6018–6021.
56. T. K. Lee and B.-Z. Shen, "A flexible memory-mpaaing scheme for parallel turbo decoders with periodic interleavers," in *IEEE Int. Symp. on Information Theory*, Jun. 2007, pp. 651–654.
57. R. Dobkin, M. Peleg, and R. Ginosar, "Parallel VLSI architecure for MAP turbo decoder," in *IEEE Int. Symp. on Personal, Indoor and Mobile Radio Communications*, Sep. 2002, pp. 15–18.
58. R. Dobkin, M. Peleg, and R. Ginosar, "Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 13, no. 4, pp. 427–438, Apr. 2005.
59. D. Gnaedig, E. Boutillon, J. Tousch, and M. Jézéquel, "Towards an optimal parallel decoding of turbo codes," in *Proc. 4th Int Symp. on Turbo Codes Related Topics*, Apr. 2006.
60. P. J. Black and T. H. Meng, "A 140 Mb/s, 32-state, radix-4 Viterbi decoder," pp. 70–71, Feb. 1992.

61. M. Bickerstaff, L. Davis, C. Thomas, D. Garrett *et al.*, "A 24Mb/s radix-4 LogMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE Int. Solid-State Circuit Conf.*, Feb. 2003, pp. 151–484.
62. C. C. Lin, C. C. Wu, and C. Y. Lee, "A low power and high speed Viterbi decoder chip for WLAN applications," in *Proc. 29th Europe Solid State Circuits Conf.*, Sep. 2003, pp. 723–726.
63. M. Anders, S. Mathew, R. Krishnamurthy, and S. Borkar, "A 64-state 2GHz 500Mbps 40mW Viterbi accelerator in 90nm CMOS," in *Symp. on VLSI Circuits Digest of Technical Papers*, 2004, pp. 174–175.
64. S. W. Choi and S. S. Choi, "200Mbps Viterbi decoder for UWB," in *Int. Conf. Advanced Communication Tech.*, vol. 2, 2005, pp. 904–907.
65. C.-C. Lin, "Channel decoder design and implementation," Ph.D. dissertation, National Chiao-Tung University, 2006.
66. C.-H. Tang, C.-C. Wong, C.-L. Chen, C.-C. Lin *et al.*, "A 952Mb/s Max-Log MAP decoder chip using radix-4×4 ACS architecture," in *IEEE Asian Solid-State Circuits Conf.*, Nov. 2006, pp. 79–82.
67. C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "A 390Mb/s 3.57mm^2 3GPP-LTE turbo decoder ASIC in 0.13μm CMOS," in *IEEE Int. Solid-State Circuit Conf.*, Feb. 2010, pp. 274–276.
68. C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, "A 188-size 2.1mm^2 reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in *Symp. on VLSI Circuits*, Jun. 2009, pp. 288–289.
69. C.-C. Cheng, Y.-M. Tsai, L.-G. Chen, and A. P. Chanderakasan, "A 0.077 to 0.168 nJ/bit/iteration scalable 3GPP LTE turbo decoder with an adaptive sub-block parallel scheme and an embedded DVFS engine," in *IEEE Custom Integrated Circuits Conf.*, Sep. 2010.
70. J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *IEEE Custom Integrated Circuits Conf.*, Sep. 2009, pp. 487–490.
71. C.-H. Lin, C.-Y. Chen, E.-J. Chang, and A.-Y. Wu, "A 0.16nJ/bit/iteration 3.38mm^2 turbo decoder chip for WiMAX/LTE standards," in *13th Symp. on Integrated Circuits*, Dec. 2011, pp. 168–171.
72. H. Moussa, O. Muller, A. Baghdadi, and M. Jézéquel, "Butterfly and bene-based on-chip communication networks for multiprocessor turbo decoding," in *Design, Automation and Test in Europe Conference and Exhibition*, Apr. 2007, pp. 1–6.
73. C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 1–10, Jan. 2011.
74. C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang *et al.*, "Turbo decoder using contention-free interleaver and parallel architecture," *IEEE J. Solid-State Circuits*, vol. 45, no. 2, pp. 422–432, Feb. 2010.
75. C.-C. Wong and H.-C. Chang, "Reconfigurable turbo decoder with parallel architecture for 3GPP LTE system," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 7, pp. 566–570, Jul. 2010.
76. C.-C. Wong and H.-C. Chang, "High-efficiency processing schedule for parallel turbo decoders using QPP interleaver," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 6, pp. 1412–1420, Jun. 2011.
77. S. Dolinar and D. Divsalar, "Weight distribution of turbo codes using random and nonrandom permutations," Jet Propulsion Lab., TDA Progress Report 42–122, Aug. 1995.
78. S. Crozier, "New high-spread high-distance interleavers for turbo-codes," in *Proc. 20th Biennial Symp. on Communications*, May 2000, pp. 3–7.
79. S. Crozier, P. Guinand, and A. Hunt, "Estimating the minimum distance of turbo-codes using double and triple impulse methods," *IEEE Commun. Lett.*, vol. 9, no. 7, pp. 631–633, Jul. 2005.