Algorithm and VLSI Architecture for Polar Codes Decoder

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

BO YUAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Keshab K. Parhi

July, 2015

Acknowledgements

First of all, I wish to thank my advisor, Professor Keshab K. Parhi, for his guidance and support throughput my Ph. D study at University of Minnesota. I am very grateful for his recognition, his advice, and the opportunities that I have received during the course of my study.

I would like to thank Professor Chris Kim, Professor Marc Riedel and Professor Victor Reiner for their support as members of my Ph. D. committee.

I would like to express my thanks the Graduate School for their financial support with Graduate School Fellowship and Doctoral Dissertation Fellowship. I would also like to thank Broadcom Corporation for supporting this research.

My thanks also go to the current and former members of our research group. Particularly, I would like to thank Dr. Chuan Zhang for our numerous discussions on various research topics. Also, I am very grateful to Dr. Sohini Roychowdury, Dr. Manohar Ayinala, Dr. Yinjie Lao, Dr. Te-Lung Kung, Yin Liu, Tingting Xu and Zisheng Zhang for their support and encouragement during my Ph.D.

This thesis is dedicated to my wonderful family. I am forever grateful to my grandma, my parents, my parents-in-law, my wife and my daughter for their support and encouragement. Without their love, this work would not have been possible.

Abstract

This thesis is devoted to the algorithm and VLSI architecture design for a new class of error-correcting codes, namely polar codes. In particular, efficient algorithm and hardware design for polar codes decoding are the focus of this thesis.

A reduced-latency 2-bit SC decoding algorithm is proposed. Compared to the original SC algorithm that output 1 bit serially, this algorithm can determine 2 bits in one cycle. As a result, the entire decoding latency can be reduced from ($2n$-2) cycles to ($1.5n$-2) cycles without any performance loss for code-length n polar codes. Then, several hardware-level optimizing techniques are applied to further reduce the latency to ($3n/4$-1) cycles. Synthesis results show that the proposed decoder architecture for example (1024, 512) polar codes has significant improvement on throughput and hardware efficiency than the prior works.

Then, a multi-bit decision algorithm, namely 2^Kb-SCL algorithm, is proposed for general SC list decoding cases. This new algorithm can determine 2^K bits at the same time without any performance loss for arbitrary K. As a result, the entire decoding latency can be reduced from $3n$-2 to $n/2^{K-2}$-2 cycles. Then, hardware architecture of the 2^Kb-SCL decoder is developed. In particular, data path balancing technique is developed to enable the proposed architecture operate in a high clock frequency. Synthesis results show that the proposed (1024, 512) 2b-rSCL and 4b-rSCL decoders have significant reduction in latency and throughput than the existing SCL decoders.

Furthermore, this thesis presented the LLR-based SCL decoder to reduce the overall silicon area. The proposed new algorithm, namely LLR-2^Kb-SCL algorithm, can

determine 2^K bits together with the use of LLR messages. As a result, it can achieve both low-complexity and short latency at the same time. Then, the corresponding VLSI architecture of the new SCL decoder is developed. Synthesis results show that the proposed example (1024, 512) LLR-4b-SCL decoder achieves great reduction in both area and latency as compared to the prior LLR or non-LLR-based SCL decoders.

Besides the investigation on SC decoders, this thesis also performs systematic optimization on BP decoders. First, a scaled min-sum (SMS) algorithm is proposed to improve the error-correcting performance. Then, folding and overlapped-scheduling techniques are applied to hardware architecture of BP decoders to reduce area and latency. Moreover, this thesis proposes several novel early-stopping criteria to terminate the iteration of BP decoders earlier. As a result, the overall energy consumption and decoding latency are reduced linearly.

Finally, this thesis presents the stochastic SC decoder. First, a stochastic SC decoding algorithm is derived from the deterministic form. Then, several techniques are applied to the stochastic SC algorithm to avoid the performance loss caused by the stochastic computation. With the use of those approaches, an example (1024, 512) stochastic SC decoder can achieve the similar error-correcting performance with its deterministic counterpart.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 INTRODUCTION

1.1 Introduction

Starting from Shannon's seminal work in 1948 [1], research on channel codes has developed for sixty years. In the past decades, the core mission of coding theory is to discover the new codes that approach Shannon limit closer than the prior codes. Motivated by this goal, information theoreticians have proposed generations of channel codes (see Fig. 1.1). To date, the state-of-the-art channel codes are Turbo codes [2] and LDPC codes [3-4]. As the two codes that are proven to be very close to Shannon limit, Turbo codes and LDPC codes show excellent error-correcting capability, hence they have been widely adopted in numerous IEEE standards and commercial products in communication and storage systems.

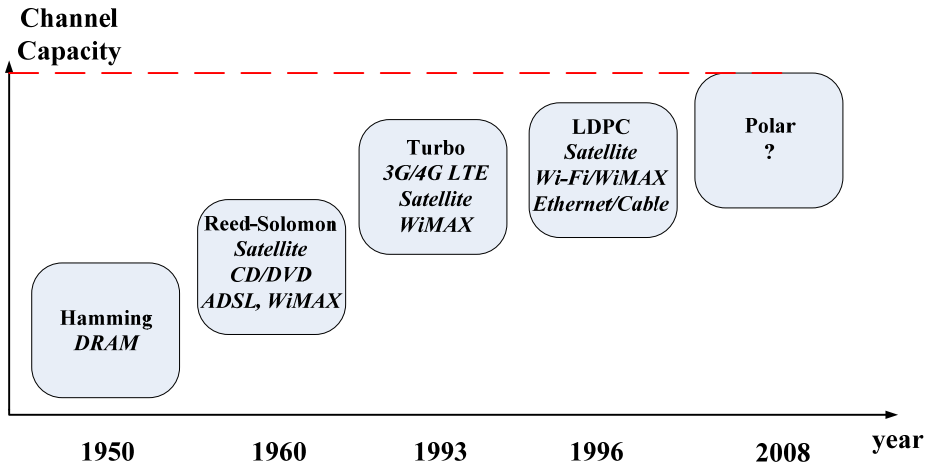


Fig. 1.1. Evolution of channel codes.

Although the existing capacity-approaching codes have already obtained great success in practical applications, the search for the ultimate capacity-achieving codes still continues. In 2008,

polar codes, as the first provable capacity-achieving codes, were discovered [5]. Since the date being introduced, polar codes have received significant attention from coding theory community [6-30]. Compared to the prior best channel codes, polar codes can potentially outperform LDPC codes in terms of error-correcting performance with the similar code-length and code rate [6]. That means polar codes can help build more reliable and robust data transmission. As a result, they are viewed as the competitive channel codes candidate in the next generation of communication and storage systems.

However, the adoption of new channel codes in IEEE standards does not only depend on the error-correcting performance, but also needs to consider the efficient VLSI design of codecs，especially for decoders. To date, inefficient hardware performance of polar codes decoder has become the severe challenges that impede the practical use of polar codes, and these disadvantages on hardware are mainly caused by the inherent property of corresponding decoding algorithms. In general, polar codes can be decoded either with successive cancellation (SC) algorithm [5] or belief propagation (BP) algorithm [13]. For the low-complexity SC and its variant SC list (SCL) algorithms [5-6], they are inherent serial decoding approaches. Hence the corresponding hardware decoders have long latency that is intolerable for real-time transmission. On the other hand, the BP algorithm can be highly parallel processed; but its hardware implementations require a large amount of logic and memory resource, which is unsuitable for embedded area-constraint applications.

1.2 Summary of Contributions

Motivated by the aforementioned challenge on the VLSI design of polar codes decoders, this thesis focuses on designing low-latency high-performance polar codes decoder for next generation of data transmission. In particular, the algorithm/hardware co-optimization is always performed as the core methodology through the entire design procedure. As a result, the hardware

performance of polar codes decoders is significantly improved. Next, the contributions of this thesis are summarized as below.

1.2.1 SC Decoder

SC algorithm is the first invented decoding algorithm for polar codes. Due to its specific decoding procedure that can fully utilize the property of polar encoding, SC algorithm has much lower computation complexity than its BP counterpart; hence SC algorithm is the most popular decoding approach for polar codes. However, the inherent serial decoding manner causes the long latency problems for SC decoder design. In general, for code-length n polar codes, the entire decoding latency for SC decoder is as long as $2n$-2 clock cycles. This extreme long latency impedes the deployment of polar codes in any real-time applications.

In this thesis we propose a reformulated SC decoding algorithm. The proposed approach, namely as *2b-SC* algorithm, can determine 2 bits simultaneously in the same cycle without any error-correcting performance loss. As a result, the entire decoding latency can be reduced from $2n$-2 cycles to $1.5n$-2 cycles. Furthermore, based on the proposed 2b-SC algorithm, we develop *2b-SC-overlapped-scheduling* and *2b-SC-precomputation* architectures for polar codes decoder. With the use of overlapped-scheduling and precomputation techniques, these two architectures can further reduce the decoding latency to n-1 cycles and $0.75n$-1 cycles, respectively. Synthesis results show that the proposed 2b-SC-precomputation decoder for (1024, 512) polar codes can achieve at least 4 times increase in throughput and 40% increase in hardware efficiency than the prior SC decoders.

1.2.2 SC List Decoder

Reduced-Latency SC List Decoder

Although polar codes have capacity-achieving property, their error-correcting performance is inferior to LDPC or Turbo codes in low and medium code-length regions. To address this

problem, in [6] the SC list decoding (SCL) algorithm was proposed to aid polar codes achieve beyond-LDPC performance. However, since an L-size SCL decoder consists of L copies of SC decoders, SCL decoder suffers from long latency problem as well. Even worse, because SCL decoder requires extra operations to sort the $2L$ path metrics, the entire decoding latency of (n, k) SCL decoder is $3n$-2 cycles.

Inspired by our prior proposed 2b-SC algorithm, we propose a reformulated *2b-SCL* algorithm that is targeted for polar list decoding. With the dedicated design for path metric update, this new algorithm can determine 2 bits simultaneously for any code parameters. Furthermore, we extend the 2b-SCL algorithm to a more general 2^K*b-SCL* algorithm, which can determine 2^K bits at the same time for arbitrary K. As a result, the overall latency for SCL decoding is reduced from $3n$-2 to $n/(2^K\text{-}2)$-2 cycles. Based on the proposed algorithm, the corresponding hardware architecture is developed. A data path balancing technique is utilized to reduce the critical path. Compared with a prior SCL decoder, the proposed (1024, 512) 2b-SCL and 4b-SCL decoders can achieve 21% and 60% reduction in latency, 1.66 times and 2.77 times increase in coded throughput with list size 2, and 2.11 times and 3.23 times increase in coded throughput with list size 4, respectively.

Area-Efficient SC List Decoder

To date, the soft-message in the digital transmission system is usually based on the log-likelihood-ratio (LLR) form. However, the original SCL algorithm in [6] was described with the likelihood form. Although in [31-32] the log-likelihood (LL) messages were used to reduce the complexity of the hardware component, the SCL decoders in [31-32] were still based on the non-LLR form, which means they needed a large amount of logic computation and memory resource. In addition, these non-LLR decoders were not compatible for the modern LLR-based digital communication and storage systems.

In this thesis we propose a LLR-based SCL algorithm. With the careful investigation of the inherent procedure of SCL decoding, the path metric update, as the key component of the SCL

decoder, is re-derived in the LLR form. The proposed new decoding approach, namely as *LLR-2^Kb-SCL* algorithm, can achieve both reduction on latency and hardware complexity at the same time. Synthesis results show that for an example (1024, 512) polar code, the proposed LLR-4b-SCL decoders achieve 2.2 times and 2.1 times increase on the hardware efficiency as compared to the state-of-the-art works with list sizes 2 and 4, respectively.

1.2.3 BP Decoder

BP algorithm is a general iterative decoding approach for any channel codes that can be represented by the factor graph [33]. In [13], a polar-oriented BP algorithm was proposed to decode polar codes over their FFT-like factor graph. Different from the serial-decoding SC algorithm, the BP algorithm is inherently parallel, which is very suitable for high-throughput low-latency applications. However, because their computation procedure contains the bi-directional message propagation, BP decoders require much larger amount of energy dissipation and hardware resource than the SC decoders.

In this thesis, we propose several hardware-level techniques to optimize the hardware architecture of BP decoders. With the use of cross-layer overlapping and folding, the latency and complexity of BP decoders are significantly reduced. Furthermore, we propose several early-stopping criteria that can terminate the iteration of BP decoders earlier than the pre-set maximum number. Because the require latency and energy for the iterative BP decoders increase linearly with the number of iterations, the proposed early-stopping strategy can lead to linear reduction on the decoding latency and total energy consumption significantly with negligible performance loss. Synthesis results show that with the use of the proposed stopping criteria, the energy dissipation and average latency of polar (1024, 512) BP decoder can be reduced by 10%~30% with 2%~5% hardware overhead, and average throughput can be increased by 20%~55%.

1.2.4 Stochastic SC Decoder

Different from the conventional computation that uses binary representation, the stochastic computation [34-35] utilizes bit-stream to carry the required information. Here the portion of "1" in each bit stream is the value that the corresponding stream represents. Compared to the conventional binary weighted representation, this even-weighted representation leads significant amenity on error-resilience and low hardware complexity. As a result, to date stochastic computation has been applied to many applications that process the information in the form of probability, such as signal processing [36] and image processing [37].

In this thesis, the behavior of the stochastic SC decoder is investigated for the first time. Starting from the original likelihood-based SC algorithm, the stochastic SC algorithm is derived. Then, several approaches that can potentially improve the error-correcting performance of the stochastic SC decoder are discussed and analyzed.

1.3 Outline of the Thesis

This thesis is outlined as follows. The background of polar codes, including the encoding process and SC algorithm, is introduced in Chapter 2. Afterwards, we propose the 2b-SC decoding algorithm. Then, the hardware architecture and performance of the proposed decoders are presented.

Chapter 3 presents the reduced-latency SCL decoder. 2b-SCL algorithm and its general version as 2^Kb-SCL algorithm are proposed in this chapter. Then, the corresponding hardware architecture and performance are analyzed.

Chapter 4 develops the LLR-based SCL decoder. With the derivation of the LLR-2^Kb-SCL algorithm, hardware architecture is further developed. Then, the hardware performance of the proposed decoder is discussed and compared with the prior works.

Chapter 5 presents the efficient design of BP decoder. First, several optimization techniques are

applied at the hardware-level to reduce latency and complexity. Afterwards, several early-stopping criteria are proposed at the algorithm-level to reduce latency and energy. Then, hardware performance of the optimized BP decoder is presented and discussed.

Chapter 6 develops the stochastic SC decoder.

Chapter 2

2 SC DECODER

In this chapter, we present the reduced-latency SC decoder. Section 2.2 reviews the original SC decoding algorithm. In Section 2.3, a reduced-latency SC decoding algorithm is proposed. Section 2.4 presents the hardware architecture of the proposed 2b-SC decoder. Hardware performance is analyzed and compared with prior works in Section 2.5.

2.1 Introduction

SC algorithm was first proposed in [5]. Different from other general approaches that can be applied to polar codes, SC algorithm fully utilizes the structure of polar codes, and hence it is specifically suitable for decoding polar codes with low computation complexity. To date, SC algorithm has become the most popular decoding approach for polar codes. In particular, one of its variants, referred as SC list decoding algorithm, is viewed as the most promising approach that can make polar codes outperform Turbo codes and LDPC codes. As a result, the investigation of SC algorithm is very important for the research of polar codes.

From the view of hardware design, the main challenge of SC algorithm is the long decoding latency, which results from the inherent serial decoding manner of SC algorithm. In [8], a simplified SC (SSC) algorithm was proposed to reduce the latency. In [38-39], several variant of SSC algorithms and the corresponding hardware architectures were proposed. However, the latency reduction of SSC-class decoders highly depends on the distribution of frozen bits. In particular, when the code rate is not high and code-length is not large, the latency reduction led by the SSC decoder is not significant. Therefore, a general reduce-latency SC algorithm is required for polar codes decoding.

This chapter presents new reduced-latency SC algorithm and architecture. First, a novel reformulation of the last stage of the original SC decoder allows two bits to be decoded in the same clock cycle, which leads to a reduction in latency from (2*n*-2) to (1.5*n*-2). This architecture is referred to as the 2b-SC decoder. Second, the use of overlapped scheduling technique [40] further reduces the latency to (*n*-1). This architecture is referred to as the 2b-SC-Overlapped-scheduling decoder. Third, the use of precomputation [41-43] and look-ahead [42-43] techniques further reduces the latency from (*n*-1) to (3*n*/4-1). This architecture is referred to as the 2b-SC-Precomputation decoder. Synthesis results show that the proposed decoder architecture for (1024, 512) has significant improvement on throughput and hardware efficiency as compared to the prior works.

2.2 SC Decoding Algorithm

2.2.1 Polar Codes

The name "polar" code is derived from the phenomenon of channel polarization. As proved in [5], with efficient construction approach, the reliability of decoded bits will be polarized based on their different positions at the source data. Therefore, an efficient polar-based transmitter can be constructed based on the following principles: 1) sending required information bits at "good" positions, which can strongly guarantee the reliability of transmission; and 2) sending fixed "0" at "bad" positions, since after the transmission any decoded bits at these "bad" positions are highly unreliable. In [5], those "0" bits are called "frozen" bits since these are fixed and their positions are known at both the encoder and the decoder. Similarly, the non-frozen information bits are referred as "free" bits. Accordingly, an (*n*, *k*) polar code contains *k* information ("free") bits and (*n*-*k*) "frozen" bits.

In general, an (*n*, *k*) polar code can be constructed from the original *k*-bit information message $\boldsymbol{c} = c_1^k \triangleq (c_1, c_2, \ldots c_k)$ in two steps. If we denote the set of positions of frozen and free bits as **frz** =

$\{frz_1, \ldots, frz_{n-k}\}$ and **free** = $\{free_1, \ldots, free_k\}$, respectively, where $1 \le frz_i \le n$, $1 \le free_i \le n$, then the first encoding step is to construct an n-bit source data vector as $\boldsymbol{u} = u_1^n \triangleq (u_1, u_2, \ldots u_n)$, where $u_i = c_j$, if $i = free_j$; or $u_i = 0$, if $i \in$ **frz**.

After obtaining $\boldsymbol{u}$, the second step computes the transmitted codeword $\boldsymbol{x} = x_1^n \triangleq (x_1, x_2, \ldots x_n)$ by the generator matrix $\boldsymbol{G}$:

$$\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G} . \tag{2.1}$$

Here $\boldsymbol{G} = \boldsymbol{F}^{\otimes m}$, where $\boldsymbol{F}^{\otimes m}$ denotes the m-th Kronecker power of $\boldsymbol{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$.

It should be noted that in some literature, the mapping from $\boldsymbol{u}$ to $\boldsymbol{x}$ is represented as $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G}\boldsymbol{B}$ instead of (2.1), where $\boldsymbol{B}$ is the bit-reverse operation. As indicated in [5], both of these two mapping approaches are equivalent and have the same performance. In this thesis, we adopt (2.1) as the encoding equation. An example implementation for n=8 polar encoder is illustrated in Fig. 2.1.

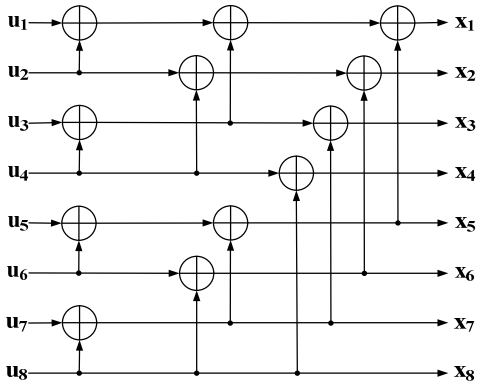


Fig. 2.1. An implementation of polar encoder with n=8.

2.2.2 Original SC Algorithm

At the receiver end, corrupted by the transmission noise, the received codeword will no longer be $\boldsymbol{x}$, but change to $\boldsymbol{y} = y_1^n \triangleq (y_1, y_2, \ldots y_n)$. Since the required information bits are contained in the

original source data vector $\boldsymbol{u}$, the goal of polar decoding is to recover $\boldsymbol{u}$ from $\boldsymbol{y}$. In [5], it is proved that this recovery can be accomplished by the SC algorithm. With a recursive computation procedure, the SC algorithm can use the likelihood ratios (LRs) of $\boldsymbol{y}$ to output an estimated $\boldsymbol{u}$. In this thesis, we denote this estimated $\boldsymbol{u}$ as $\hat{\boldsymbol{u}} = \hat{u}_1^n \triangleq (\hat{u}_1, \hat{u}_2, ..., \hat{u}_n)$. Here each decoded bit $\hat{u}_i$ is determined by the following decision function $\boldsymbol{h}(\bullet)$:

$$\hat{u}_i = \boldsymbol{h}(LR[\boldsymbol{y}, \hat{u}_1^{i-1}]), \tag{2.2}$$

where $\boldsymbol{h}(LR[\boldsymbol{y}, \hat{u}_1^{i-1}]) = 1$ if $LR[\boldsymbol{y}, \hat{u}_1^{i-1}] < 1$ and i is not frozen position; otherwise $\boldsymbol{h}(LR[\boldsymbol{y}, \hat{u}_1^{i-1}]) = 0$.

Here $LR[\boldsymbol{y}, \hat{u}_1^{i-1}] \triangleq \dfrac{W(\boldsymbol{y}, \hat{u}_1^{i-1} \mid u_i = 0)}{W(\boldsymbol{y}, \hat{u}_1^{i-1} \mid u_i = 1)}$ is the LR value of the bit $\hat{u}_i$, and $W(\boldsymbol{y}, \hat{u}_1^{i-1} \mid u_i = s)$ is the conditional probability that the received codeword is $\boldsymbol{y}$ and the previously decoded bits are $\hat{u}_1^{i-1} \triangleq (\hat{u}_1, \hat{u}_2, ..., \hat{u}_{i-1})$, given the condition that $u_i = s \in \{0,1\}$.

From (2.2) it can be seen that the essence of the SC algorithm is how to determine $LR[\boldsymbol{y}, \hat{u}_1^{i-1}]$. In [5], Arıkan proposed an efficient recursive approach to compute these likelihood ratios. Fig. 2.2 shows the decoding procedure for an example n=8 polar code. Based on the LR values of $\boldsymbol{y}$, two types of processing nodes, namely white node (**f** node) and grey node (**g** node), are employed to calculate $LR[\boldsymbol{y}, \hat{u}_1^{i-1}]$. Here $LR[\boldsymbol{y}, \hat{u}_1^{i-1}]$ in stage-3 can be calculated from the messages from stage-2, while the calculation in stage-2 needs the messages output from stage-1. Since these intermediate propagating messages are also LR values, we present a unified notation for all the LRs in this graph. The likelihood ratio output from the node at row i and stage j is denoted as $L(i, j)$. With this new notation, $LR[\boldsymbol{y}, \hat{u}_1^{i-1}]$ is now represented as $L(i, m)$, where m=$\log_2 n$. Meanwhile, the LR value for the received bit y_i can be denoted as $L(i, 0)$. Hence, (2.3) is now expressed as:

$$\hat{u}_i = \boldsymbol{h}(L(i,m)), \tag{2.3}$$

where $\boldsymbol{h}(L(i,m))=1$ if $L(i,m)<1$ and i is not frozen position; otherwise $\boldsymbol{h}(L(i,m))=0$.

To calculate $L(i, m)$, in [5] Arıkan proposed to compute the following equation recursively:

$$L(i,j+1)=\begin{cases}\boldsymbol{f}(L(i,j),L(i+n/2^{j+1},j)) & \text{for } f \text{ node}\\ \boldsymbol{g}(L(i-n/2^{j+1},j),L(i,j),\hat{u}_{sum}) & \text{for } g \text{ node}\end{cases} \tag{2.4}$$

where $\boldsymbol{f}$ and $\boldsymbol{g}$ functions were defined in [5] as:

$$\boldsymbol{f}(a,b)=\frac{1+ab}{a+b} \tag{2.5}$$

$$\boldsymbol{g}(a,b,\hat{u}_{sum})=a^{1-2\hat{u}_{sum}}b. \tag{2.6}$$

Notice that in (2.6), $\hat{u}_{sum}$ is the module-2 sum of partial previous decoded bits. The term $\hat{u}_{sum}$ depicts the "successive" operation in the SC algorithm. The decision of current bit strongly depends on the estimate of previous decoded bits; therefore, the decoded bits can only be computed in a successive manner. To clearly illustrate this phenomenon, we label a specific number for each node in Fig. 2.2. Here each number indicates the index of the clock cycle when the corresponding node is activated. It can be seen that $\hat{u}_1,\hat{u}_2,...,\hat{u}_8$ are output from stage-3 at cycles 3, 4, 6, 7, 10, 11, 13, 14, respectively. Accordingly, this serial decoding leads to an overall latency of 14 cycles. In general, for (n, k) polar code, the latency of SC decoder is ($2n$-2).

(2.3)-(2.6) describe the conventional SC decoding algorithm based on the LR representation. However, because (2.5) and (2.6) contain division and exponentiation operations, they are not attractive for hardware implementation. To solve this problem, a log-likelihood ratio (LLR)-based SC algorithm was proposed in [5] to simplify the hardware design. Accordingly, (2.3)-(2.6) in the natural domain are transformed to (2.7)-(2.10) in the logarithm domain:

$$\hat{u}_i=\boldsymbol{h}(LL(i,m)), \tag{2.7}$$

where $\boldsymbol{h}(LL(i,m))=1$ if $LL(i,m)<0$ and i is not frozen position; otherwise $\boldsymbol{h}(LL(i,m))=0$.

$$LL(i,j+1)=\begin{cases}\boldsymbol{f}(LL(i,j),LL(i+n/2^{j+1},j)) & \text{for } f \text{ node}\\ \boldsymbol{g}(LL(i-n/2^{j+1},j),LL(i,j),\hat{u}_{sum}) & \text{for } g \text{ node.}\end{cases} \tag{2.8}$$

$$\boldsymbol{f}(a,b)=2\tanh^{-1}(\tanh(a/2)\tanh(b/2)) \tag{2.9}$$

$$\boldsymbol{g}(a,b)=a(-1)^{\hat{u}_{sum}}+b\,, \tag{2.10}$$

where LLR value is defined as $LL(i,j)\triangleq\ln(L(i,j))$.

Since (2.9) is still too complex for hardware design, similar to LDPC decoding, a min-sum approximation [44] can be further employed to reduce the complexity of (2.9):

$$\boldsymbol{f}(a,b)\approx sign(a)sign(b)\min(|a|,|b|). \tag{2.11}$$

In general, (2.7)-(2.11) describe the LLR version of the conventional SC algorithm.

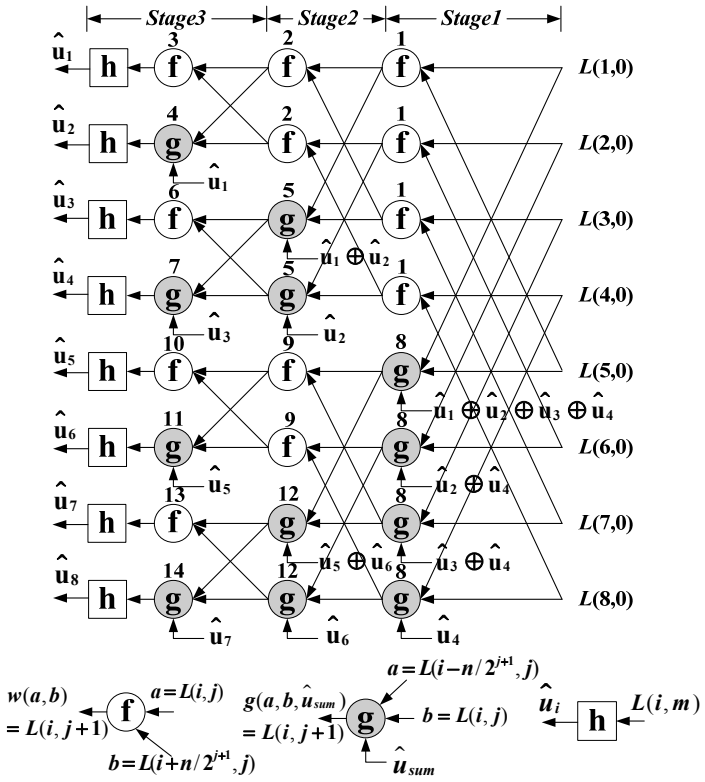


Fig. 2.2. The decoding procedure of SC algorithm with n=8.

2.3 Reduced-latency SC Decoding Algorithm

According to [5], n, the code length of the polar code, should be large enough to guarantee the required error-correcting performance in practical applications. Since the original SC decoder requires ($2n$-2) cycles to output a codeword, the latency with large n is not suitable for real-time high-speed applications. Therefore, design of low-latency polar decoder is an important problem to solve. In this section, using optimization at the algorithm level, we propose a novel reformulation of the last stage of the SC decoding procedure. Then, based on this reformulation, a novel *2-bit-decoding SC* (*2b-SC*) algorithm is presented. This new algorithm can decode two successive bits in the same cycle. Therefore, the latency can be reduced by 25% without any penalty on the performance or hardware complexity.

We now review the original SC algorithm under interpretation of probability. As introduced in Section 2.2, the LR version of the SC algorithm is described by (2.3)-(2.6). Section 2.3.1 reviews the inherent principle of the SC algorithm in detail. This review is helpful in developing the new reduced-latency 2b-SC algorithm in Section 2.3.2.

2.3.1 Review of SC Algorithm under interpretation of probability

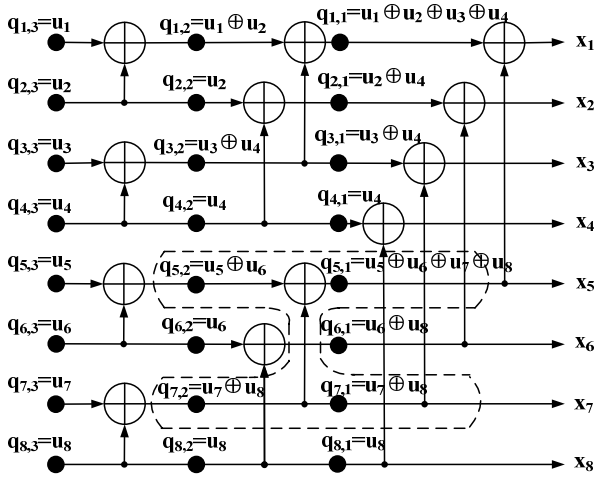
As indicated in [5] [13], the architectures of polar encoder (Fig. 2.1) and decoder (Fig. 2.2) can be re-defined in a unified framework. Fig. 2.3 illustrates this unified encoding/decoding architecture for n=8. Under this framework, the encoding procedure can be viewed as a left-to-right transformation from u_1^n to x_1^n. As shown in Fig. 2.3(a), this transformation is accomplished by computing intermediate value $q_{i,j}$. Similarly, when the probabilities of y_1^n are available at the right side of this architecture (Fig. 2.3(c)), the decoding procedure can be viewed as estimating those intermediate $q_{i,j}$ in the right-to-left direction. These estimated values, denoted as $\hat{q}_{i,j}$, will be finally used to calculate the leftmost $\hat{u}_1^n$, which is just the estimation of u_1^n.

Fig. 2.3 (b) and Fig. 2.3 (d) show the basic computation units of the overall architecture. For polar encoding, each unit represents an exclusive-or operation, while for decoding it represents the combination of ***f*** and ***g*** functions. When the unified architecture is in encoding phase, as shown in Fig. 2.3(b), it is easy to compute the outputs of the basic unit (denoted as *c* and *d*) from inputs (denoted as *a* and *b*) as:

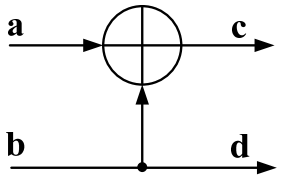
$$c = a \oplus b \text{ and } d = b. \quad . \tag{2.12}$$

On the other hand, when the unified architecture is in decoding phase, since SC decoding is just the right-to-left estimation procedure for those $q_{i,j}$ (see Fig. 2.3(c)), we can derive the expected relationship between these estimated values in Fig. 2.3(d) as:

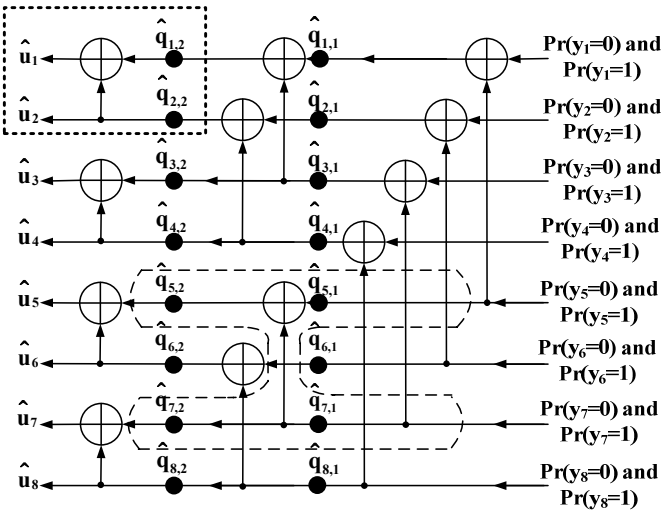
$$\hat{a} = \hat{c} \oplus \hat{d} \text{ and } \quad \hat{b} = \hat{d} . \tag{2.13}$$



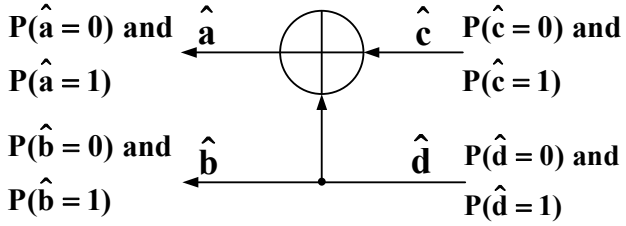
(a)



(b)



(c)



(d)

Fig. 2.3. Unified polar encoding/decoding architecture with n=8.
(a) left to right encoding procedure. (b) basic encoding computation unit. (c) right to left decoding procedure. (d) basic decoding computation unit.

It should be noted that (2.13) cannot be directly used to estimate $\hat{a}$ and $\hat{b}$. This is because the "soft" bit probability, instead of "hard" bit value, is employed in the soft-decision SC decoding. For example, in Fig. 2.3(d), the probability of $\hat{c}$ and $\hat{d}$ are the inputs of the basic unit to compute probability of $\hat{a}$ and $\hat{b}$. Therefore, (2.13) is only a "guideline" that depicts the "expected" relationship between $\hat{a},\hat{b}$ and $\hat{c},\hat{d}$. Next we will show how to exactly calculate the probability of $\hat{a}$ and $\hat{b}$ with the use of (2.13).

Now consider the probability of $\hat{a}$ denoted as $\mathrm{P}(\hat{a}=s) \triangleq \Pr(\hat{a}=s,\hat{u}_1^{i-1} \mid \boldsymbol{y})$ where $s \in \{0,1\}$. Notice in

the case that $\hat{a} = 0$, according to (2.13), there are two possible combinations of $\hat{c}$ and $\hat{d}$ that can make $\hat{a}$ equal to 0: $\hat{c} = 0$, $\hat{d} = 0$ or $\hat{c} = 1$, $\hat{d} = 1$.

Therefore, the probability for $\hat{a} = 0$ is given by:

$$\mathrm{P}(\hat{a} = 0) = \mathrm{P}(\hat{c} = 0)\,\mathrm{P}(\hat{d} = 0) + \mathrm{P}(\hat{c} = 1)\,\mathrm{P}(\hat{d} = 1). \tag{2.14}$$

Similarly, for the case $\hat{a} = 1,$ we have

$$\mathrm{P}(\hat{a} = 1) = \mathrm{P}(\hat{c} = 0)\,\mathrm{P}(\hat{d} = 1) + \mathrm{P}(\hat{c} = 1)\,\mathrm{P}(\hat{d} = 0). \tag{2.15}$$

Denote the likelihood ratio of $\hat{a}$ as $LR(\hat{a})$, since $LR(\hat{a}) = \dfrac{\mathrm{P}(\hat{a} = 0)}{\mathrm{P}(\hat{a} = 1)}$, using (2.14) and (2.15), we have:

$$LR(\hat{a}) = \frac{\mathrm{P}(\hat{a} = 0)}{\mathrm{P}(\hat{a} = 1)} = \frac{\mathrm{P}(\hat{c} = 0)\,\mathrm{P}(\hat{d} = 0) + \mathrm{P}(\hat{c} = 1)\,\mathrm{P}(\hat{d} = 1)}{\mathrm{P}(\hat{c} = 0)\,\mathrm{P}(\hat{d} = 1) + \mathrm{P}(\hat{c} = 1)\,\mathrm{P}(\hat{d} = 0)} = \frac{1 + LR(\hat{c})LR(\hat{d})}{LR(\hat{c}) + LR(\hat{d})}, \tag{2.16}$$

where $LR(\hat{c}) = \dfrac{\mathrm{P}(\hat{c} = 0)}{\mathrm{P}(\hat{c} = 1)}$ and $LR(\hat{d}) = \dfrac{\mathrm{P}(\hat{d} = 0)}{\mathrm{P}(\hat{d} = 1)}$.

Notice that (2.16) is just the same as (2.4) and (2.5), where $LR(\hat{a}) = L(i, j+1)$, $LR(\hat{c}) = L(i, j)$ and $LR(\hat{d}) = L(i + n/2^{j+1}, j)$. This completes the derivation of the LR version of the ***f*** function based on bit probability representation and (2.13). Next we show how to derive LR version of the ***g*** function, which is equivalent to the calculation of probability of $\hat{b}$.

Due to the successive computation of the SC algorithm, the probability of $\hat{b}$ being 0 or 1 depends on the decision of $\hat{a}$. In the case $\hat{a} = 0$, in order to make $\hat{b}$ equal to 0, the combination of $\hat{c}$ and $\hat{d}$ can only be $\hat{c} = 0$ and $\hat{d} = 0$. Therefore, if we denote $\mathrm{P}(\hat{a} = s_1, \hat{b} = s_2) \triangleq \Pr(\hat{a} = s_1, \hat{b} = s_2, \hat{u}_1^{i-1} \mid \mathbf{y})$ where $s_1, s_2 \in \{0,1\}$, then we have:

$$\mathrm{P}(\hat{a} = 0, \hat{b} = 0) = \mathrm{P}(\hat{c} = 0)\,\mathrm{P}(\hat{d} = 0). \tag{2.17}$$

Similarly, in order to make $\hat{b}$ equal to 1 under the condition that $\hat{a} = 0$, the combination of $\hat{c}$ and

$\hat{d}$ can only be $\hat{c}=1$ and $\hat{d}=1$. Thus, we have:

$$\mathrm{P}(\hat{a}=0,\hat{b}=1)=\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1). \tag{2.18}$$

Based on (2.17) and (2.18), we can obtain the likelihood ratio of $\hat{b}$ for the case $\hat{a}=0$:

$$LR_{\hat{a}=0}(\hat{b})=\frac{\mathrm{P}(\hat{a}=0,\hat{b}=0)}{\mathrm{P}(\hat{a}=0,\hat{b}=1)}=\frac{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0)}{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1)}=LR(\hat{c})LR(\hat{d}). \tag{2.19}$$

Now consider the probability of $\hat{b}$ when $\hat{a}=1$. In this case, for $\hat{b}$ to be 0, $\hat{c}=1$ and $\hat{d}=0$. Thus:

$$\mathrm{P}(\hat{a}=1,\hat{b}=0)=\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0). \tag{2.20}$$

Similarly, to make $\hat{b}$ equal to 1 when $\hat{a}=1$, the only combination of $\hat{c}$ and $\hat{d}$ is $\hat{c}=0$ and $\hat{d}=1$. Hence:

$$\mathrm{P}(\hat{a}=1,\hat{b}=1)=\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1). \tag{2.21}$$

Based on (2.20) and (2.21), we have:

$$LR_{\hat{a}=1}(\hat{b})=\frac{\mathrm{P}(\hat{a}=1,\hat{b}=0)}{\mathrm{P}(\hat{a}=1,\hat{b}=1)}=\frac{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)}{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1)}=LR(\hat{c})^{-1}LR(\hat{d}) \tag{2.22}$$

We can derive a unified representation for LR value of $\hat{b}$ for different conditions of $\hat{a}$ from (2.19) and (2.22) as:

$$LR(\hat{b})=LR(\hat{c})^{1-2\hat{a}}LR(\hat{d}). \tag{2.23}$$

It can be seen that (2.23) is the same as (2.4) and (2.6), where $LR(\hat{b})=L(i,j+1)$, $LR(\hat{d})=L(i,j)$, $LR(\hat{c})=L(i-n/2^{j+1},j)$, and $\hat{a}=\hat{u}_{sum}$. Therefore, the LR version of $\boldsymbol{g}$ function can also be derived from (2.13). Note that here the equality of $\hat{a}$ and $\hat{u}_{sum}$ can be easily verified by examining the estimation characteristics of the decoding procedure. For example, in the dashed unit of Fig. 2.3(c), the corresponding $\hat{a}$ is $\hat{q}_{5,2}$, which is the estimation of $q_{5,2}=u_5\oplus u_6$ (Fig. 2.3(a)). Therefore, $\hat{q}_{5,2}$ is equal to $\hat{u}_5\oplus\hat{u}_6$, which is just $\hat{u}_{sum}$ of index-12 node in Fig. 2.2.

In this section, starting from (2.13), we have shown how the LR version of the SC algorithm can be derived under the interpretation of bit probability. This forms the basis of the new 2b-SC algorithm developed in the next section.

2.3.2 2b-SC Algorithm

Section 2.3.1 discusses the general computation unit of Fig. 2.3(c). In this section, we focus on those units on the leftmost side (the last stage) of Fig. 2.3(c), which compute the decoded bits $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ (see Fig. 2.4). One of these units is highlighted with dotted rectangular line at the top left of Fig. 2.3(c).

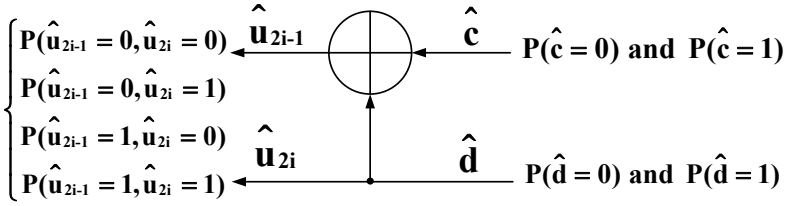


Fig. 2.4. The computation unit of the last stage of the decoder.

According to (2.3), the value of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ depend not only on their LR values, but also on whether they are frozen bits or not. Therefore, since $\hat{u}_{2i-1}$ or $\hat{u}_{2i}$ can be either a free or frozen bit, we discuss four possible cases based on different frozen conditions of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$.

Case-1*: *None of $\hat{u}_{2i-1}$ or $\hat{u}_{2i}$ is a frozen bit

In this case, since none of $\hat{u}_{2i-1}$ or $\hat{u}_{2i}$ is frozen, its value is completely determined by the probability that it is 0 or 1. Therefore, according to (2.17)(2.18)(2.20)(2.21), the probabilities of different combinations of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ can be expressed as follows:

$$\begin{aligned}
&\mathrm{P}(00) \triangleq \mathrm{P}(\hat{u}_{2i-1}=0,\hat{u}_{2i}=0) = \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0)\\
&\mathrm{P}(01) \triangleq \mathrm{P}(\hat{u}_{2i-1}=0,\hat{u}_{2i}=1) = \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1)\\
&\mathrm{P}(10) \triangleq \mathrm{P}(\hat{u}_{2i-1}=1,\hat{u}_{2i}=0) = \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)\\
&\mathrm{P}(11) \triangleq \mathrm{P}(\hat{u}_{2i-1}=1,\hat{u}_{2i}=1) = \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1).
\end{aligned} \tag{2.24}$$

Recall that in the SC algorithm, the value of the unfrozen bit $\hat{u}_j$ is determined by comparing $\mathrm{P}(\hat{u}_j = 0)$ and $\mathrm{P}(\hat{u}_j = 1)$. (2.24) describes the joint probabilities of different combinations of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ and is the key to decoding two successive bits. Thus, $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ can be directly determined by finding the largest one among the four joint probabilities in (2.24).

The above hypothesis leads to two benefits. First, since a pair of bits, instead of a single bit, is determined each time, one clock cycle is saved. Considering the whole decoding procedure, this approach reduces the latency by 25%. Second, because we only need to find the largest one among four probabilities, the hardware complexity will be much less than that of the original **f** and **g** nodes. In summary, if the validity of the proposed approach can be verified, it will improve the hardware performance with respect to both latency and hardware complexity.

Motivated by the potential advantage of this hypothesis, we explore its validity. Fortunately, the proposed hypothesis is proved to be valid, and it can be verified that the decoded bit values determined by this approach are strictly equal to the outputs from the conventional SC algorithm. Therefore, we formalize this hypothesis to a proposition as follows:

***Proposition*1:** *For arbitrary polar codes, assume the largest joint probability in (2.24) is* $\mathrm{P}(\alpha\beta)$*, and unfrozen decoded bits output from the original SC algorithm are* $\hat{u}_{2i-1}$ *and* $\hat{u}_{2i}$*. Then* $\hat{u}_{2i-1} = \alpha$ *and* $\hat{u}_{2i} = \beta$*.*

***Proof*:** This proposition is proved in the Appendix. □

As mentioned in the above paragraph, since the proposed hypothesis has been proved, we can obtain a fast approach to simultaneously determine unfrozen $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$: *Given the probabilities of* $\hat{c}$ *and* $\hat{d}$*, once the largest joint probability* $\mathrm{P}(\alpha\beta)$ *in (2.24) is found,* $\hat{u}_{2i-1}$ *and* $\hat{u}_{2i}$ *are immediately determined as* $\hat{u}_{2i-1} = \alpha$ *and* $\hat{u}_{2i} = \beta$*.*

In practical applications, likelihood ratio, instead of probability, is used for representing soft

information. Therefore, the probability-based (2.24) needs to be transformed to LR-based form:

$$\begin{aligned} LR(00) &\triangleq \mathrm{P}(00)/\mathrm{P}(01) = LR(\hat{c})LR(\hat{d}) \\ LR(01) &\triangleq \mathrm{P}(01)/\mathrm{P}(01) = 1 \\ LR(10) &\triangleq \mathrm{P}(10)/\mathrm{P}(01) = LR(\hat{d}) \\ LR(11) &\triangleq \mathrm{P}(11)/\mathrm{P}(01) = LR(\hat{c}). \end{aligned} \tag{2.25}$$

To avoid potential overflow and reduce computation complexity, (2.25) is further transformed to the logarithm domain:

$$\begin{aligned} LLR(00) &\triangleq LLR(\hat{c}) + LLR(\hat{d}) \qquad LLR(01) \triangleq 0 \\ LLR(10) &\triangleq LLR(\hat{d}) \qquad\qquad\quad LLR(11) \triangleq LLR(\hat{c}). \end{aligned} \tag{2.26}$$

In the remainder of this section, we will use LLR-based equation (2.26) to describe the new algorithm and its hardware architectures.

Case-2*: *Both $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ are frozen bits

In this case, since both of these two bits are frozen, their values can be directly determined as 0.

Case-3*: *Only $\hat{u}_{2i-1}$ is frozen bit

When $\hat{u}_{2i-1}$ is frozen, $\hat{u}_{2i-1}$=0. Then, according to (2.23), we have

$$LR(\hat{u}_{2i}) = LR(\hat{c})^{1-2\hat{u}_{2i-1}} LR(\hat{d}) = LR(\hat{c})LR(\hat{d}) . \tag{2.27}$$

Under the representation of LLR, (2.27) becomes

$$LLR(\hat{u}_{2i}) = LLR(\hat{c})(-1)^{\hat{u}_{2i-1}} + LLR(\hat{d}) = LLR(\hat{c}) + LLR(\hat{d}) . \tag{2.28}$$

Therefore, the decision scheme for $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ in this case is

$$\begin{aligned} &\hat{u}_{2i-1} = 0 \\ &\hat{u}_{2i} = \begin{cases} 0 \;\; if \;\; LLR(\hat{c}) + LLR(\hat{d}) \geq 0 \\ 1 \;\; if \;\; LLR(\hat{c}) + LLR(\hat{d}) < 0. \end{cases} \end{aligned} \tag{2.29}$$

Case-4*: *Only $\hat{u}_{2i}$ is frozen bit

When $\hat{u}_{2i}$ is frozen bit, $\hat{u}_{2i}$=0. According to (2.16), we have:

$$LR(\hat{u}_{2i-1}) = \frac{1 + LR(\hat{c})LR(\hat{d})}{LR(\hat{c}) LR(\hat{d})}. \tag{2.30}$$

Under the representation of LLR, (2.30) becomes

$$LLR(\hat{u}_{2i-1}) = 2\tanh^{-1}(\tanh(LR(\hat{c})/2)\tanh(LR(\hat{d})/2)). \tag{2.31}$$

With min-sum approximation, we have:

$$LLR(\hat{u}_{2i}) \approx sign(LLR(\hat{c}))sign(LLR(\hat{d}))\min(\left|LLR(\hat{c})\right|, \left|LLR(\hat{d})\right|). \tag{2.32}$$

Therefore, decision scheme for $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ in this case is

$$\begin{aligned} &\hat{u}_{2i} = 0 \\ &\hat{u}_{2i-1} = \begin{cases} 0 & sign(LLR(\hat{c}))sign(LLR(\hat{d})) \geq 0 \\ 1 & sign(LLR(\hat{c}))sign(LLR(\hat{d})) < 0. \end{cases} \end{aligned} \tag{2.33}$$

Summarizing the above four cases, it can be seen that $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ can always be determined at the same time. This leads to the decision scheme (Scheme 2.1) for the last stage of SC decoding.

With the proposed reformulated scheme, the corresponding *2b-SC algorithm* can be developed. Fig. 2.5 shows the corresponding 2b-SC decoding procedure with the same n=8 polar code in Fig. 2.2. Compared with the conventional SC scheme in Fig. 2.2, the proposed 2b-SC algorithm replaces the **f** and **g** nodes at stage-3 with new **p** nodes. The **p** node, whose function is described in Scheme 2.1, can output the successive $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ at the same time. Therefore, the overall latency is reduced. For example, the original latency of 14 cycles in Fig. 2.2 is now reduced to 10 cycles in Fig. 2.5. Tables 2.1 and 2.2 describe the timing information of the conventional SC and 2b-SC algorithms in detail. The original SC algorithm requires n=8 cycles in stage-3 to output $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$. By employing **p** nodes to compute the decoded bits, n/2=4 cycles are saved by the 2b-SC algorithm. In general, compared with the original SC algorithm, the overall latency of 2b-SC algorithm is reduced from (2n-2) to (1.5n-2).

Scheme 2.1: Reformulation for last stage (stage-m) computation in SC decoding

1: Input: Log-Likelihood ratios $LLR(\hat{c})$ **and** $LLR(\hat{d})$ **from stage-**$(m-1)$

2: Judge $\hat{u}_{2i-1}$ **and** $\hat{u}_{2i}$ **are frozen bits or not**

3: Case1: None of $\hat{u}_{2i-1}$ **or** $\hat{u}_{2i}$ **is a frozen bit**

4: Find the largest element among $\{LLR(\hat{c})+LLR(\hat{d}), 0, LLR(\hat{d}), LLR(\hat{c})\}$

5: If the largest element is $LLR(\hat{c})+LLR(\hat{d})$: $\hat{u}_{2i-1}=0, \hat{u}_{2i}=0$

6: If the largest element is 0: $\hat{u}_{2i-1}=0, \hat{u}_{2i}=1$

7: If the largest element is $LLR(\hat{d})$: $\hat{u}_{2i-1}=1, \hat{u}_{2i}=0$

8: If the largest element is $LLR(\hat{c})$: $\hat{u}_{2i-1}=1, \hat{u}_{2i}=1$

9: Case2: Both $\hat{u}_{2i-1}$ **and** $\hat{u}_{2i}$ **are frozen bits**

10: $\hat{u}_{2i-1}=0, \hat{u}_{2i}=0$

11: Case3: Only $\hat{u}_{2i-1}$ **is frozen bit**

12: $\hat{u}_{2i-1}=0$

13:
$$\hat{u}_{2i}=\begin{cases}0 & \text{if } LLR(\hat{c})+LLR(\hat{d})\geq 0\\ 1 & \text{if } LLR(\hat{c})+LLR(\hat{d})<0\end{cases}$$

14: Case4: Only $\hat{u}_{2i}$ **is frozen bit**

15:
$$\hat{u}_{2i-1}=\begin{cases}0 & sign(LLR(\hat{c}))sign(LLR(\hat{d}))\geq 0\\ 1 & sign(LLR(\hat{c}))sign(LLR(\hat{d}))<0\end{cases}$$

16: $\hat{u}_{2i}=0$

17: Output: $\hat{u}_{2i-1}$, $\hat{u}_{2i}$

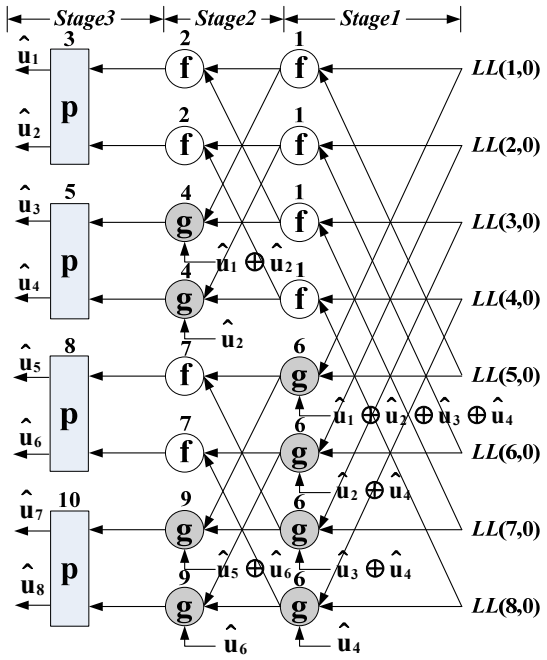


Fig. 2.5. The decoding procedure of 2b-SC algorithm with n=8.

Table 2.1. Decoding scheme of conventional SC for n=8 polar code

Conventional SC decoding scheme [44]														
Clock cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Stage1	**f**							**g**						
Stage2		**f**			**g**				**f**			**g**		
Stage3			**f**	**g**		**f**	**g**			**f**	**g**		**f**	**g**
Output			$\hat{u}_1$	$\hat{u}_2$		$\hat{u}_3$	$\hat{u}_4$			$\hat{u}_5$	$\hat{u}_6$		$\hat{u}_7$	$\hat{u}_8$

Table 2.2. Decoding scheme of 2B-SC algorithm for n=8 polar code

2-bit-decoding SC (2b-SC) scheme										
Clock cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Stage1	**f**					**g**				
Stage2		**f**		**g**			**f**		**g**	
Stage3			**p**		**p**			**p**		**p**
Output			$\hat{u}_1$ & $\hat{u}_2$		$\hat{u}_3$ & $\hat{u}_4$			$\hat{u}_5$ & $\hat{u}_6$		$\hat{u}_7$ & $\hat{u}_8$

2.4 Hardware Architecture

In this section, three hardware architectures of the new 2b-SC algorithm are presented. According to Fig. 2.5, the overall 2b-SC decoder mainly consists of three types of processing nodes: **f, g** and **p** nodes. Besides these nodes, a simple partial sum generator (PSG) is also needed to generate partial sum $\hat{u}_{sum}$. Since PSG block is similar to polar encoder with simple architecture, therefore in this section we focus on the architectures of **f**, **g** and **p** nodes.

2.4.1 Processing Elements

As shown in Fig. 2.5, **p** nodes are used in stage-m, and **f** and **g** nodes are used in other stages to calculate the propagated LLR values. For simplicity of hardware design, the functions of **f** and **g** nodes are always implemented by unified processing elements (PEs). Fig. 2.6 shows the architecture of this PE based on the LLR version of (2.16) and (2.23) with min-sum approximation. Here S2C is the block that performs the conversion from sign-magnitude form to 2's complement form, while C2S unit carries out the inverse conversion. Additionally, adder and

subtractor are employed to carry out addition and subtraction between the two inputs. The corresponding sum and difference are selected by the partial sum signal $\hat{u}_{sum}$ from the PSG block. Finally, at the output end of the PE, *control* signal is used to determine the output as $LLR(\hat{a})$ or $LLR(\hat{b})$, which is propagated to the next stage. In summary, the architecture shown in Fig. 2.6 mainly consists of one comparator-selector, one adder, one subtractor, two multiplexers, two C2S and two S2C blocks. Accordingly, the critical path delay of PE is $T_{S2C}+T_{adder}+T_{C2S}+2T_{MUX}$.

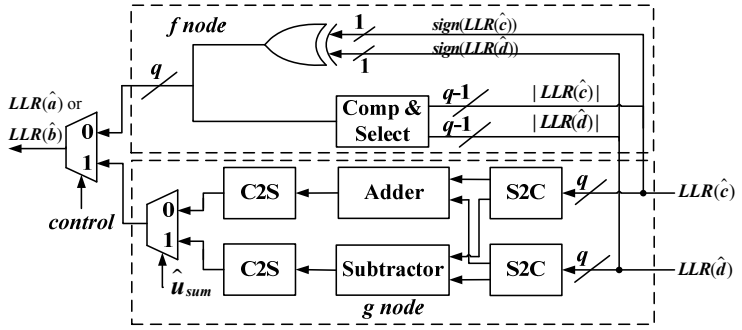


Fig. 2.6. The architecture of PE for **f** and **g** nodes.

2.4.2 p node

In Scheme 2.1, the decision scheme in **p** node has been described based on the LLR representation. To implement its function, a straightforward approach is to employ a sorting circuit and a signed adder. However, this method is too complex and is not hardware-efficient. After careful examination of Scheme 2.1, we observe that the **p** node can be implemented with a very simple method, which is described as below.

First, since the function of **p** node depends on the frozen conditions of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$, signals *frozen*1 and *frozen*2 are introduced to indicate whether $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ are frozen bits or not. If $\hat{u}_{2i-1}$ is frozen, *frozen*1 will be 1, otherwise 0. Similarly, *frozen*2 will be 1 or 0 when $\hat{u}_{2i}$ is frozen or not. Secondly, the sign bits of $LLR(\hat{c})$ and $LLR(\hat{d})$ are employed for simplifying computations. Denoted as $sign(LLR(\hat{c}))$ and $sign(LLR(\hat{d}))$, these sign bits will be, respectively, 0 or 1 when the

corresponding LLR values are non-negative or negative. Furthermore, the *comp* signal, which is the result of comparison between absolute value of $LLR(\hat{c})$ and $LLR(\hat{d})$, is also employed. When $\left|LLR(\hat{c})\right| \geq \left|LLR(\hat{d})\right|$, *comp* will be 1, otherwise 0. Accordingly, with the above five signals, we can obtain the truth table shown in Table 2.3 for $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ based on Scheme 2.1.

Table 2.3. The truth table of **p** node

Input					**Output**	
*frozen*1	*frozen*2	$sign(LLR(\hat{c}))$	$sign(LLR(\hat{d}))$	*comp*	$\hat{u}_{2i-1}$	$\hat{u}_{2i}$
0	0	0	0	don't care	0	0
		1	1	don't care	0	1
		1	0	don't care	1	0
		0	1	don't care	1	1
1	1	don't care	don't care	don't care	0	0
1	0	1	1	don't care	0	1
		1	0	1		1
		1	0	0		0
		0	1	0		1
		0	1	1		0
		0	0	don't care		0
0	1	0	0	don't care	0	0
		1	1	don't care	0	
		0	1	don't care	1	
		1	0	don't care	1	

Then, with the help of above truth table, Boolean expression of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ can be derived as follows:

$$\hat{u}_{2i-1} = \overline{frozen1}(sign(LLR(\hat{c}) \oplus sign(LLR(\hat{d})) \tag{2.34}$$

$$\begin{aligned} \hat{u}_{2i} = {} & \overline{comp}\ \overline{frozen2}\ sign(LLR(\hat{d})) \\ & + comp\ \overline{frozen1}\ \overline{frozen2}\ sign(LLR(\hat{d})) \\ & + comp\ frozen1\ \overline{frozen2}\ sign(LLR(\hat{c})). \end{aligned} \tag{2.35}$$

Based on (2.34) and (2.35), a hardware architecture of the **p** node with q-bit quantization is shown in Fig. 2.7. Here $LLR(\hat{c})$ and $LLR(\hat{d})$ are represented in sign-magnitude (SM) form, and they are output from the **f** and **g** nodes in stage-(m-1). In addition, since the frozen conditions of

$\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ have been pre-determined before the transmission, signals *frozen*1 and *frozen*2 can be easily obtained from the control unit.

It can be seen that the circuit of **p** node in Fig. 2.7 is much simpler than that of the PE in Fig. 2.6. This leads to two benefits. First, since all the **f** and **g** nodes in stage-*m* are replaced by **p** nodes, the hardware complexity of stage-*m* in 2b-SC decoder (Fig. 2.5) is less than the original SC decoder (Fig. 2.2). Second, because the critical path delay of **p** node is only T_{comp}+2T_{AND}+2T_{OR}, which is much shorter than that of the PE, the latency can be further reduced from (1.5*n*-2) to (*n*-1) as discussed in Section 2.4.4.

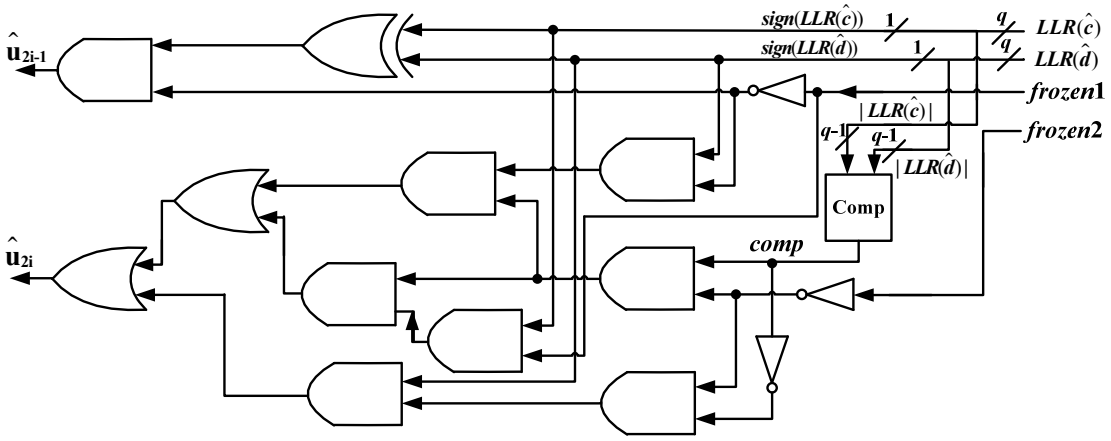


Fig. 2.7. The architecture of **p** node.

2.4.3 Overall Architecture of 2b-SC Decoder

Based on the circuits of the PE and the **p** node in Fig. 2.6 and Fig. 2.7, respectively, the overall 2b-SC decoder can be constructed as a butterfly-like architecture (Fig. 2.5). However, this straightforward design is not hardware-efficient. For the architecture in Fig. 2.5, at least half of nodes in each stage are always idle during decoding procedure. Therefore, in order to increase hardware utilization, two types of architectures, referred as tree-based and line-based architectures [44], are usually used to construct overall SC decoder. In this section we develop our 2b-SC decoder with these two approaches as well.

Fig. 2.8 shows the architecture of a tree-based 2b-SC decoder with n=8. In this design, when a particular stage is activated, all the nodes in that stage are activated. Therefore, a total of (n-2) PEs and a single **p** node are needed.

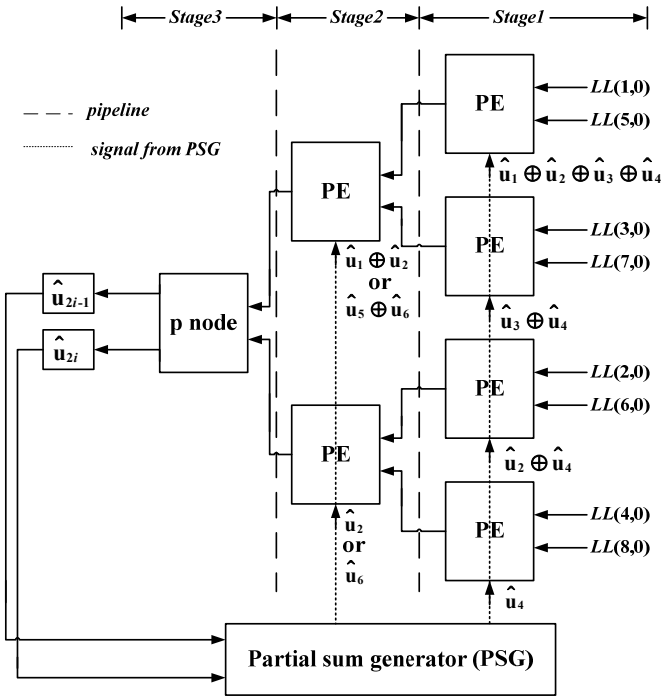


Fig. 2.8. The tree-based 2b-SC architecture with n=8.

One of the disadvantages of the tree-based architecture is that only the activated stage can achieve 100% hardware utilization in each cycle. Considering the waste of idle resource, line-based 2b-SC architecture, which merges (m-1) stages into a single stage, is illustrated in Fig. 2.9. In this figure, the numbers associated with the switches indicate the time index when the switches will be turned on. Compared with the tree-based architecture, the line-based architecture is attractive for moderate-speed applications due to its low hardware cost and better hardware utilization efficiency.

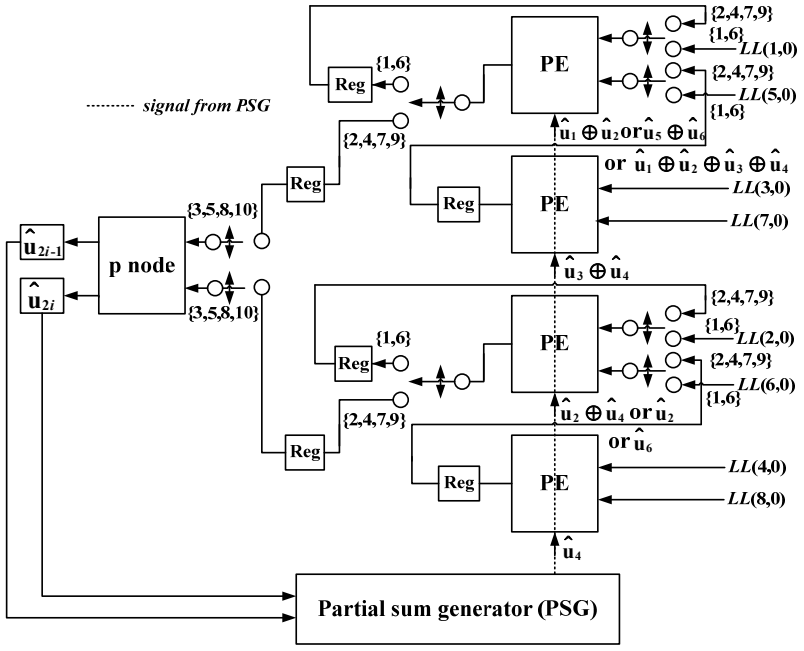


Fig. 2.9. The line-based 2b-SC architecture with n=8.

Besides the aforementioned tree-based and line-based architectures, overlapped architecture [44] and semi-parallel architecture [45] are two other types of architectures. In [44], the overlapped architecture was proposed to process multiple codeword to overcome the hardware underutilization of the tree-based architecture. The disadvantage of the overlapped architecture is the need for extra register/memory resource. In [45], the semi-parallel architecture was proposed to achieve low complexity by using fewer PEs. As a result, the hardware utilization is improved at the expense of increasing decoding latency.

As a general latency-reducing approach, the proposed 2b-SC decoding scheme can also be applied to the overlapped architecture in [44] and semi-parallel architecture in [45]. Similar to tree-based and line-based 2b-SC architectures, the 2b-SC version of overlapped and semi-parallel architectures can be easily developed by replacing the original last stage with our proposed **p** node. Therefore, in this section the 2b-SC designs based on overlapped and semi-parallel architectures are not discussed in detail.

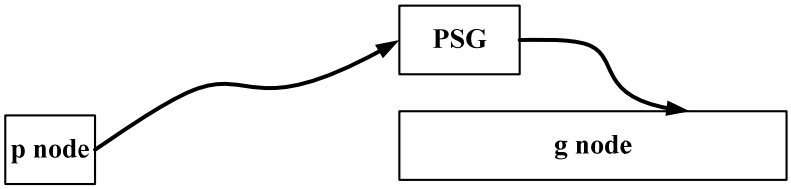
2.4.4 2b-SC-Overlapped Scheduling Architecture

In Section 2.4.2, it is observed that the **p** node has shorter critical path than the PE and this can be exploited to reduce the overall latency to (n-1). This section explains the reason for this reduction and then develops the corresponding architecture, referred as *2b-SC-Overlapped-scheduling architecture*.

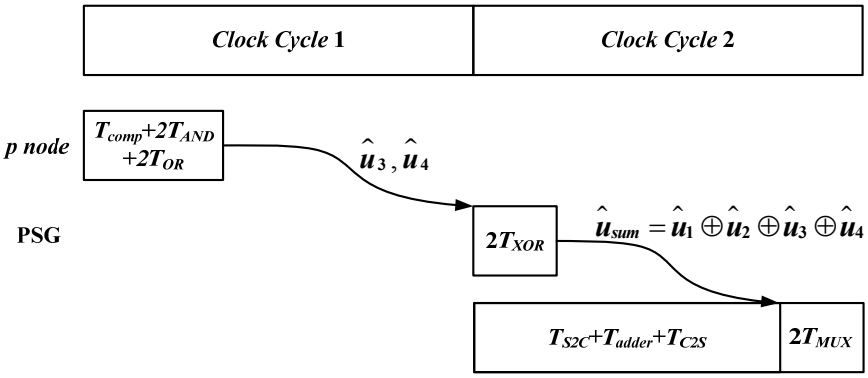
As illustrated in Fig. 2.5, after **p** node computes current $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$, in the next cycle, the **g** node, instead of **f** node, will be activated each time. The decoding sequence between these two nodes is illustrated in Fig. 2.10(a), and its example timing chart for hardware architecture is shown in Fig. 2.10(b). First it takes $T_{comp}+2T_{AND}+2T_{OR}$ for **p** node to compute $\hat{u}_3$ and $\hat{u}_4$ (see Fig. 2.7), and then the PSG block will use these two bits to calculate $\hat{u}_{sum}$. Finally $\hat{u}_{sum}$ is input to the PE for the computation of the **g** node (see Fig. 2.6). Note that here the critical path delay of the PSG block is always $2T_{XOR}$. This is because the computation of $\hat{u}_{sum}$ can be executed in a recursive manner. For example, in order to compute $\hat{u}_{sum}=\hat{u}_1\oplus\hat{u}_2\oplus\hat{u}_3\oplus\hat{u}_4$, because $\hat{u}_1$ and $\hat{u}_2$ have been obtained and $\hat{u}_1\oplus\hat{u}_2$ has been computed and stored in the PSG block in the previous cycle, only two exclusive-or operations are needed to obtain $\hat{u}_{sum}$ from $\hat{u}_3$ and $\hat{u}_4$.

After a careful examination of the decoding sequence in Fig. 2.10(a), it is found that the computations of **p** and **g** nodes can be overlapped. The new decoding sequence with *overlapped scheduling* [43] is shown in Fig. 2.10(c). Here the computations of the **p** and **g** nodes are carried out in the same clock cycle; therefore, one cycle can be saved each time. The validity of the proposed overlapped scheduling is shown in Fig. 2.10(d). The arrival time of $\hat{u}_{sum}$ for PE is $T_{pnode}+2T_{XOR}$, which is much less than its maximum allowable arrival time $T_{S2C}+T_{adder}+T_{C2S}$ (according to Fig. 2.6). For example, with 5-bit quantization and FreePDK 45nm standard CMOS technology, synthesis results show that $T_{S2C}+T_{adder}+T_{C2S}$=0.9539ns while $T_{pnode}+2T_{XOR}$ is only

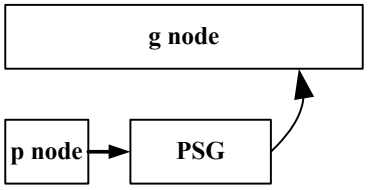
0.5417ns. Therefore, the overlapped computation of **p** node and **g** node in the PE can be accurately carried out without timing conflict. Considering **p** node is activated for 0.5n cycles, this overlapped scheduling approach reduces the overall latency to (1.5n-2)-(0.5n-1)=(n-1). Table 2.4 shows a scheme of the 2b-SC-Overlapped-scheduling decoder for n=8 polar code. Based on this scheme, the corresponding tree-based and line-based architectures can also be easily derived from Fig. 2.8 and Fig. 2.9 by removing the registers between the **p** node and the PSG block.



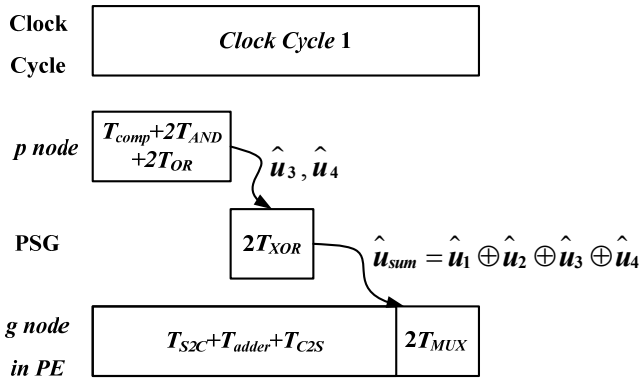
(a)



(b)



(c)



(d)

Fig. 2.10. The effect of overlapped scheduling.
(a) Original 2b-SC decoding sequence between **p** and **g** nodes. (b) example timing chart for original decoding scheme.
(c) decoding sequence between **p** node and **g** node in PE after overlapped scheduling. (d) example timing chart after overlapped scheduling.

Table 2.4. Overlapped scheduling of 2b-SC for n=8 polar code

Overlapped scheduling of 2b-SC decoding scheme							
Clock cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Stage1	**f**			**g**			
Stage2		**f**	**g**		**f**	**g**	
Stage3			**p**	**p**		**p**	**p**
Output			$\hat{u}_1$ & $\hat{u}_2$	$\hat{u}_3$ & $\hat{u}_4$		$\hat{u}_5$ & $\hat{u}_6$	$\hat{u}_7$ & $\hat{u}_8$

2.4.5 2b-SC-Precomputation Architecture

In [42], *precomputation* technique was exploited to reduce the overall latency of the original SC algorithm. The essential idea of this method is to merge the computation of **f** and **g** nodes in the same stage. Table 2.5 shows a schedule of the SC-Precomputation decoding scheme. In each clock cycle, the computations of **f** and **g** nodes are carried out at the same time. As a result, the overall latency is 50% less than that of the conventional scheme in Table 2.1. Moreover, in order to implement the precomputation scheme, [46-47] proposed to employ merged PEs (see Fig. 2.11). Different from conventional 2-input 1-output PE (Fig. 2.6), this modified 2-input 3-output PE can calculate the exact output of **f** node and 2 output candidates of **g** node at the same time.

The valid output of the **g** node will be selected and propagated to the next stage when corresponding $\hat{u}_{sum}$ is available.

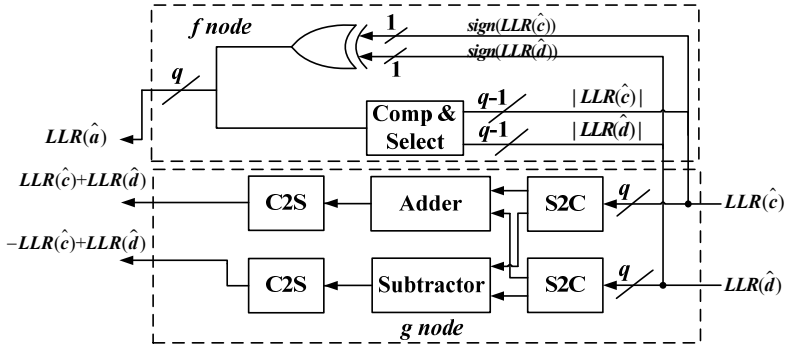


Fig. 2.11. The architecture of merged PE for SC-Precomputation in [46].

Although SC-Precomputation decoder in [46-47] has saved half of the clock cycles, with the help of the reformulation (**p** node) of the last stage, further reduction on latency can be obtained. Recall that the function of the **p** node is to output 2 bits in one cycle; therefore, the merged computations for **f** and **g** nodes in the last stage of SC-Precomputation scheme (Table 2.5) can be completely replaced by the **p** node. In addition, since the critical path of the **p** node is short, the computation of **p** node in adjacent cycles can be merged into one cycle. Table 2.6 shows the example decoding scheme of this 2b-SC-Precomputation decoder. Based on this new scheme, the overall latency is further reduced from (n-1) to (3n/4-1).

When merging two successive computations of **p** nodes into one cycle, a potential problem is the increase of critical path delay. Because the longest data path between two successive computations of **p** nodes is longer than that in the merged PE in Fig. 2.11, a straightforward implementation of the merge operation will increase the critical path delay. To solve this problem, *look-ahead* technique [43] is applied to the last stage. An example of this reformulation is illustrated in Fig. 2.12. By using look-ahead technique, the critical path of the last stage is reduced from $2T_{pnode}+T_{PSG}+T_{MUX}$ in Fig. 2.12(a) to $T_{pnode}+T_{PSG}+T_{4\text{-}1MUX}$ in Fig. 2.12(b), which is

smaller than the longest path delay in the PE. The validity of this assumption has been verified by synthesis results. With 5-bit quantization and 45nm technology, $T_{pnode}+T_{PSG}+T_{4\text{-}1MUX}$=0.6738ns while T_{PE} is about 0.9539ns. Therefore, the critical path delay of overall 2b-SC-Precomputation decoder will be the same as that of the SC-Precomputation decoder. Table 2.7 shows the example decoding scheme of 2b-SC-Precomputation after look-ahead reformulation.

Table 2.5. Decoding schemes of SC-Precomputation [46]

SC-Precomputation decoding scheme[46]							
Clock cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Stage1	**Merged f&g**						
Stage2		**Merged f&g**			**Merged f&g**		
Stage3			**Merged f&g**	**Merged f&g**		**Merged f&g**	**Merged f&g**
Output			$\hat{u}_1$ & $\hat{u}_2$	$\hat{u}_3$ & $\hat{u}_4$		$\hat{u}_5$ & $\hat{u}_6$	$\hat{u}_7$ & $\hat{u}_8$

Table 2.6. Decoding schemes of 2b-SC-Precomputation before look-ahead

2b-SC-Precomputation decoding scheme before look-ahead reformulation						
Clock cycle	**1**	**2**	**3**	**4**	**5**	
Stage1	**Merged f&g**					
Stage2		**Merged f&g**		**Merged f&g**		
Stage3			**p** **p**		**p**	**p**
Output			$\hat{u}_1$ & $\hat{u}_2$	$\hat{u}_3$ & $\hat{u}_4$	$\hat{u}_5$ & $\hat{u}_6$	$\hat{u}_7$ & $\hat{u}_8$

Table 2.7. Decoding schemes of 2b-SC-Precomputation after look-ahead

2b-SC-Precomputation decoding scheme after look-ahead reformulation					
Clock cycle	**1**	**2**	**3**	**4**	**5**
Stage1	**Merged f&g**				
Stage2		**Merged f&g**		**Merged f&g**	
Stage3			**p** **p**		**p** **p**
Output			$\hat{u}_1$ & $\hat{u}_2$, $\hat{u}_3$ & $\hat{u}_4$		$\hat{u}_5$ & $\hat{u}_6$, $\hat{u}_7$ & $\hat{u}_8$

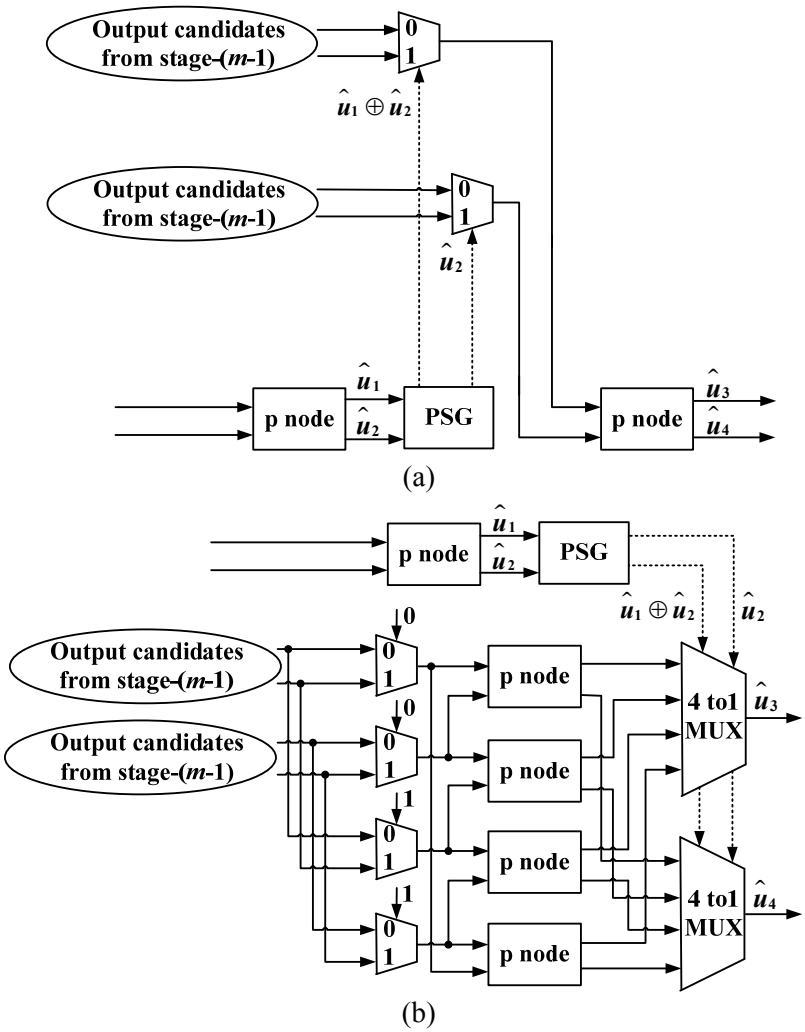


Fig. 2.12. Reformulation of **p** node
(a) Original design for two successive computations of **p** nodes in the last stage (stage-*m*). (b) Look-ahead reformulation.

2.5 Hardware Performance

In this section, we analyze the hardware performance characteristic of the proposed 2b-SC architectures and compare them with the state-of-the–art designs. Tables 2.8 shows the required hardware resource, latency and throughput of different (n, k) polar tree-based and line-based SC architectures, respectively. In this table all the listed designs are assumed to be constructed based on the same PE with q-bit quantization. Notice that in order to achieve good decoding performance with small word length, non-uniform quantization scheme similar to those in [48-49]

can be used.

From Table 2.8 it can be seen that that the normalized throughput of the 2b-SC, 2b-SC-Overlapped-scheduling, 2b-SC-Precomputation decoders are 1.33, 2, and 2.67, respectively, where these are normalized to the SC decoder in [44]. Compared with SC design in [44], the 2b-SC and 2b-SC-Overlapped-scheduling decoders have much shorter decoding latency. Since the critical path remains the same, this reduction in latency can lead to increased throughput. Meanwhile, unlike SC-Precomputation decoders [46], the 2b-SC and 2b-SC-Overlapped-scheduling decoders succeed in reducing latency without requiring any extra registers. Therefore, these two decoders maintain low complexity. Besides, by applying precomputation technique to the 2b-SC design, the latency of the 2b-SC-Precomputation architecture is reduced to ($3n/4$-1). To the best of our knowledge, this is the shortest decoding latency among all known SC decoders. Since **p** node occupies very small area of the whole decoder (<0.01%), the proposed 2b-SC-Precomputation decoder has about 30% higher normalized throughput than the SC-Precomputation decoder in [46] with the same complexity.

Additionally, in order to demonstrate the advantage of the proposed architectures, we have implemented our designs for polar (1024, 512) code with Verilog HDL. Here tree-based 2b-SC-Precomputation architecture is selected for implementation. After developing the RTL models, we synthesize our decoders with FreePDK 45nm standard CMOS library by using Synopsys Design Complier.

Table 2.9 lists the implementation results of reported polar (1024, 512) SC decoders. Notice that [50] used a speculative method to achieve 2 bits output in one cycle. Compared with the hardware-based method in [50], our proposed 2b-SC approach is more general since it reformulates the algorithm. As a result, this reformulation reduces the critical path of the last stage, and then enables the reduced-latency 2b-SC-Overlapped-scheduling and 2b-SC-Precomputation architectures.

Table 2.8. Hardware comparison of (n, k) SC decoders

Tree-based and Line-based Architecture						
Hardware		**SC-Precomputation [46]**	**SC [44]**	**2b-SC**	**2b-SC with Overlapped-scheduling**	**2b-SC with Precomputation**
# of PE	**Tree-based**	n-1	n-1	n-2	n-2	n-2
	Line-based	$n/2$	$n/2$	$n/2$	$n/2$	$n/2$
# of p node		0	0	1	1	5
# of 1-bit REG		$\sim 3qn$	$\sim qn$	$\sim qn$	$\sim qn$	$\sim 3qn$
Latency (cycle)		n-1	$2n$-2	$1.5n$ -2	n-1	$0.75n$-1
Throughput (Normalized)		2	1	1.33	2	2.67

Table 2.9. Comparison of (1024, 512) SC decoders with 5-bit quantization

Design	**[50]***	**[45]**	**Tree-based 2b-SC-Precomputation**
CMOS Technology	180nm	65nm	45nm
Total gate counts	183637	214370**	338499
Frequency (MHz)	150	500	750
Decoding latency (cycle)	1560†	2080†	767
Throughput (Mbps)	49	123	500
TSNT (Mbps/Kgate) (scaled to 45nm)‡	1.07	0.83	1.48

* Results in [50] are measurement results.

** Gate count is calculated based on the area information in [45] and unit gate area in TSMC 65nm CMOS library.

† Decoding latency is calculated based on the equation (12) in [45].

‡ Technology scaled normalized throughput, referred as TSNT, is defined by $\frac{Throughput * (technology / 45nm)}{Total\ gate\ count}$.

From Table 2.9 it can be seen that our design can achieve at least twice reduction in latency as well as 4 times increase in throughput. When scaling to the same technology (45nm), the technology scaled normalized throughput (TSNT) metric, defined as throughput per Kgate, increases by at least 40% for our design. Notice that the designs in [45] [50] are based on semi-parallel architecture while our design is based on high-complexity tree architecture. If the proposed 2b-SC-Precomputation design is also implemented on the same low-complexity semi-parallel architecture, the advantage of our design on hardware performance will be further

improved. We estimate that the semi-parallel-based 2b-SC decoding, 2b-SC with overlapped scheduling and 2b-SC-Precomputation decoders require latencies of around 1.5n, n and 0.75n with area overhead of 0, 0, and 40%, respectively. Therefore, these architectures offer the throughput/area advantages by factors 1.33, 2 and 1.92, respectively, as compared to the semi-parallel architecture in [45].

Due to the generality of 2b-SC decoding scheme, it can be widely applied to current and future SC decoders, independent of the design of the **f** and **g** nodes. In summary, the proposed 2b-SC decoding algorithm and architectures are very attractive for hardware implementations of low-latency polar SC decoders.

2.6 Conclusion

In this chapter, a novel reformulation for the last stage of theSC decoding is proposed. Based on this reformulation, a reduced-latency 2b-SC decoding algorithm is presented. In addition, with the use of overlapped scheduling and precomputation approaches, the decoding latency of 2b-SC design is further reduced. Analysis shows that the proposed 2b-SC architectures have significant advantages with respect to both throughput and hardware efficiency.

Chapter 3

3 SC LIST DECODER

In this chapter, we present the reduced-latency SC list decoder. Section 3.2 reviews the original SC list decoding algorithm from the aspect of coding tree. In Section 3.3, reduced-latency SC list decoding algorithms are proposed. Section 3.4 presents the hardware architecture of the decoders. Hardware performance is analyzed and compared with prior works in Section 3.5.

3.1 Introduction

Although polar codes have inherent capacity-achieving property, in the regime of small and medium code-length, their error-correcting performance is inferior to LDPC and Turbo codes with SC decoding algorithm. To address this problem, in [6] SC list (SCL) decoding algorithm was proposed to improve the performance of polar codes. The main idea of SCL decoding is to utilize multiple searching paths instead of one searching path in SC algorithm. As a result, the probability of finding the valid codeword increases significantly. In [9-11][26-27], several variants of SCL algorithms and the hardware architectures were presented.

Because SCL algorithm consists of multiple SC decoding procedure, an SCL decoder also suffers from long latency problem. Even worse, since sorting operation is needed for each bit decision, the latency of SCL algorithm is much longer than SC algorithm. As a result, how to reduce latency of SCL decoder is an important research topic for the practical use of polar codes.

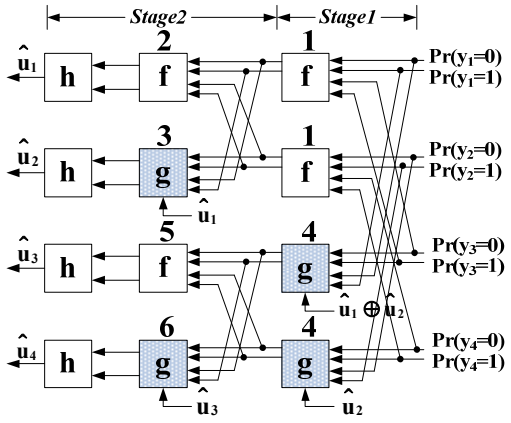
This chapter presents multi-bit-decision approaches that can reduce the latency of SCL decoders. First, 2-bit reformulated SCL (2b-rSCL) algorithm, which can perform intermediate decoding of 2 bits simultaneously, is presented to reduce the overall latency from ($3n$-2) cycles to ($2n$-2) cycles. Then, by generalizing the 2-bit-decision idea, we propose a general 2^K-bit reformulated

SCL (2Kb-rSCL) algorithm. By performing intermediate decoding of 2^K bits together, the proposed 2^Kb-rSCL decoder has latency as short as n/2^K-2-2 cycles. Synthesis results show that compared with the prior SCL decoder, the proposed (1024, 512) 2b-rSCL and 4b-rSCL decoders have significant reduction in latency and throughput.

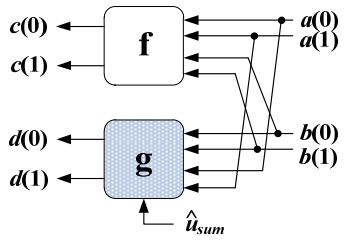
3.2 SC List Decoding Algorithm

3.2.1 Revisit SC Algorithm in Coding Tree

Because the original SC list decoding algorithm was described in likelihood form and it consists of multiple SC decoders, the likelihood-based SC algorithm needs to be reviewed first. As discussed in Chapter 2, the function of SC decoder is to recover the $\boldsymbol{u}$ from the $\boldsymbol{y}$. Fig. 3.1 shows the example decoding procedure of likelihood-based SC decoder for n=4 polar code. As seen in this figure, the SC decoder consists of m=$\log_2 n$=2 stages, where each stage consists of two types of 4-input-2-output units, referred as **f** unit and **g** unit, respectively. In addition, a 2-input-1-output hard-decision unit denoted as **h** is used at the last stage of SC decoder (stage-2) to determine the estimate of u_i, referred as $\hat{u}_i$. Similar to the case in Chapter 2, each **f** or **g** unit is labeled a number to indicate the clock cycle index when it is activated.



(a)



(b)

Fig. 3.1. Likelihood-based SC decoder
(a) SC decoding procedure with n=4. (b) Basic unit of SC decoder.

As discussed in Chapter 2, the functions of **f** and **g** units can be described as follows.

$$c(0) = a(0)b(0) + a(1)b(1) \tag{3.1}$$

$$c(1) = a(0)b(1) + a(1)b(0) \tag{3.2}$$

$$d(0) = \Pr(\widehat{in}_2 = 0, \widehat{in}_1 = \hat{u}_{sum}, \hat{u}_1^{i-1} = z_1^{i-1}) = a(\hat{u}_{sum})b(0) \tag{3.3}$$

$$d(1) = \Pr(\widehat{in}_2 = 1, \widehat{in}_1 = \hat{u}_{sum}, \hat{u}_1^{i-1} = z_1^{i-1}) = a(1-\hat{u}_{sum})b(1)\,, \tag{3.4}$$

where $a(0)$, $a(1)$, $b(0)$, $b(1)$ are the inputs of **f** or **g** unit.

On the other hand, from the view of code tree, the SC algorithm can be described as a path searching process. Fig. 3.2 shows an example for n=4 and k=4 SC decoding procedure over the code tree. This n=4 code tree consists of 4 levels, where each level represents a decoded bit. The value associated with each node is the likelihood-based metric for the decoding path from root node to the current node. For example, 0.33 on the leftmost side indicates that for the path $\hat{u}_1$=0 and $\hat{u}_2$ =0, denoted as the length-2 path (00), its metric is given by Pr($\hat{u}_1$=0, $\hat{u}_2$=0)=0.33. For the 0.12 on the rightmost side, it indicates the metric for path $\hat{u}_1$ =1, $\hat{u}_2$=1 and $\hat{u}_3$=1, denoted as the length-3 path (111), is given by Pr($\hat{u}_1$ =1, $\hat{u}_2$=1 and $\hat{u}_3$ =1)=0.12. In particular, the path metrics associated with the nodes at the lowest level (level 4) represent the different likelihoods for the different combinations of ($\hat{u}_1\hat{u}_2\hat{u}_3\hat{u}_4$). The valid output of this n=4 SC polar decoder should be the length-4 path which has the largest metric at the lowest level. In this example it is (0010) with path metric Pr($\hat{u}_1 = 0, \hat{u}_2 = 0, \hat{u}_3 = 1, \hat{u}_4 = 0$)=0.19.

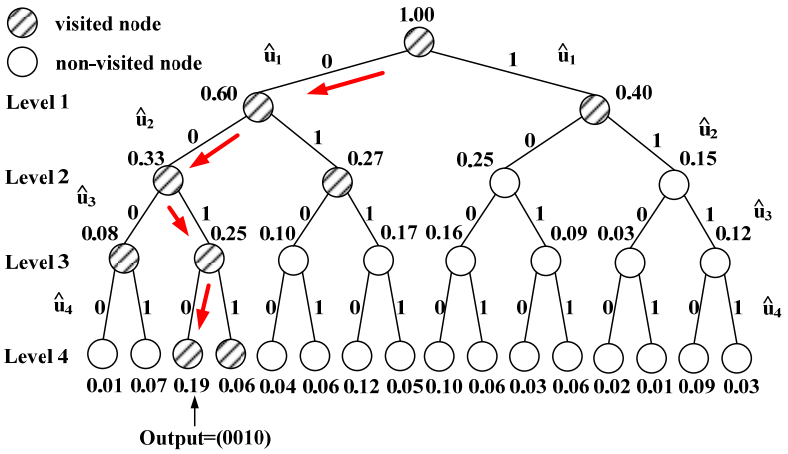


Fig. 3.2. Searching process of SC decoder (n=4 and k=4).

Notice the aforementioned path metrics are calculated by the **f** or **g** units in the last stage of the SC decoder (for example stage-2 in Fig. 3.1(a)). For the length-i path, its path metric is computed by the **f** or **g** unit associated with $\hat{u}_1$. For example, for n=4 polar code, the path metric for path ($\hat{u}_1$ $\hat{u}_2$) is computed by index-3 **g** unit in Fig. 3.1(a). Similarly, the path metric for path ($\hat{u}_1$ $\hat{u}_2$ $\hat{u}_3$) is computed by index-5 **f** unit in Fig. 3.1(a).

In order to find the decoding path with the largest metric, SC algorithm adopts a locally optimal searching strategy. As shown in Fig. 3.2, the arrows represent the survival decoding path of the SC decoder. In the i-th level, the SC decoder first visits the two children nodes (striped nodes in Fig. 3.2) that are connected to the current survival length-(i-1) path. Since the metrics of length-i paths are associated with these children nodes, the SC decoder then can obtain the metrics of length-i paths. After comparing the metrics, the SC decoder only selects the length-i path which has the larger metric as the updated survival path, while the path which has the smaller metric will never be explored in the future. Based on this searching strategy, in Fig. 3.2 the length-4 path (0010) with metric 0.19 is selected as the output of SC decoder. In this example, the SC decoder works well since it finds the valid length-4 path with the largest metric.

3.2.2 SC List Decoding Algorithm

An essential drawback of the SC algorithm is that its searching strategy over the code tree is only locally optimal, but not globally optimal. As a result, in many cases the (n, k) SC decoder cannot find the length-n path with the largest metric. For example, if we apply SC decoding approach in Fig. 3.3, its output is (0010) with metric 0.19; however, the valid length-4 path with largest metric should be (1000) with metric 0.23.

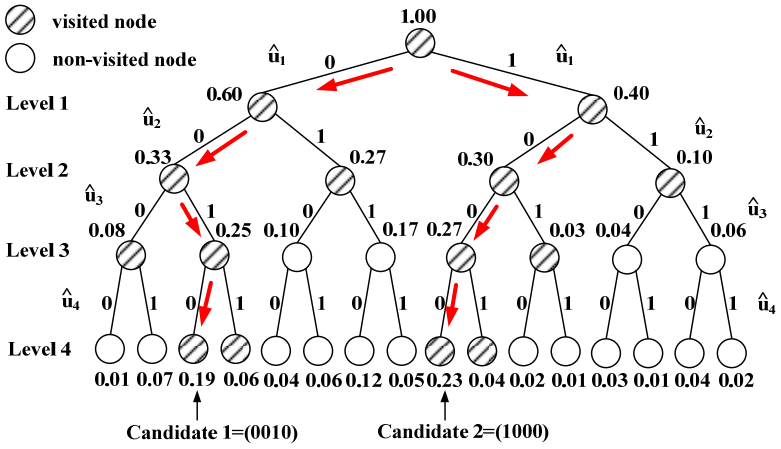


Fig. 3.3. Searching process of SCL decoder (n=4, k=4, L=2).

The reason for the inefficiency of SC algorithm in this example is that sometimes the unexplored path, instead of the chosen survival path, has the larger path metric. Based on this observation, successive cancellation list (SCL) algorithm [6] was proposed to perform searching process along multiple survival paths at the same time. Here the maximum number of the survival paths is referred as the list size (L). Fig. 3.3 shows an example for the n=4 and k=4 SCL decoder with L=2. As shown in Fig. 3.3, at the i-th level, the SCL decoder visits all the 2L children nodes (striped nodes in Fig. 4) that are connected to the length-(i-1) survival paths. After calculating all the 2L new path metrics associated with these children nodes, the SCL decoder selects the L length-i paths which have the larger metrics as the updated survival paths. From Fig. 3.3 it can be seen that the valid decoding path (1000), which could not be traced by SC decoder before, now can be found by the SCL decoder.

3.3 Reduced-latency SC List Decoding Algorithm

In general, the SCL algorithm can improve decoding performance significantly over the SC algorithm [6]. However, one of the major challenges for the practical use of SCL decoder is the long latency problem. Because an L-size (n, k) SCL decoder can be viewed as the combination of L copies of (n, k) SC component decoders (see Fig. 3.4), an (n, k) SCL decoder needs the same ($2n$-2) cycles to process its **f** and **g** units as its SC component decoders do. In addition, since SCL decoders need to sort $2L$ path metrics and select L largest metrics for each decoded bit (see Fig. 3.4), extra n cycles are needed to carry out the sorting and selecting function to avoid long critical path [51]. Therefore, the latency of an (n, k) SCL decoder is $3n$-2 cycles. As discussed in Section 3.2, although some methods have been proposed to reduce the latency of SC decoders, these approaches cannot be directly applied to the SCL decoder. As a result, the latency of current known SCL decoder is still very long. Table 3.1 shows an example decoding scheme of conventional SCL decoder for n=4 polar code. Here in this table the symbols **f** and **g** represent the **f** and **g** units in each SC component decoder of Fig. 3.1 respectively. Besides, the symbol **s** represents the path metrics sorting and selecting operation for each intermediately decoded bit.

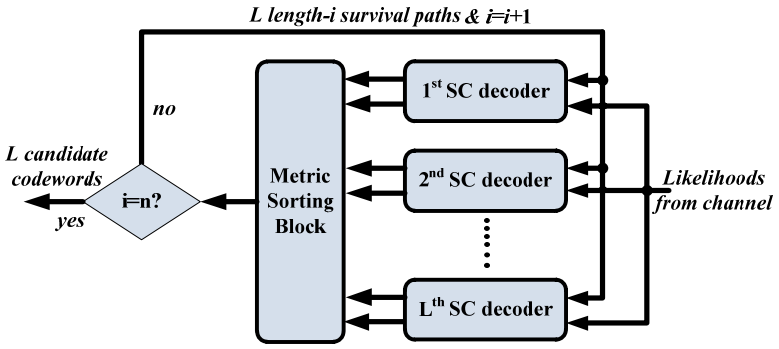


Fig. 3.4. Block diagram of L-size SCL decoder.

Table 3.1. Decoding scheme of SCL decoder with n=4

Clock cycle	1	2	3	4	5	6	7	8	9	10
Stage-1	**f**					**g**				
Stage-2		**f**	**s**	**g**	**s**		**f**	**s**	**g**	**s**
Bit decision			$\hat{u}_1$		$\hat{u}_2$			$\hat{u}_3$		$\hat{u}_4$

3.3.1 2b-SC List Decoding Algorithm

As seen in Table 3.1, more than 60% latency of SCL decoder is due to the computation of **f**, **g** and **s** in the last stage (stage-2, in Table 3.1). This phenomenon implies that the reduction of latency in the last stage can lead to significant reduction of the overall latency of SCL decoder. Therefore, in this section we propose to reformulate the original computation of the last stage. This reformulated computation in the last stage can save many clock cycles without any performance loss.

Table 3.1 shows that the computation of the last stage can be viewed as multiple "**f s g s**" functions to perform intermediate decoding of two consecutive bits $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$. Since the **f**/**g** units and **s** in the last stage contribute to path metrics calculation and selection, respectively, hence the goal of our reformulation on the last stage is to find a simplified method that can compute path metrics and sort/select them to perform intermediate decoding of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ more quickly.

Fig. 3.5 (a) and (b) show the block diagram of the original and reformulated SC component decoder for SCL decoding, respectively. From these two figures it can be seen that the reformulated SC decoder replaces the original stage-m with two new units, referred as metric computation unit (MCU) and zero-forcing unit (ZFU), respectively. Besides that, as shown in Fig. 3.4, a sorting block (**s** symbol in Table 3.1) is also needed to sort the path metrics output from all the L SC component decoders. Because the sorting block is an individual block that does not belong to any SC component decoder, in this subsection we do not discuss sorting block but focus on the functions of MCU and ZFU. The architecture of sorting block will be presented in Section

3.4.3.

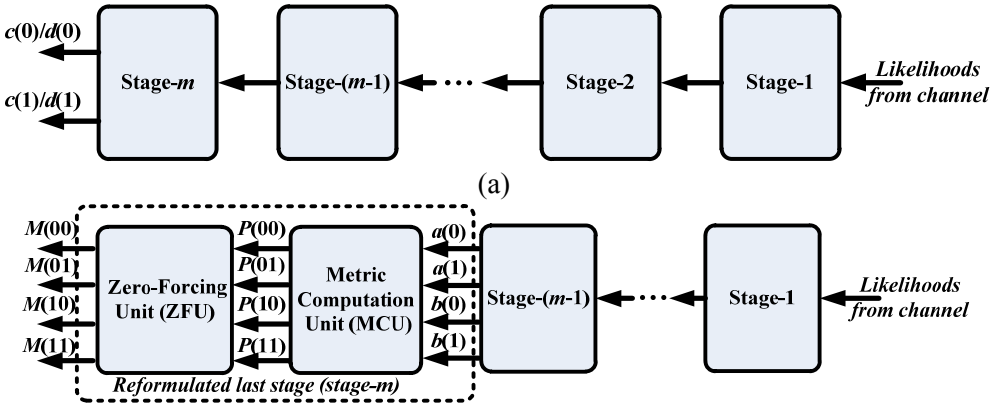


Fig. 3.5. Block diagram of SC component decoder
(a) original SC component decoder of SCL decoder. (b) reformulated SC component decoder of 2b-SCL decoder.

Metric Computation Unit (MCU)

As shown in Fig. 3.6, metric computation unit (MCU) calculates the likelihoods for different combinations of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ with the use of the messages $a(0)$, $a(1)$, $b(0)$ and $b(1)$ output from stage-(m-1). The principle of this calculation can be derived from (1)-(4). Since for the last stage of each SC component decoder, $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ are the estimates, therefore with (3.1)-(3.4) we can have:

$$\begin{aligned}
P(00) &\triangleq \Pr(\hat{u}_{2i-1}=0,\ \hat{u}_{2i}=0, \hat{u}_1^{2i-2}=z_1^{2i-2}) = a(0)b(0)\\
P(01) &\triangleq \Pr(\hat{u}_{2i-1}=0,\ \hat{u}_{2i}=1, \hat{u}_1^{2i-2}=z_1^{2i-2}) = a(1)b(1)\\
P(10) &\triangleq \Pr(\hat{u}_{2i-1}=1,\ \hat{u}_{2i}=0, \hat{u}_1^{2i-2}=z_1^{2i-2}) = a(1)b(0)\\
P(11) &\triangleq \Pr(\hat{u}_{2i-1}=1,\ \hat{u}_{2i}=1, \hat{u}_1^{2i-2}=z_1^{2i-2}) = a(0)b(1)\ .
\end{aligned} \tag{3.5}$$

where $\hat{u}_1^{2i-2}=z_1^{2i-2}$ denotes that the previously decoded bits $\hat{u}_1$, $\hat{u}_2 \ldots \hat{u}_{2i-2}$ are assumed to have been determined as z_1, z_2,… z_{2i-2}, respectively.

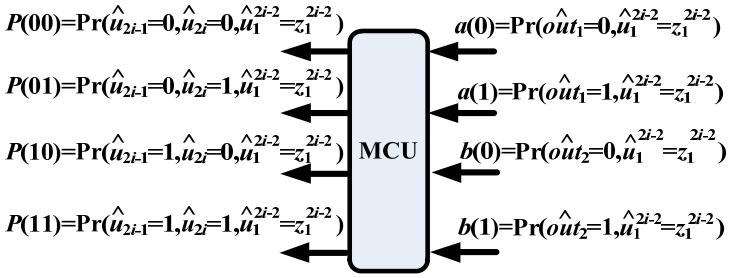


Fig. 3.6. Block diagram of MCU for 2b-rSCL decoder.

(3.5) describes the calculation of the joint likelihoods of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ and $\hat{u}_1^{2i-2} = z_1^{2i-2}$. Now we show that these joint likelihoods are just the actual metrics of length-2i paths. Consider one of the current length-(2i-2) survival path in the code tree as $(\hat{u}_1,\ldots \hat{u}_{2i-2})=(z_1 \ldots z_{2i-2})$. As shown in Fig. 3.7, with the different combination of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$, this length-(2i-2) path can be extended to four length-(2i) paths as $(\hat{u}_1,\ldots \hat{u}_{2i})=(z_1 \ldots z_{2i-2}pq)$, where p and q are binary 0 or 1. According to the definition of path metric, with the four combinations of p and q, $\Pr(\hat{u}_{2i-1}=p,\ \hat{u}_{2i}=q,\ \hat{u}_1^{2i-2} = z_1^{2i-2})$ in (3.5) are just the actual metrics of the above four extended length-(2i) paths. As a result, according to (3.5), with the knowledge of $a(0)$, $a(1)$, $b(0)$ and $b(1)$ output from the stage-(m-1), we can obtain the actual path metrics of four length-(2i) paths $(\hat{u}_1,\ldots \hat{u}_{2i})=(z_1 \ldots z_{2i-2}pq)$.

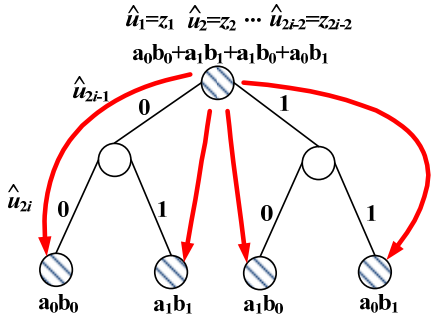


Fig. 3.7. Extension from one path to four paths.

Zero-Forcing Unit (ZFU)

Although (3.5) provide a fast approach to compute the actual metrics of length-2i paths, a post-processing operation is still needed before inputting those calculated metrics into the sorting

block. This is because the values of $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ do not only depend on the corresponding path metrics, but also on whether they are frozen bits or not. Notice that when the current decoded bit $\hat{u}_i$ is a frozen bit, the paths with $\hat{u}_i$=1 are not qualified and should never be selected even if they have larger metrics than their counterparts. As a result, in order to avoid selecting those unqualified paths, we need a zero-forcing unit (ZFU) to force the metrics of those unqualified paths to 0. The reason of this zero-forcing operation is that since the SCL decoder only selects the L survival paths with larger metrics for each $\hat{u}_i$, therefore the unqualified paths with metric values 0 will never be classified into the group of L paths with larger metrics. As a result, the validity of the function is guaranteed.

Since the proposed reformulated last stage involves both $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$, the function of ZFU is derived as follows:

Assign $M(pq)=P(pq)$ for path $(z_1^{2i-2}pq)$ with $p, q \in \{0,1\}$;

If $\hat{u}_{2i-1}$ is frozen, then reassign $M(10)=0$ and $M(11)=0$;

If $\hat{u}_{2i}$ is frozen, then reassign $M(01)=0$ and $M(11)=0$. (3.6)

(3.5) and (3.6) describe the reformulated function of the last stage of SC component decoder. With the help of this reformulation, $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$, as the two successive decoded bits, can now be intermediately decoded at the same time. Fig. 3.8 (a) and (b) show the decoding procedure of original SCL decoding and the proposed reformulated approach with list size L, respectively. In the conventional SCL algorithm, with the comparison of their metrics, the L length-(2i-1) survival paths are selected from 2L candidates for each time. And each selection can only perform intermediate decoding of one decoded bit (see Fig. 3.8(a)). Instead, in the reformulated approach, the L length-(2i) survival paths are selected from 4L candidates for each time. As a result, the two successive bits can now be intermediately decoded simultaneously in each selection (see Fig.

3.8(b)).

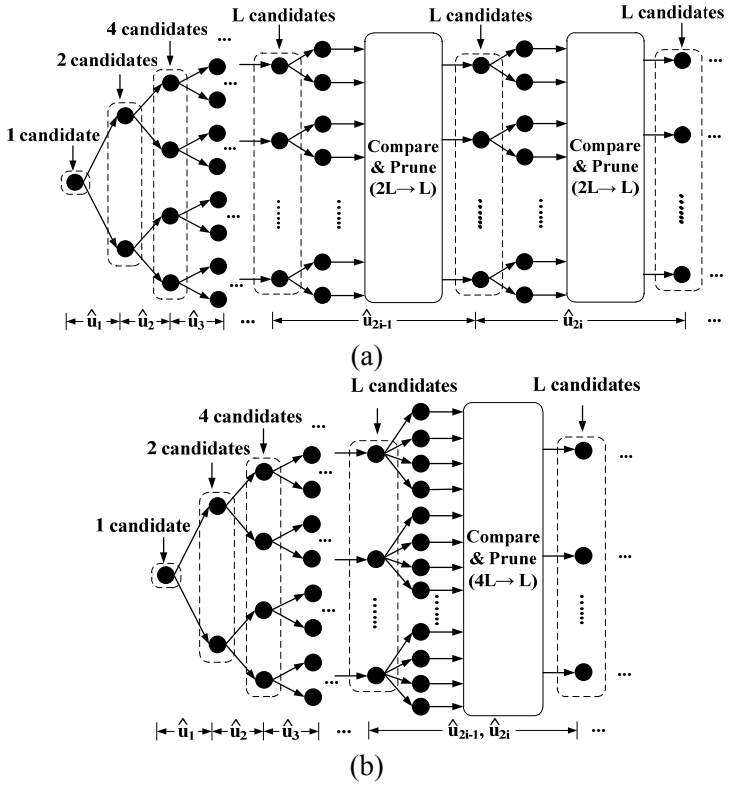


Fig. 3.8. *L*-size decoding scheme
(a) original SCL decoder (b)2b-SCL decoder

Considering the proposed reformulation can allow two bits to be intermediately decoded at the same time, this new SCL algorithm is referred as *2-bit SCL* (*2b-SCL*) algorithm and described in Scheme 3.1.

The proposed 2b-SCL algorithm can greatly reduce the latency of the original SCL decoder. Recall that in the original searching procedure (see Fig. 3.3), the SCL decoder needs to compute the path metrics associated with the striped nodes in each level of the code tree. On the other hand, since the 2b-SCL only needs to compute the metrics for length-(2i) paths, the metrics computation for length-(2i-1) paths are totally avoided (see Fig. 3.7). As a result, for the same code tree the 2b-SCL decoder only needs to visit the striped nodes at even levels instead of at all the levels. For example, by comparing Fig. 3.3 and Fig. 3.9, it can be found that the reformulated

SCL decoder does not need to visit the nodes at level 1 and level 3 anymore. As a result, this new decoding scheme leads to immediate saving in clock cycles.

Scheme 3.1: 2-bit SCL decoding (2b-SCL) with list size L for (n, k) polar codes

1: Input: *Likelihoods of each bit in the received codeword*
2: For $i = 1$ **to** $n/2$
3: For each length-$(2i-2)$ **survival path** $(\hat{u}_1 \ldots \hat{u}_{2i-2}) \triangleq z_1^{2i-2}$
4: SC decoding:
5: *Activate stage* $-1 \sim$ *stage* $-(m-1)$ *of SC component decoder*
6: *stage* $-(m-1)$ *outputs* $a(0), a(1), b(0), b(1)$
7: Path Expansion:
8: *Expand survival path* z_1^{2i-2} *to 4 candidate paths* $(z_1^{2i-2}\hat{u}_{2i-1}\hat{u}_{2i})$:
9: *1 length* $-(2i-2)$ *path* $\Rightarrow$ *4 length* $-(2i)$ *paths*
10: Metric Computation:
11: *Calculate actual path metrics* $P(pq)$ *for 4 length* $-(2i)$ *paths*:
12: $P(00)=a(0)b(0)$ *for path* $(z_1^{2i-2}00)$, $P(01)=a(1)b(1)$ *for path* $(z_1^{2i-2}01)$,
13: $P(10)=a(1)b(0)$ *for path* $(z_1^{2i-2}10)$, $P(11)=a(0)b(1)$ *for path* $(z_1^{2i-2}11)$
14: Forcing Zero:
15: *Calculate the new path metrics* $M(pq)$ *with forcing - zero operation*:
16: $M(pq) = P(pq)$ *for path* $(z_1^{2i-2}pq)$ *with* $p, q \in \{0,1\}$
17: *if* $\hat{u}_{2i-1}$ *is frozen, then* $M(10) = 0$ *and* $M(11) = 0$
18: *if* $\hat{u}_{2i}$ *is frozen, then* $M(01) = 0$ *and* $M(11) = 0$
19: End for
20: Compare and Prune:
21: *Compare the metrics* $M(pq)$ *of all the 4L length* $-(2i)$ *candidate paths*
22: *Select L paths with the L largest metrics as the new survival paths*
23: End for
24: Output: *Choose the length - n survival path with the largest metric*

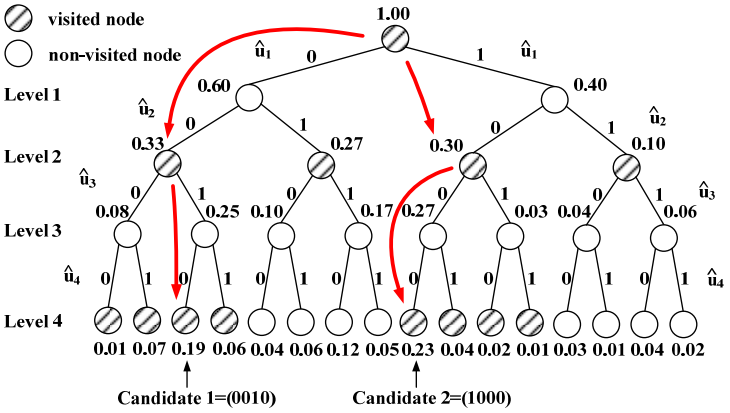


Fig. 3.9. Searching process of 2b-SCL decoder.

Table 3.2 shows the example decoding scheme of the proposed 2b-SCL decoder with n=4. Here **mc&zf** in Table 3.2 denotes the *metric computation* and *zero-forcing* operations, which are described from line 10 to line 18 of Scheme 3.1 in detail. Compared with the scheme of conventional SCL decoder (see Table 3.1), it can be seen that the reformulation at the last stage (stage-2 in this example) leads to significant reduction in clock cycles. For the intermediate decoding of each two successive bits $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$, the original SCL decoder (see Table 3.1) needs 4 cycles (**f**, **s**, **g**, **s**), while the 2b-SCL decoder in Table 3.2 only needs 2 cycles (**mc&zf**, **s**). In general, for an (n, k) polar code, the overall latency of 2b-SCL decoder can reduce from 3n-2 to 2n-2 clock cycles.

Table 3.2. Decoding scheme of 2b-SCL decoder with n=4

Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**
Stage-1	**f**			**g**		
Stage-2		**mc&zf**	**s**		**mc&zf**	**s**
Bit decision			$\hat{u}_1$ $\hat{u}_2$			$\hat{u}_3$ $\hat{u}_4$

3.3.2 2^Kb-SC List Decoding Algorithm

In subsections 3.3.1 we presented 2b-SCL algorithm that can perform intermediate decoding of 2 bits at the same time. In this subsection, we extend the prior approach to a more general case, and propose a new algorithm, referred as *2^K-bit SC List (2^Kb-SCL)*, which can perform intermediate decoding of 2^K bits simultaneously.

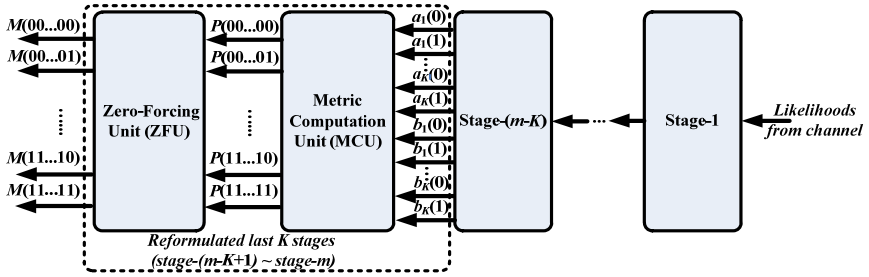


Fig. 3.10. Block diagram of reformulated SC decoder of 2^Kb-SCL decoder.

As shown in Fig. 3.10, the 2^Kb-SCL decoder reformulates the last K stages of original SCL decoder. Similar to the case in 2b-rSCL decoder, the reformulated part of 2^Kb-SCL decoder consists of MCU and ZFU as well.

Metric Computation Unit (MCU)

The function of MCU in 2^Kb-SCL decoder is to compute the joint probabilities of 2^K successive bits as $\hat{u}_{2^K(i-1)+1}$, $\hat{u}_{2^K(i-1)+2}$, … and $\hat{u}_{2^K i}$. Similar to the discussion in 2-bit-decision case, we first investigate the transformation of $u_{2^K(i-1)+1} \ldots u_{2^K i}$.

As shown in Fig. 3.11, the transformation of 2^K successive bits can be viewed as the multiplication with matrix $\boldsymbol{U}$, where $\boldsymbol{U}$ is $2^K \times 2^K$ generator matrix $\boldsymbol{G}$.

Denote $\vec{\boldsymbol{u}}_{2^K,i} \triangleq (u_{2^K(i-1)+1}, u_{2^K(i-1)+2}, \ldots, u_{2^K i})$ and $\overrightarrow{\boldsymbol{out}}_{2^K} \triangleq (out_1, out_2, \ldots out_{2^K})$, then we have:

$$\overrightarrow{\boldsymbol{out}}_{2^K} = \vec{\boldsymbol{u}}_{2^K,i}\boldsymbol{U} . \tag{3.7}$$

In particular, if we denote the j-th column vector of $\boldsymbol{U}$ as $\boldsymbol{U(j)}$, then we have:

$$out_j = \vec{\boldsymbol{u}}_{2^K,i}\,\boldsymbol{U(j)}. \tag{3.8}$$

(3.8) describes the left-to-right transformation of the $u_{2^K(i-1)+1}, u_{2^K(i-1)+2}$, … and $u_{2^K i}$ in encoding phase.

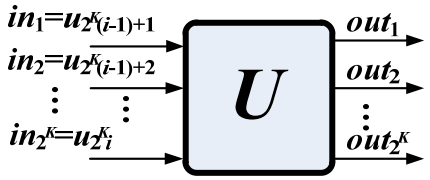


Fig. 3.11. Encoding procedure for $u_{2^K(i-1)+1}, u_{2^K(i-1)+2}$, … and $u_{2^K i}$.

Then, based on (3.8), the right-to-left "guideline" in decoding procedure should be:

$$\hat{\vec{\boldsymbol{u}}}_{2^K,i} = \widehat{\overrightarrow{\boldsymbol{out}}}_{2^K}\boldsymbol{U}^{-1} = \widehat{\overrightarrow{\boldsymbol{out}}}_{2^K}\boldsymbol{U}, \tag{3.9}$$

where $\hat{\vec{\boldsymbol{u}}}_{2^K,i} \triangleq (\hat{u}_{2^K(i-1)+1}, \hat{u}_{2^K(i-1)+2}, \ldots, \hat{u}_{2^K i})$ and $\widehat{\overrightarrow{\boldsymbol{out}}}_{2^K} \triangleq (\widehat{out}_1, \widehat{out}_2, \ldots \widehat{out}_{2^K})$, respectively.

According to (3.8) and (3.9), we have:

$$\widehat{out}_j = \hat{\vec{u}}_{2^K,i}\boldsymbol{U(j)} \text{ and } \hat{u}_{2^K(i-1)+1} = \widehat{\overrightarrow{out}}_{2^K}\boldsymbol{U(j)}. \tag{3.10}$$

Note that in (3.10) we use the special property that $\boldsymbol{U^{-1}=U}$.

As shown in Fig. 3.12, the inputs of MCU are $a_1(0)$, $a_1(1)$,..., $a_{2^{K-1}}(0)$, $a_{2^{K-1}}(1)$, $b_1(0)$, $b_1(1)$,..., $b_{2^{K-1}}(0)$ and $b_{2^{K-1}}(1)$, respectively. With the help of (3.10), we can obtain the joint probabilities of $\hat{u}_{2^K(i-1)+1}$, $\hat{u}_{2^K(i-1)+2}$, … and $\hat{u}_{2^K i}$ as follows.

$$\begin{aligned}
&P(\alpha_1\alpha_2\alpha_3...\alpha_{2^K}) \\
&\triangleq \Pr(\hat{u}_{2^K(i-1)+1}=\alpha_1, \hat{u}_{2^K(i-1)+2}=\alpha_2,..., \hat{u}_{2^K i}=\alpha_{2^K}, \hat{u}_1^{2^K(i-1)}=z_1^{2^K(i-1)}) \\
&= \Pr(\widehat{out}_1 = \hat{\vec{u}}_{2^K,i}\boldsymbol{U(1)}, \widehat{out}_2 = \hat{\vec{u}}_{2^K,i}\boldsymbol{U(2)},..., \widehat{out}_{2^K} = \hat{\vec{u}}_{2^K,i}\boldsymbol{U(2^K)}, \hat{u}_1^{2^K(i-1)}=z_1^{2^K(i-1)}) \\
&= \Pr(\widehat{out}_1 = \vec{\alpha}_{2^K}\boldsymbol{U(1)}, \hat{u}_1^{2^K(i-1)}=z_1^{2^K(i-1)}) \times \Pr(\widehat{out}_2 = \vec{\alpha}_{2^K}\boldsymbol{U(2)}, \hat{u}_1^{2^K(i-1)}=z_1^{2^K(i-1)}) \\
&\quad \times ... \times \Pr(\widehat{out}_{2^K} = \vec{\alpha}_{2^K}\boldsymbol{U(2^t)}, \hat{u}_1^{2^K(i-1)}=z_1^{2^K(i-1)}) \\
&= a_1(\vec{\alpha}_{2^K}\boldsymbol{U(1)})a_2(\vec{\alpha}_{2^K}\boldsymbol{U(2)})...a_{2^{K-1}}(\vec{\alpha}_{2^K}\boldsymbol{U(2^{K-1})}) \times b_1(\vec{\alpha}_{2^K}\boldsymbol{U(2^{K-1}+1)})...b_{2^{K-1}}(\vec{\alpha}_{2^K}\boldsymbol{U(2^K)})
\end{aligned} \tag{3.11}$$

where $\vec{\alpha}_{2^K} \triangleq (\alpha_1, \alpha_2, \ldots \alpha_{2^K})$ is a vector consisting of 2^K binary 0 or 1.

According to (3.11), since $P(\alpha_1\alpha_2...\alpha_{2^K})$ is the joint probability of $\hat{u}_{2^K(i-1)+1}=\alpha_1$, $\hat{u}_{2^K(i-1)+2}=\alpha_2$, …, $\hat{u}_{2^K i}=\alpha_{2^K}$ and $\hat{u}_1^{2^K(i-1)}=z_1^{2^K(i-1)}$, it is just the metric of length-$2^K i$ path $(z_1^{4i-4}\alpha_1\alpha_2...\alpha_{2^K})$. Therefore, with $a_1(0)$, $a_1(1)$,..., $a_{2^{K-1}}(0)$, $a_{2^{K-1}}(1)$, $b_1(0)$, $b_1(1)$,..., $b_{2^{K-1}}(0)$ and $b_{2^{K-1}}(1)$ output from stage-(m-K) and (3.11), MCU can directly output the actual metrics of 2^{2^K} length-$2^K i$ paths.

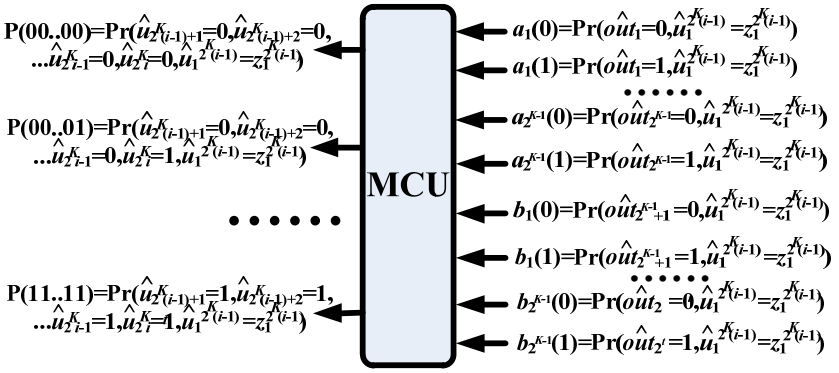


Fig. 3.12. Block diagram of MCU for 2^Kb-SCL decoder.

Zero-Forcing Unit (ZFU)

Similar to the 2-bit-decision case, the function of ZFU in 2^K-bit-decision scenario is also to force the metric of unqualified length-2^K paths to 0. Therefore, we can derive the function of ZFU for 2^Kb-SCL decoder as follows:

Assign $M(\alpha_1\alpha_2...\alpha_{2^K}) = P(\alpha_1\alpha_2...\alpha_{2^K})$ for path $(z_1^{4i-4}\alpha_1\alpha_2...\alpha_{2^K})$ with α_1, α_2,…, $\alpha_{2^K} \in \{0,1\}$;

If $\hat{u}_{2^K(i-1)+1}$ is frozen, then reassign all $M(1\alpha_2\alpha_3...\alpha_{2^K}) = 0$;

If $\hat{u}_{2^K(i-1)+2}$ is frozen, then reassign all $M(\alpha_1 1\alpha_3...\alpha_{2^K}) = 0$;

.....

If $\hat{u}_{2^K i}$ is frozen, then reassign all $M(\alpha_1\alpha_2...\alpha_{2^K-1}1) = 0$. (3.12)

Based on MCU in (3.11) and ZFU in (3.12), we can develop a general 2^Kb-SCL decoding algorithm as shown in Scheme 3.2. Fig. 3.13 shows the decoding procedure of 2^Kb-SCL algorithm with list size L. It can be seen that during the decoding procedure $2^{2^K}L$ metrics of candidate paths are compared each time, and the L paths with larger $M(\alpha_1\alpha_2...\alpha_{2^K})$ metrics are selected as the survival paths. As a result, 2^K successive bits can be determined simultaneously.

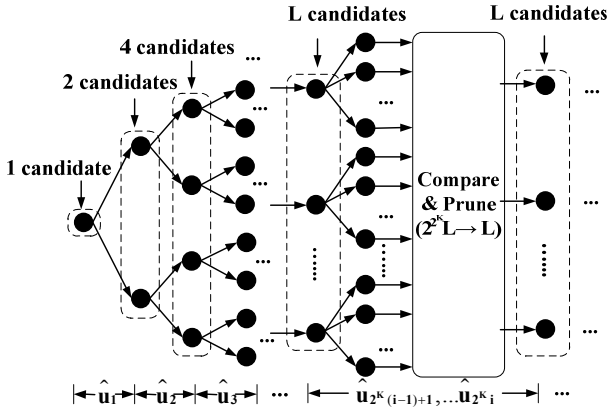


Fig. 3.13. L-size decoding scheme of 2^Kb-SCL.

Table 3.3 lists the latency of 2^Kb-SCL decoder with different values of K for (n, k) polar codes. From this table it can be seen that 2b-SCL decoder in Section 3.3.1 can be viewed as the specific case of 2^Kb-SCL with K=1. For a general 2^Kb-SCL decoder, its latency is $n/2^{K-2}$-2 clock cycles. Therefore, as K increases, the overall latency is reduced. In an extreme case, when K reaches m=$\log_2 n$, the 2^Kb-SCL decoder becomes a maximum likelihood (ML) decoder with latency as small as only 2 cycles.

Scheme 3.2: 2^K-bit SCL decoding (2^Kb-SCL) with list size L for (n, k) polar codes

1: Input: *Likelihoods of each bit in the received codeword*

2: For $i = 1$ **to** $n / 2^K$

3: For each length-$(2^K(i-1))$ **survival path** $(\hat{u}_1 \ldots \hat{u}_{2^K(i-1)}) \triangleq z_1^{2^K(i-1)}$

4: SC decoding:

5: *Activate stage* $-1 \sim$ *stage* $-(m-t)$ *of SC component decoder*

6: *stage* $-(m-K)$ *outputs* $a_1(0), a_1(1) \ldots, a_{2^{K-1}}(0), a_{2^{K-1}}(1), b_1(0), b_1(1) \ldots, b_{2^{K-1}}(0), b_{2^{K-1}}(1)$

7: Path Expansion:

8: *Expand survival path* $z_1^{2^K(i-1)}$ *to* 2^{2^K} *candidate paths* $(z_1^{2^K(i-1)} \hat{u}_{2^K(i-1)+1} \ldots \hat{u}_{2^K i})$:

9: *1 length* $-(2^K(i-1))$ *path* $\Rightarrow 2^{2^K}$ *length* $-(2^K i)$ *paths*

10: Metric Computation:

11: *Calculate actual path metrics* $P(\alpha_1 \alpha_2 \ldots \alpha_{2^K})$ *for* 2^{2^K} *length* $-(2^K i)$ *paths*:

12: $P(\alpha_1 \alpha_2 \ldots \alpha_{2^K}) = a_1(\vec{\boldsymbol{\alpha}}_{2^K}\mathbf{U}(1)) a_2(\vec{\boldsymbol{\alpha}}_{2^K}\mathbf{U}(2)) \ldots a_{2^{K-1}}(\vec{\boldsymbol{\alpha}}_{2^K}\mathbf{U}(2^{K-1}))$

13: $\times b_1(\vec{\boldsymbol{\alpha}}_{2^K}\mathbf{U}(2^{K-1}+1)) \ldots b_{2^{K-1}}(\vec{\boldsymbol{\alpha}}_{2^K}\mathbf{U}(2^K))$ *for path* $(z_1^{2^K(i-1)} \alpha_1 \alpha_2 \ldots \alpha_{2^K})$

14: Forcing Zero:

15: *Calculate the new path metrics* $M(\alpha_1 \alpha_2 \ldots \alpha_{2^K})$ *with forcing - zero operation*:

16: $M(\alpha_1 \alpha_2 \ldots \alpha_{2^K}) = P(\alpha_1 \alpha_2 \ldots \alpha_{2^K})$ *for path* $(z_1^{2^K(i-1)} \alpha_1 \alpha_2 \ldots \alpha_{2^K})$ *with* $\alpha_1, \ldots, \alpha_{2^K} \in \{0,1\}$

17: $\hat{u}_{2^K(i-1)+1}$ *is frozen* $\Rightarrow$ *all* $M(1 \alpha_2 \alpha_3 \ldots \alpha_{2^K}) = 0;$

18: **.....**

19: $\hat{u}_{2^K i}$ *is frozen* $\Rightarrow$ *all* $M(\alpha_1 \alpha_2 \ldots \alpha_{2^K - 1} 1) = 0.$

20: End for

21: Compare and Prune:

22: *Compare metrics* $M(\alpha_1 \alpha_2 \ldots \alpha_{2^K})$ *of all the* 2^{2^K} *L length* $-(2^K i)$ *candidate paths*

23: *Select L paths with the L largest metrics as the new survival paths*

24: End for

25: Output: *Choose the length - n survival path with the largest metric*

Table 3.3. Latency of 2^Kb-SCL decoder with different K

K	**Decoding latency (clock cycles)**	**Note**
K=0	$3n$-2	Original SCL
K=1	$2n$-2	2b-SCL
K=2	n-2	4b-SCL
K=3	n/2-2	8b-SCL
…	…	…
K=K	$n/2^{K-2}$-2	2^Kb-SCL (general case)
…	…	…
K=m=$\log_2 n$	2	Maximum Likelihood (ML) decoder

Although the increase of K can lead to the reduction of latency, K cannot be set too large for hardware implementation. That is because when K increases, the number of candidate paths, as 2^{2^K}, increases rapidly. As a result, a large K causes a large amount of path candidates and hence significantly increases the overall complexity of metric computation and path metrics comparison. For example, when K=m=$\log_2 n$ (ML decoder), the number of path candidates is 2^n. For (1024, 512) polar codes, that means 2^{1024} path metrics need to be computed and compared. The implementation of these extensive operations will cause ultra-large silicon area and ultra-long critical path. As a result, for practical implementation K is suggested to be set as no more than 3, which can achieve a good tradeoff between latency reduction and computation overhead.

3.3.3 Decoding Performance

Because the proposed reduced-latency SCL decoding algorithms only avoid the unnecessary metric computations but do not change the accuracy of metric computation, there is no performance loss for the proposed SCL algorithms over the original SCL algorithm. This is consistent with the simulation results shown in Fig. 3.14.

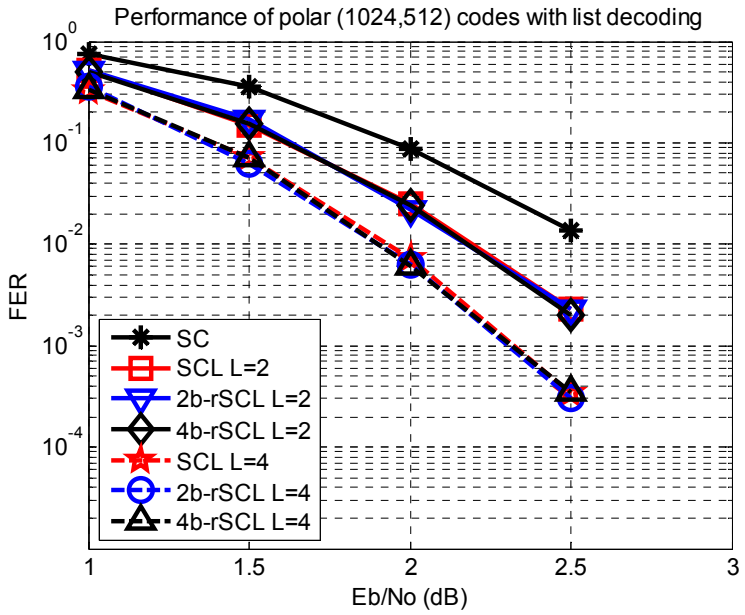


Fig. 3.14. Performance of 2^Kb-SCL algorithms.

3.4 Hardware Architecture

In this section, the hardware architectures of the reformulated SCL (2^Kb-SCL) decoders are presented. Different values of K correspond to different 2^Kb-SCL decoders. For simplicity, in this section we focus on K=1 and K=2 cases, which correspond to the 2b-SCL decoder and 4b-SCL decoder. Architectures with values of other K can be developed with a similar way.

As shown in Fig. 3.10, the difference between SC component decoder of 2b-SCL or 4b-SCL decoders and that of original SCL decoder is on the last 1 or 2 stages. Therefore, the other stages (**f**/**g** units) of original SC decoder are still used in the reformulated SCL decoders. As a result, in this section we focus on the architecture design of **f**/**g** units in the SC component decoder, MCU/ZFU in the reformulated stage, and metric sorting block, respectively.

3.4.1 Processing Elements

As indicated in Section 3.2, the likelihood-based function of **f** and **g** units are described in (3.1)-(3.4). However, these equations contain multiplication which is not feasible for hardware implementations. As a result, in order to simplify computation, the log-likelihood-based **f** and **g**

units are used in our design. In this case, the likelihood-based (3.1)-(3.4) are reformulated to the following equations:

$$c(0) = max*(a(0)+b(0),\ a(1)+b(1)) \tag{3.13}$$

$$c(1) = max*(a(0)+b(1),\ a(1)+b(0)) \tag{3.14}$$

$$d(0) = a(\hat{u}_{sum})+b(0) \tag{3.15}$$

$$d(1) = a(1-\hat{u}_{sum})+b(1)\ , \tag{3.16}$$

where $max*(x, y)=max(x, y) + ln(e^{-|x-y|})$ represents the Jacobian logarithm.

Notice that (3.13) and (3.14) contain logarithmic operation (ln(•)), which needs to be implemented using complex look-up table (LUT) with a long critical path. Fortunately, in [51] it was shown that the logarithmic item can be ignored with negligible performance loss. As a result, (3.13) and (3.14) can be further simplified as:

$$c(0) = max(a(0) + b(0), a(1) + b(1)) \tag{3.17}$$

$$c(1) = max(a(0) + b(1), a(1) + b(0)) \tag{3.18}$$

In general, equations (3.15)-(3.18) describe the log-likelihood version of **f** and **g** units. Based on these equations, the basic processing element (PE) of the SC component decoder, which contains an **f** unit and a **g** unit, is developed and is shown in Fig. 3.15. Here, *C&S* unit represents the combined comparator and 2-to-1 selector. In addition, *ctrl* signal is the control signal that indicates whether the PE functions as an **f** unit or a **g** unit.

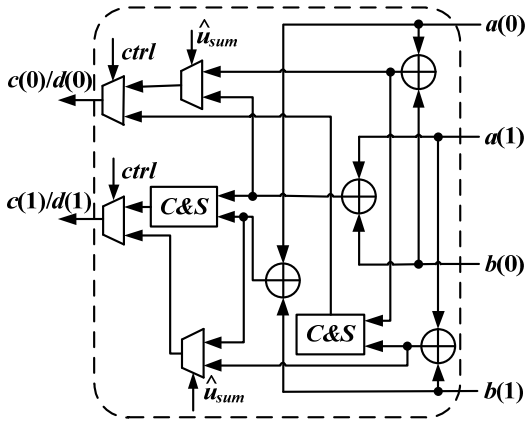


Fig. 3.15. Architecture of PE in the SC component decoder.

3.4.2 Metric Computation Unit and Zero-Forcing Unit

As shown in Fig. 3.10, MCU and ZFU are the two essential parts in 2^Kb-SCL decoders to help them decide multiple bits. Similar to the case in Section 3.4.1, the likelihood-based functions of MCU and ZFU need to be transformed to log-likelihood version as well.

For K=1 case that corresponds to 2b-SCL decoding algorithm, its likelihood-based functions of MCU and ZFU have been described in Scheme 3.1 (line10~line18). For the transformation for MCU, according to the transformation principle in Section 3.4.1, $P(pq)=a(p)b(q)$ in the line-12~line13 of Scheme 3.1 is transformed to $a(p)+b(q)$. In addition, since ln0 is negative infinite, $M(pq)$=0 (line-17~ line-18 in Scheme 3.1), as the likelihood-based function of ZFU, is reformulated to $M(pq)=-Inf$ and where $-Inf$ represents negative infinite. As a result, the hardware architecture of MCU and ZFU for 2b-SCL decoder is developed as shown in Fig. 3.16(a). Here the *ctrl1* and *ctrl2* in Fig. 3.16(a) are the two control signals that indicate whether $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ are information bits or not.

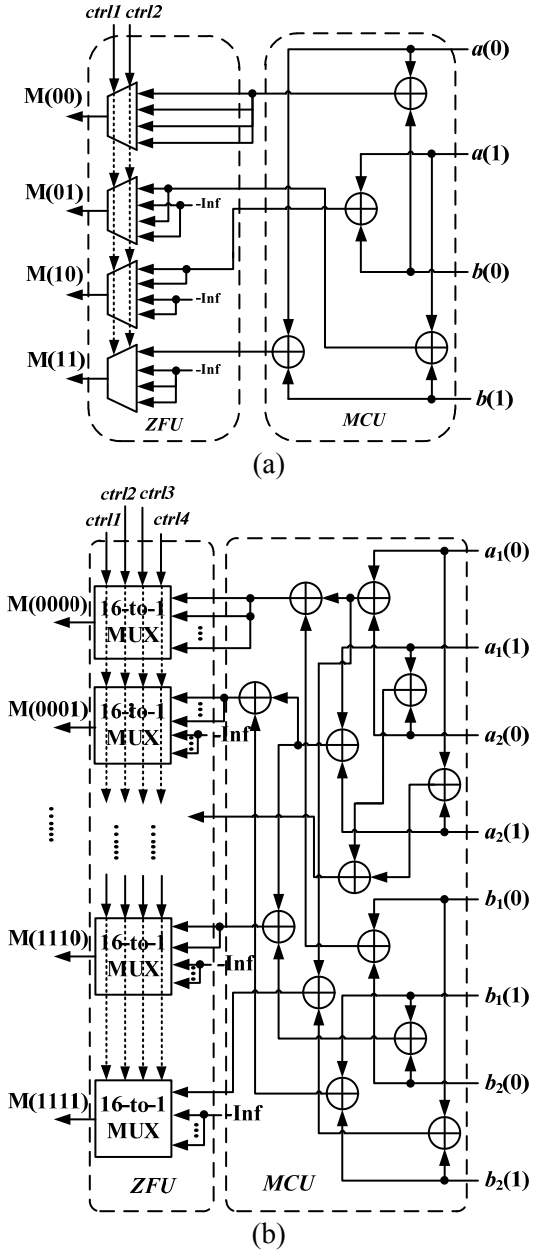


Fig. 3.16. Architecture of MCU+ZFU
(a) 2b-SCL (b) 4b-SCL decoders.

For K=2 case that corresponds to 4b-SCL decoding algorithm, its likelihood-based function of MCU and ZFU can be derived from Scheme 3.2 (line10~line19). For the function of MCU (line12~line13), in K=2 case it is $P(\alpha_1\alpha_2\alpha_3\alpha_4)=a_1(\alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4)a_2(\alpha_2 \oplus \alpha_4)b_1(\alpha_3 \oplus \alpha_4)b_2(\alpha_4)$. Then, with the likelihood-to-log-likelihood transformation, it is reformulated as

$P(\alpha_1\alpha_2\alpha_3\alpha_4)=a_1(\alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4)+a_2(\alpha_2 \oplus \alpha_4)+b_1(\alpha_3 \oplus \alpha_4)+b_2(\alpha_4)$. For the function of ZFU (line16~line19), in K=2 case it is $M(\alpha_1\alpha_2\alpha_3\alpha_4)=0$. Therefore, its log-likelihood version is $M(\alpha_1\alpha_2\alpha_3\alpha_4)=$-*Inf*. As a result, the architecture of MCU and ZFU for 4b-rSCL decoders are developed as shown in Fig. 3.16 (b). Here the *ctrl1, ctrl2, ctrl3* and *ctrl4* in Fig. 3.16(b) are the four control signals that indicate whether $\hat{u}_{4i-3}$, $\hat{u}_{4i-2}$, $\hat{u}_{4i-1}$ and $\hat{u}_{4i}$ are information bits or not.

3.4.3 Metric Sorting Block

After MCU and ZFU generate the metrics for different paths, a sorting block is needed to compare those 2L metrics and select the L paths with larger metrics. In the proposed designs, we use the bitonic sorting algorithm [52] to find out the L larger metrics. Fig. 3.17 illustrates an example architecture of the proposed 8-input 4-output metric sorting block. It contains a 4x4 increasing order bitonic sorter and a 4x4 decreasing order bitonic sorter. Each bitonic sorter is constructed by 2x2 increasing order sorters (*IOS*) and 2x2 decreasing order sorters (*DOS*). With the help of the two 4x4 bitonic sorters, in_1~in_4 are sorted as an array with increasing order ($i_1 \le i_2 \le i_3 \le i_4$) while in_5~in_8 are sorted as an array with decreasing order ($d_1 \ge d_2 \ge d_3 \ge d_4$). Then, these two pre-sorted arrays are sent to a stage of 4 *C&S* units. At the output end of these C&S units, the 4 larger elements among in_1~in_8 are found as $out_j=max(i_j, d_j)$, where j=1, 2, 3 and 4.

As indicated in [52], the critical path delay of a 2^sx2^s bitonic sorter is $1+2+\ldots+s=s(s+1)/2\ T_{C\&S}$, where $T_{C\&S}$ is the critical path delay of C&S unit. Therefore, a general 2^s-input 2^{s-1}-output metric sorting block consisting of two 2^{s-1}x2^{s-1} bitonic sorters and a stage of C&S units has an overall critical path delay of $1+2+...s-1+1=1+(s-1)s/2\ T_{C\&S}$.

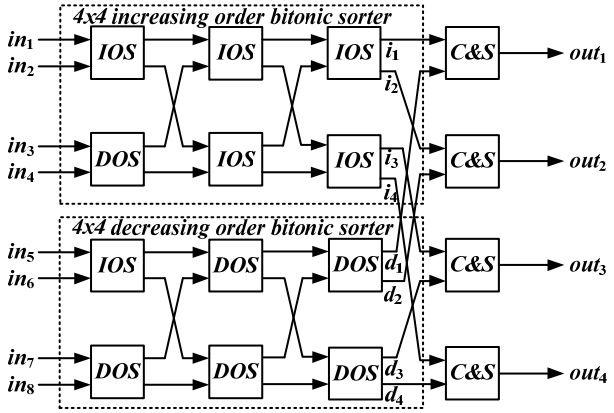


Fig. 3.17. Architecture of 8-input 4-output sorting block.

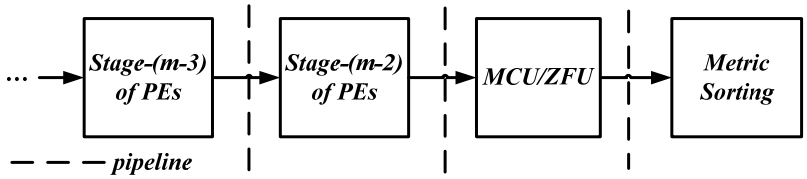
Notice that the proposed metric sorting block is a 2^s-input 2^{s-1}-output (2^sx2^{s-1}) sorter that can find the 2^{s-1} largest elements among the 2^s inputs. Since for the proposed L-size 2^Kb-SCL decoder, it only needs to find the L largest metrics among $2^{2^K} L$ candidates, hence the 2^sx2^{s-1} sorter is enough for this sorting task and we do not need the full-size sorting (2^sx2^s) function. Consider the critical path delays of 2^sx2^{s-1} and 2^sx2^s sorters are $1+(s\text{-}1)s/2\ T_{C\&S}$ and $s(s+1)/2\ T_{C\&S}$, respectively, the proposed metric sorting block can save s-1 $T_{C\&S}$ on the critical path delay than the conventional full-size sorter. Since the data path in sorting block is always the critical path of the overall list decoder, the decoder with the proposed metric sorting block can achieve higher clock frequency than the one with the conventional sorter.

3.4.4 Data Path Balancing

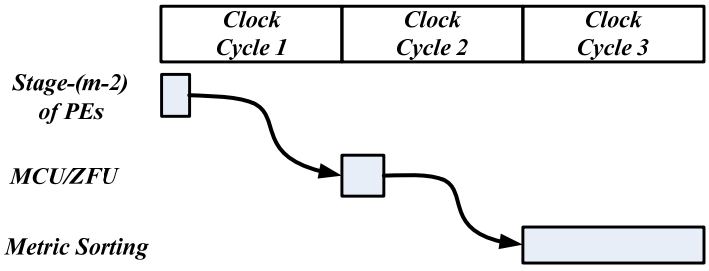
As discussed in Section 3.4.3, the critical path delay of 2^s-input 2^{s-1}-output metric sorting block is $1+(s\text{-}1)s/2\ T_{C\&S}$. This is much larger than the critical path delay of PE or MCU/ZFU. For example, for a 4b-SCL decoder with L=2, s=$\log_2(16*2)$=5. Then the critical path delay of metric sorting block is $11T_{C\&S}$, while the critical path delays of PE and MCU/ZFU are less than $3T_{C\&S}$. Because the clock speed is upper-bounded by the critical path delay, the throughput of reformulated SCL decoder is limited by the long critical path of metric sorting block.

Considering the unbalanced data path between metric sorting block and other parts of

reformulated SCL decoder, we propose to re-pipeline those data paths to reduce critical path delay. Fig. 3.18(a) shows the original pipelining of 4b-SCL decoder. Here the register arrays for pipelining are inserted between different blocks of SCL decoder. As a result, due to the unbalanced data path between different blocks, the clock cycles for processing PEs and MCU/ZFU are not fully utilized (see Fig. 3.18(b)). Fig. 3.18 (c) shows the proposed re-pipelining strategy to the same 4b-rSCL decoder. It can be seen that the original registers between stage-(*m*-2), MCU/ZFU and metric sorting block are moved into metric sorting block. Fig. 3.18(d) shows the corresponding timing chart after re-pipelining. It can be seen that the data path in each clock cycle is balanced. More importantly, since metric sorting block is deeply pipelined, the overall critical path delay is reduced significantly. Notice that in Fig. 3.18(c) the metric sorting block is 2-stage pipelined. If deeper pipelining is needed, we need to move the registers between other stages of PE into metric sorting block. For example, in order to perform 3-stage pipeline to metric sorting block, we need to move the registers between stage-(*m*-3) and stage-(*m*-2) in Fig. 3.18(c) into metric sorting block as well.



(a)



(b)

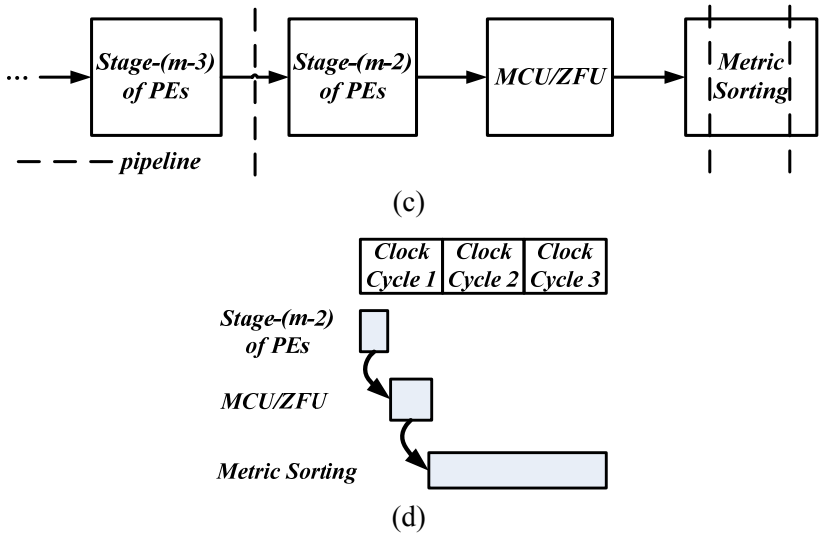


Fig. 3.18. Timing chart of last stages.
(a) Original pipelining for 4b-SCL decoder. (b) Original timing chart. (c) Re-pipelining for 4b-SCL decoder. (d) Timing chart with balanced data path.

The proposed data path balancing strategy is very useful for high-speed polar list decoder design. For practical use of polar codes, in order to achieve comparable error-correcting performance with LDPC or Turbo codes with the similar code-length, a large list size L is required. For example, [6] reported that the 2048-length polar codes can achieve beyond LDPC performance under the condition of L=32. In that case, for the conventional SCL decoder, the s for sorting block is $\log_2(2*32)=6$. As a result, even the proposed metric sorting block is used, the critical path delay is still very large ($1+(s-1)s/2\ T_{C\&S}=16T_{C\&S}$), which impedes the application of polar codes in high-speed systems. Notice that this phenomenon becomes even more severe for 2^Kb-SCL decoder. For example, for 4b-SCL decoder with L=32, the number of path metric candidates is 32*16=512, which corresponds to $s=\log_2 512=9$. As a result, the critical path delay of metric sorting block increases to $1+(s-1)s/2\ T_{C\&S}=37T_{C\&S}$. However, if we apply the proposed data path balancing technique to this case, the critical path delay can be significantly reduced. For example, in the case of 2048-length polar codes with L=32, with the balance of the data path of metric sorting block, MCU/ZFU block and all the stages of PE (stage-1~stage-9), the critical path delay

of 4b-SCL decoder after data path balancing is less than (37+3+3*9)/11≈6.1$T_{C\&S}$. This new critical path delay is 4 times less than the case without use of data path balancing, and it is even 1.5 times less than that of the original SCL decoder. As a result, the use of the proposed data path balancing strategy guarantees the high-speed design of polar list decoder.

3.4.5 Quantization Scheme and Memory Requirement

Similar to the case of SCL decoders, the architecture of 2^Kb-SCL decoders contain multiple stages of PE. As a result, in order to avoid saturation problem that is pointed out in [51], the quantization schemes for different stages of PE are different. If we assume the log-likelihood (LL) information from channel is quantized as Q_{ch} bits, then for the stage-i of 2^Kb-SCL decoder, the corresponding bit-width is $Q_{ch}+i$. In addition, for the MCU/ZFU and metric sorting blocks, they are quantized with $Q_{ch}+m$ bits. Notice that because the LL information in different stages has different bit-widths, the corresponding memories that store the LL information have different bit-widths as well.

Besides the aforementioned blocks, a large portion of the 2^Kb-SCL decoders is the memory banks. Similar to SCL decoders [51], multi-bit-width memory banks in the proposed design store the LL information from the channel as well as the LL information processed by each stage. Because the quantization scheme for LL information is non-uniform and varies depending on the corresponding stages, the memory banks for different stages have different bit-widths. In addition, 1-bit-width memory banks are needed to store the updated survival paths and partial sum bits $\hat{u}_{sum}$. Notice that compared to [51], the memory requirement of the proposed 2^Kb-SCL decoder is larger. This is because the number of path metric candidates increases in the proposed decoders. As a result, more memories are required for storing the calculated metrics from MCU/ZFU block. For example, with L=32 and K=2, 32*16=512 LL messages for metrics needs to be stored, while SCL decoder only needs to store 64 LL message for metrics. Consider these metrics are always

quantized to more than 10 bits, the extra memory requirement of 2^Kb-SCL decoder causes inevitable area overhead, especially in the case of large L or K.

3.4.6 Overall Architecture

Based on the aforementioned PE, ZFU&MCU and metric sorting block, the overall architecture of an L-size reformulated SCL decoder can be developed as illustrated in Fig. 3.19. Besides the above presented blocks, the decoder needs LL memory bank to store and update the log-likelihood information that are processed by L SC component decoders. In addition, survival path bank is also needed to store and update the L survival paths during the list decoding procedure. Besides that, the reformulated SCL decoder needs a polar-encoder-like partial sum generator (PSG) to compute $\hat{u}_{sum}$ for corresponding SC component decoder. The architecture of PSG is similar to the polar encoder.

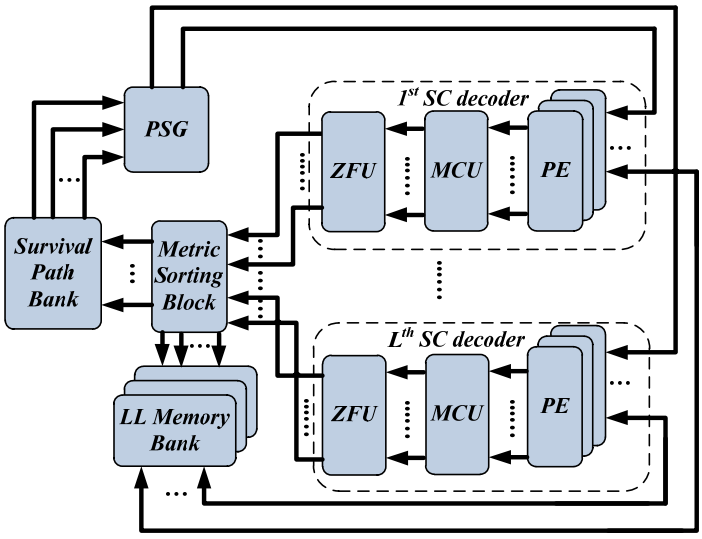


Fig. 3.19. Overall architecture of the new SCL decoders.

3.5 Hardware Performance

In this section, the hardware performance characteristics of the proposed reformulated SCL decoding architectures are analyzed. Table 3.4 shows the hardware performance of different SCL

decoders with list size L=2 and 4 for polar (1024, 512) code. Here the designs of 2b-SCL decoder and 4b-SCL decoder are synthesized by Synopsys Design Compiler with ST CMOS 65nm library. Notice that in the proposed designs 3-bit quantization scheme is used for the LL information output from channel, which is the same as in [51]. Based on the quantization scheme described in Section 3.4.5, the bit width of stage-i is 3+i. For the MCU/ZFU block and metric sorting block, they are quantized to 3+m=13 bits.

From Table 3.4 it can be seen that, compared with prior LL-based SC list decoder design [51], the proposed 2b-SCL decoder and 4b-SCL decoder can achieve 21.0% and 60.5% reduction in latency, respectively. Notice these reductions are less than the analysis in Table 3.3. This is because the latency listed in Table 3.4 is calculated based on the (12) in [51], where code rate R=k/n is considered, while the analysis in Table 3.3 discuss the general case without the specific discussion on different code rate or distribution of frozen bit positions. In general, as the code rate increases, the proposed reformulated SCL decoders can save more clock cycles than the original one in [51]. For example, for an R=1 polar code, 2b-SCL decoder and 4b-SCL decoder can achieve 33% and 66% less latency than the original SCL decoder, respectively.

With the use of data path balancing technique, the proposed 2b-SCL and 4b-SCL designs can achieve high clock frequency. Therefore, as seen in Table 3.4, the coded throughputs of 2b-SCL decoder and 4b-SCL decoder with L=2 are 1.66 times and 3.45 times of that of original SCL decoder, respectively. In addition, when L=4, the coded throughputs of 2b-SCL decoder and 4b-SCL decoder are 2.11 times and 3.23 times of that of original SCL decoder, respectively. Besides, the hardware efficiency of our designs, which is defined as the ratio of throughput to area, increases as well. When L=2, the hardware efficiencies of 2b-SCL and 4b-SCL decoders are 1.36 times and 2.08 times of that of original SCL decoder; when L=4, the hardware efficiencies of 2b-SCL and 4b-SCL decoders are 1.87 times and 2.66 times of that of original SCL decoder.

Table 3.4. Hardware performance of (n=1024, k=512) SC list decoders.

Hardware	SCL [51]	2b-SCL	4b-SCL
List size	2		
CMOS technology (nm)	90	65	65
Area(mm^2) (scaled to 65nm)	0.8	0.97	1.06
Clock frequency (MHz)	459	600	500
Latency (clock cycles)	2592*	2046	1022
Coded Throughput (Mbps)	181	300	501
Hardware efficiency (Mbps/mm^2) †	226.2	309.2	472.6
Power Consumption (mW)	N/A	321	395
List size	4		
Area(mm^2) (scaled to 65nm)	1.76	1.98	2.14
Clock frequency (MHz)	314	525	400
Latency (clock cycles)	2592*	2046	1022
Coded Throughput (Mbps)	124	262	401
Hardware efficiency (Mbps/mm^2) †	70.4	132.3	187.3
Power Consumption (mW)	N/A	734	718

***** Decoding latency of [16] is calculated based on the equation (12) in [51].
† Hardware Efficiency is defined as Throughput/Area.

3.6 Conclusion

In this chapter, reformulated SC list decoding algorithms are presented. These reformulated algorithms can reduce the latency significantly without any performance loss. Then, based on the proposed algorithm, corresponding latency-reducing hardware architectures for SCL decoders are developed. Hardware analysis shows that the proposed 2b-SCL decoder and 4b-SCL decoder can achieve significant improvement in throughput and hardware efficiency.

Chapter 4

4 LLR-BASED SC LIST DECODER

In this chapter, we present the area-efficient SC list decoder. Section 4.2 proposes the LLR-based SC list decoding algorithm. In Section 4.3, hardware architecture of the decoder is presented. Section 4.4 analyzes and compares the hardware performance of the decoder with the prior works.

4.1 Introduction

Chapter 3 proposed a 2^Kb-SCL algorithm. However, this algorithm is based on the likelihood messages. For most communication systems, LLR messages are used for soft information for low complexity. As a result, the non-LLR-based 2^Kb-SCL decoder has much larger computation and memory complexity than the LLR-based SCL decoder in [31-32]. On the other hand, those LLR-based SCL decoders in are only able to determine one bit in one cycle; hence they have very long latency. As a result, the simultaneous reduction in both the latency and area has not been achieved in the prior works.

This chapter presents an LLR-based SCL decoding algorithm with 2^K bits decision, namely *LLR-2^Kb-SCL*. The proposed new algorithm can determine 2^K bits in one cycle for arbitrary K with the use of LLR messages. As a result, it can achieve both low-complexity and short latency. Based on this new algorithm, a VLSI architecture of the SCL decoder is developed. Synthesis results show that for an example (1024, 512) polar code, the proposed LLR-4b-SCL decoder achieves great reduction in both area and latency as compared to the prior works, respectively.

4.2 LLR-based SC List Decoding Algorithm

4.2.1 LLR-based 2^Kb-SC List Decoding Algorithm

This section show how to determine each successive 2^K bits as $\hat{u}_{2^K(i-1)+1}$, …, $\hat{u}_{2^K i}$ at the same time

in the LLR form. Because he SCL algorithm can be interpreted from the view of coding tree. This means the multiple-bit decision indicates that the SCL decoder is able to directly calculate the metrics of length-($2^K i$) paths from the metrics of length-(2^K(i-1)) paths. In the proposed algorithm in this section, such direct computation is performed by an LLR-based metric computation unit, which replaces the original last K stages of each LLR-based SC decoder. In the following paragraph, we show how to derive the function of the LLR-based MCU.

Similar to Chapter 3, assume that the previously decoded $2^K(i\text{-}1)$ bits $\hat{u}_1$, …, $\hat{u}_{2^K(i-1)}$ are z_1, ..., $z_{2^K(i-1)}$, respectively. And this event is denoted as $\hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)}$. Therefore, in the logarithmic domain the length-($2^K i$) path metric can be represented as:

$$M(\alpha_1 \ldots \alpha_{2^K}, z_1^{2^K(i-1)}) \triangleq \ln(\Pr(\hat{u}_{2^K(i-1)+1}^{2^K i} = \alpha_1^{2^K}, \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})), \tag{4.1}$$

where $\hat{u}_{2^K(i-1)+1}^{2^K i}$ is defined as ($\hat{u}_{2^K(i-1)+1}$, …, $\hat{u}_{2^K i}$) that is the set of the current 2^K decoded bits. In addition, $\alpha_1^{2^K}$ is defined as (α_1, …, α_{2^K}) whose elements are the binary values.

(4.1) contains the probabilistic information of the current 2^K decoded bits, which is unknown during the decoding procedure. To address this problem, we need to further represent the logarithmic path metrics with the LLR messages that are input to the MCU. Such reformulation is based on the fact that the polar decoding procedure is inherently "guided" by its encoding procedure. Simultaneous right-to-left decoding procedure of the successive 2^K bits (see Fig. 4.1(a)) involves the estimation of the left-to-right encoding procedure (see Fig. 4.1(b)). Hence if $\hat{u}_{2^K(i-1)+1}^{2^K i}$ is estimated to be $\alpha_1^{2^K}$, then $\widehat{out}_1^{2^K} \triangleq (\widehat{out}_1, \ldots, \widehat{out}_{2^K})$ should be the estimation of $\alpha_1^{2^K} \boldsymbol{U}$, where $\boldsymbol{U}$ is the 2^K-by-2^K generator matrix. As a result, (4.1) can be further re-written as:

$$M(\alpha_1 \ldots \alpha_{2^K}, z_1^{2^K(i-1)}) = \ln(\Pr(\widehat{out}_1^{2^K} = \alpha_1^{2^K} U, \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})) \tag{4.2}$$

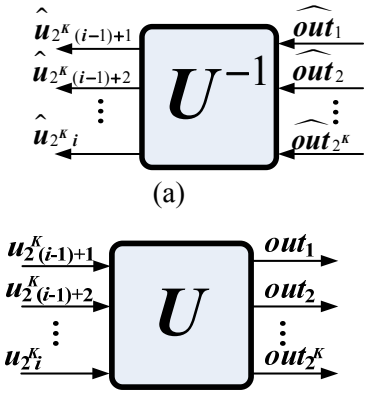


Fig. 4.1. encoding and decoding of 2^K bits
(a) right-to-left decoding of 2^K bits. (b) left-to-right encoding bits.

Notice that the determination of $\widehat{out}_1,\ldots,\widehat{out}_{2^K}$ are independent. In addition, if we denote the j-th column vector of $\boldsymbol{U}$ as $\boldsymbol{U}(j)$, then we have $\widehat{out}_j = \alpha_1^{2^K}\boldsymbol{U}(j)$. As a result, (4.2) can be further derived as below:

$$\begin{aligned} M(\alpha_1\ldots\alpha_{2^K}, z_1^{2^K(i-1)}) &= \ln(\Pr(\widehat{out}_1^{2^K} = \alpha_1^{2^K}U \mid \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})\Pr(\hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})) \\ &= \sum_{j=1}^{2^K}\ln(\Pr(\widehat{out}_j = \alpha_1^{2^K}U(j) \mid \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})) + M(z_1^{2^K(i-1)}) \end{aligned} \tag{4.3}$$

where $M(z_1^{2^K(i-1)}) = \ln(\Pr(\hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)}))$ is the logarithmic length-($2^K(i$-1)) path metric.

Recall that each SC component decoder is based on LLR form. In that case, the j-th input to the MCU block is: $s_j = \ln\left(\dfrac{\Pr(\widehat{out}_j = 0 \mid \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})}{\Pr(\widehat{out}_j = 1 \mid \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)})}\right)$

As a result, we can obtain the elements of the first item in (4.3):

$$\Pr(\widehat{out}_j = 0 \mid \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)}) = \frac{e^{s_j}}{e^{s_j}+1},\quad \Pr(\widehat{out}_j = 1 \mid \hat{u}_1^{2^K(i-1)} = z_1^{2^K(i-1)}) = \frac{1}{e^{s_j}+1}. \tag{4.4}$$

Substituting (4.4) into (4.3), we have:

$$M(\alpha_1\ldots\alpha_{2^K}, z_1^{2^K(i-1)}) = \sum_{j=1}^{2^K}(s_j(1-\alpha_1^{2^K}U(j)) - \ln(e^{s_j}+1)) + M(z_1^{2^K(i-1)}). \tag{4.5}$$

(4.5) describes the LLR-based update principle for path metrics. Once the MCU block receives

the 2^K input LLR messages s_j and the previous metric of length-($2^K(i$-1)) path, it can immediately calculate the new metric of length-(2^Ki) path with the use of (4.5), which corresponds to the simultaneous decision for 2^K bits.

Notice that (4.5) contains exponential and logarithmic functions, which require long critical paths in hardware design. Therefore, (4.5) needs to be simplified for feasible VLSI implementation.

Consider $\ln(1+e^x) \approx \begin{cases} x & for\ large\ x \\ 0 & for\ small\ x \end{cases}$, (4.5) can be further approximated as below:

$$M(\alpha_1 ... \alpha_{2^K}, z_1^{2^K(i-1)}) \approx \sum_{j=1}^{2^K} (s_j(1-\alpha_1^{2^K} U(j)) - \delta(s_j)) + M(z_1^{2^K(i-1)}), \tag{4.6}$$

where $\delta(s_j) = s_j$ if $s_j \geq 0$; otherwise 0.

(4.6) shows how to directly calculate the metric of length-(2^Ki) paths from the metric of length-($2^K(i$-1)) paths. With the use of this update principle, we can develop the LLR-based SCL decoding algorithm with 2^K bits decision as Scheme 4.1. In general, an L-size LLR-2^Kb-SCL decoder consists of L copies of LLR-based SC decoder. To decode every 2^K successive bits, each SC component decoder first performs the regular SC decoding procedure till the last-(m-K) stage, where m=$\log_2 n$. At this time, the (m-K) stage outputs 2^K LLR messages s_j (j=1, 2,…2^K) to the MCU block. Then, the MCU block in each SC component decoder calculates the new path metrics with the use of (4.6). After that, all of the updated path metrics from the L SC component decoders are compared and only L largest are selected as the metrics of the survival paths. The above entire procedure is repeated for every 2^K bits until all the n bits are determined. Notice that similar to Chapter 3, a simple zero-forcing unit (ZFU) is needed after the computation of (4.7), which helps to drop the unqualified paths that violate the frozen conditions.

Scheme 4.1: L-size LLR-2^Kb-SCL Algorithm for (n, k) polar codes

1: Input: *Log - Likihood ratios of each bit in the received codeword*

2: Initilization: *Path metric* $M_0 = 0$ *for each survival path*

3: For $i = 1$ **to** $n / 2^K$

4: For each length-$(2^K (i-1))$ **survival path** $\hat{u}_1^{2^K (i-1)} = z_1^{2i-2}$

5: SC component decoding:

6: *Activiate stage* - 1 *to stage* - $(m - K)$ *of LLR - based SC decoder*

7: *stage* - $(m - K)$ *output* 2^K *LLR - based message* s_j $(j = 1, 2, ..., 2^K)$

8: Path Expansion:

9: *Expand survival path* $z_1^{2^K (i-1)}$ *to* 2^{2^K} *candidate paths* $(\hat{u}_1^{2^K i} = (\alpha_1^{2^K}, z_1^{2i-2}))$

10: *1 length* - $(2^K (i-1))$ *path* $\Rightarrow 2^{2^K}$ *length* - $(2^K i)$ *paths*

11: Metric Computation:

12: *Calculate* 2^{2^K} *actual path metrics* $M(\alpha_1 ... \alpha_{2^K}, z_1^{2^K (i-1)})$ *by* (8)

13: Forcing Zero:

14: $M(\alpha_1 ... \alpha_{2^K}, z_1^{2^K (i-1)})$ *for path* $\hat{u}_1^{2^K i} = (\alpha_1^{2^K}, z_1^{2i-2})$

15: $\hat{u}_{2^K (i-1)+j}$ *is frozen* $\Rightarrow$ *all* $M(\alpha_1 \alpha_2 ... \alpha_{j-1} 0 \alpha_{j+1} ... \alpha_{2^K}, z_1^{2^K (i-1)}) = -\inf$

16: End for

17: Compare and Prune:

18: *Compare* $M(\alpha_1 ... \alpha_{2^K}, z_1^{2^K (i-1)})$ *for all the* 2^{2^K} *length* - $(2^K i)$ *candidate paths*

19: *Select L paths with the L largest metrics as the new survival paths*

20: End for

21: Output: *Choose the length - n survival path with the largest metric*

4.2.2 Decoding Performance

Fig. 4.2 shows the simulation results for the proposed LLR- 2^Kb-SCL algorithm with different combinations of L and K for (1024, 512) polar codes. Here the simulation environment is AWGN channel with BPSK modulation. From the figure it can be seen that the proposed LLR-2^Kb-SCL algorithm has no performance loss as compared to prior different SCL algorithms.

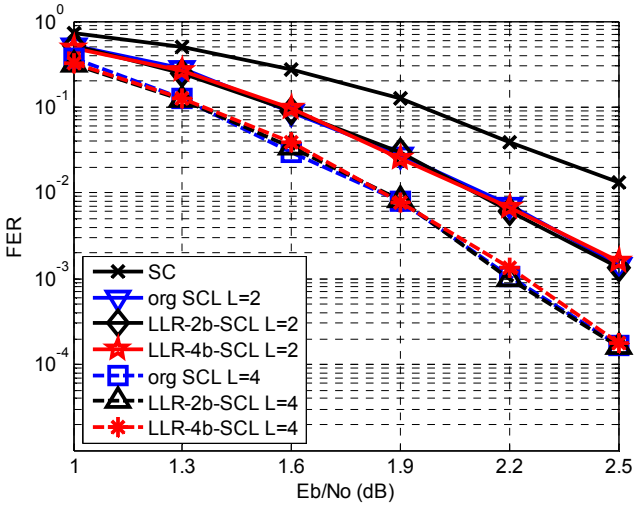


Fig. 4.2. Simulation results for (1024, 512) polar codes.

4.3 Hardware Architecture

4.3.1 Overall Architecture

Fig. 4.3 shows the overall hardware architecture of the proposed L-size LLR-2^Kb-SCL algorithm decoder. The data path of the entire decoder contains L LLR-based SC decoders plus a metric sorting block. For each SC component decoder, it is reformulated from the LLR-based decoder in [46], which remains the (m-K) stages but the last K stages are replaced by the LLR-based MCU and ZFU. In addition, the required memory resource of the entire decoder consists of register arrays, bulk memory and buffer for survival paths, path metrics, propagating LLR messages and channel outputs, respectively.

Because the hardware designs of the LLR-based SC component decoder and sorting block have been discussed in Chapter 2 and Chapter 3, respectively. In this section we focus on the design of MCU. Notice that since ZFU can be easily implemented with multiplexers; the analysis of this block is omitted as well.

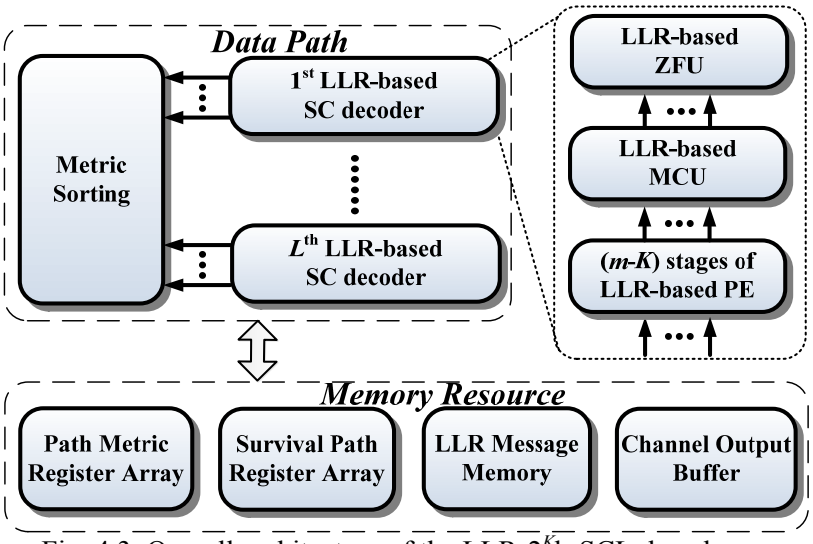
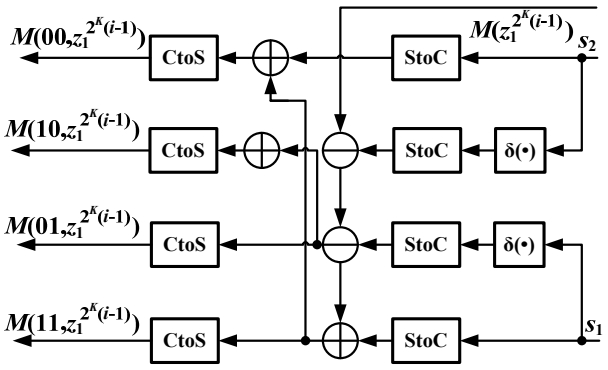


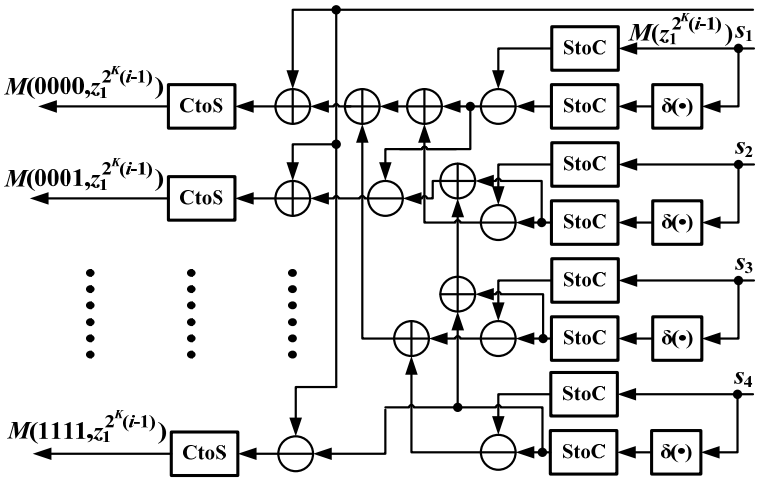
Fig. 4.3. Overall architecture of the LLR-2^Kb-SCL decoder.

4.3.2 LLR-based Metric Computation Unit

In Section 4.2.1 (4.6) describes the function of MCU. Since this function depends on K, the hardware design of MCU varies with different choices of K. Fig. 4.4 (a) and (b) illustrate the inner architecture of MCU for K=1 and 2, respectively. Here δ(•) block can be simply implemented with a multiplexer. In addition, StoC and CtoS blocks represent the components that perform the conversion between sign-magnitude and 2's complement forms.



(a)



(b)

Fig. 4.4. Architecture of MCU for (a) K=1. (b) K=2.

4.4 Hardware Performance

Table 4.1 shows the estimated technology-independent hardware performance of different q-bit SCL decoders for (1024, 512) polar codes with L=2 and K=2. Here all the three types of decoders are based on the tree architecture. From the table it can be seen that the proposed LLR-4b-SCL decoder has the same reduced-latency as [53] with about 50% reduction in complexity. This is because the unit gate count per PE in this work is only half of that in [53]. In addition, the required memory resource in this work is about half of that in [53] as well. Besides, compared with [31-32], the LLR-4b-SCL decoder has 66% reduction in latency than the LLR-SCL with small area overhead. As a result, the hardware efficiency of the proposed work, defined as throughput per gate, is 1.85 times than that of [31-32].

Table 4.1. Estimation of SCL decoders with L=2 and K=2

Design		This work	[31-32]	[53]
Form of soft information		LLR	LLR	LL
Multi-bit decision		Yes	No	Yes
PE	**Gate count**	12q	12q	24q
	# of PE	Ln=2048		
MCU	**Gate count**	320q	21q/2	192q
	# of MCU	L=2		
ZFU	**Gate Count**	240q	N/A	240q
	# of ZFU	L=2	N/A	L=2
Sorting	**Gate Count**	384q	20q	384q
Bits for LLR Memory		4094q	4094q	8188q
Bits for Path and Metrics		2q+2048		
Total Gate Count (Norm.)		~30176q	~28713q	~58588q
Latency (Clock Cycle)		n-2=1022	3n-2=3070	n-2=1022
Critical Path (Norm.)		1		
Throughput (Norm.)		3	1	3
Efficiency (Norm.)		2.85	1	1.47

***** Data path balancing technique [53] is applied to reduce the critical path delay.

Table 4.2 shows the technology-dependent hardware performance of different SCL decoders with different list size L for the same (1024, 512) polar codes. For the designs with different technology libraries, the results on area and frequency are scaled to the same 90nm nodes. From this table it can be seen that; the proposed designs have 75% less area than [53]. Notice that this reduction in area exceeds the estimation in Table 4.1. This is because the proposed design is implemented by semi-parallel architecture with 64 PEs in each SC component decoder, while [53] adopted tree-based architecture that needs n=1024 PEs per SC decoder.

Compared with the LLR-SCL decoders in [32], the proposed LLR-4b-SCL decoders have 60% shorter latency. In addition, because the propose designs can achieve high clock frequency with the use of data path balancing technique [53], they have 25% and 14% increase on decoding throughput than the works in [32] with list size 2 and 4, respectively. As a result, the hardware efficiency of the proposed designs are 2.2 times and 2.1 times of that in [32] with list size 2 and 4, respectively.

Table 4.2. Hardware performance of SCL decoders with K=2.

Design		This work	[32]	[53]
Form of soft information		LLR	LLR	LL
Multi-bit decision		Yes	No	Yes
CMOS Technology (nm)		65	90	65
Quantization scheme		6-bit	6-bit	Dynamic
List Size		2		
Area (mm^2)		0.26	0.88	1.06
	Scaled to 90nm	0.50		2.03
Freq. (MHz)		600	847	500
	Scaled to 90nm	433		361
Latency (Clock cycle)		1056	2589	1022
Throughput (Mbps)		419	335	360
Efficiency (Mbps/mm^2)		838	380	177
List Size		4		
Area (mm^2)		0.49	1.78	2.14
	Scaled to 90nm	0.94		4.10
Freq. (MHz)		500	794	400
	Scaled to 90nm	361		288
Latency (Clock cycle)		1056	2648	1022
Throughput (Mbps)		350	307	288
Efficiency (Mbps/mm^2)		372	172	70

4.5 Conclusion

In this paper, an LLR-2^Kb-SCL algorithm for polar codes decoding is presented.. The proposed algorithm can reduce complexity and decoding latency at the same time without performance loss. Then, based on the proposed algorithm, the corresponding VLSI architecture is developed. Hardware analysis shows that the proposed SCL decoders have significant reduction in area and decoding latency.

Chapter 5

5 BP DECODER

In this chapter, we present the high-performance BP decoder for polar codes. Section 5.2 reviews the original BP decoding algorithm. In Section 5.3, various optimizing techniques for BP decoder design are presented. Section 5.4 proposes several early stopping criteria for energy-efficient low-latency BP decoders. Bases on the proposed techniques in Section 5.3 and Section 5.4, the hardware architecture of proposed BP decoders are developed in Section 5.5. Section 5.6 analyzes the hardware performance of the example (1024, 512) BP decoders.

5.1 Introduction

As another decoding approach for polar codes, belief propagation (BP) algorithm does not receive as much attention as SC algorithm. To data, only a few works [12-14] [54] were reported on the theoretical analysis and hardware design of BP decoders. This phenomenon is mainly due to the fact that BP algorithm has much higher computation complexity than its SC counterpart. In particular, compared to SC algorithm that only needs one-directional message propagation, the BP algorithm requires bi-directional message propagation. As a result, the computation and memory resource for polar BP decoding is very huge.

On the other hand, as an iterative decoding approach, BP algorithm has inherent advantage on parallel processing. Recall the drawback of long latency problem for SC algorithm, BP algorithm is much more attractive for high-throughput low-latency applications. As a result, how to enhance the advantage of BP algorithm on timing, as well as alleviate its weakness on complexity, is very important for practical BP decoder design.

This chapter systemically investigates the optimization of BP decoders. First, a scaled min-sum

approach is applied to improve the error-correcting performance of BP algorithm. Then several hardware-level optimization approaches are proposed to reduce the latency and area of BP decoder. Furthermore, several early stopping-criteria are presented to reduce the energy dissipation and decoding latency linearly. Synthesis results show that the proposed BP decoder for (1024, 512) polar codes achieve significant reduction in energy consumption and decoding latency as compared to the prior works.

5.2 BP Algorithm

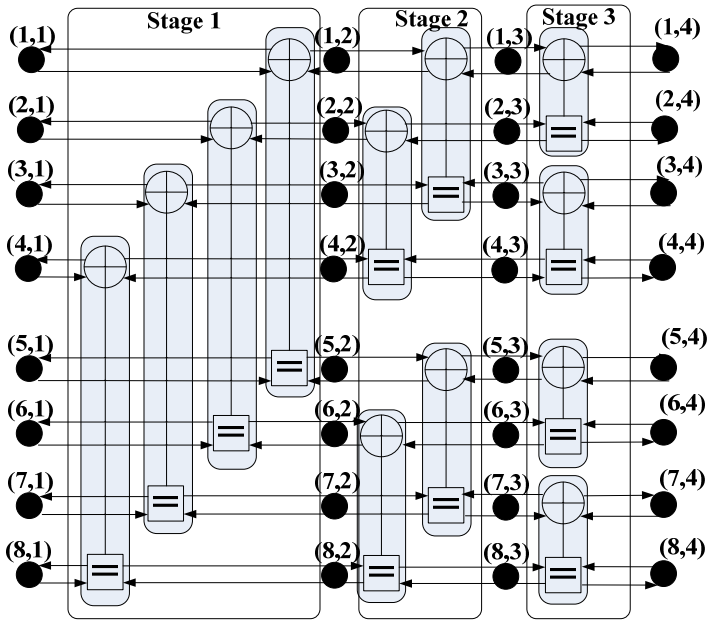
Derived from factor graph theory [33], the belief propagation (BP) algorithm can be applied for polar decoding [13]. Generally, an (n, k) polar code (n=2^m) can be iteratively decoded via an m-stage factor graph network consisting of (m+1)n nodes. Each node (i, j) is associated with two types of likelihood message: left-to-right and right-to-left. In BP decoding procedure, these messages are propagated and updated between adjacent nodes.

Fig. 5.1(a) shows an example BP factor graph for the case of (8, 4) polar code. Here the graph network has a total of m=$\log_2 8$=3 stages. Each stage consists of n/2=4 processing elements (PEs) (see Fig. 5.1(b)), which are used for updating the propagating messages. To avoid overflow, these updates are always performed in logarithmic domain. Therefore, the propagating messages are based on logarithmic likelihood ratio (LLR) form, and are updated using (5.1). Here $L_{i,j}^{t}$ and $R_{i,j}^{t}$ represent left and right propagating messages, and t is the current iteration index. Notice that at each iteration t=0,1,2,3…, depending on whether i is a frozen position or not, $R_{i,0}^{t}$ will be set either as a large constant or 0, respectively.

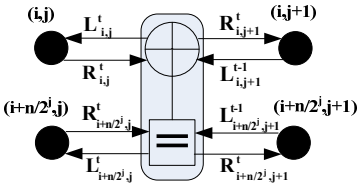
$$
\begin{aligned}
L_{i,j}^{t} &= sign(L_{i,j+1}^{t-1})sign(L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{j},j}^{t})\min(\left|L_{i,j+1}^{t-1}\right|,\left|L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{j},j}^{t}\right|) \\
L_{i+n/2^{j},j}^{t} &= L_{i+n/2^{j},j+1}^{t-1} + sign(L_{i,j+1}^{t-1})sign(R_{i,j}^{t})\min(\left|L_{i,j+1}^{t-1}\right|,\left|R_{i,j}^{t}\right|) \\
R_{i,j+1}^{t} &= sign(R_{i,j}^{t})sign(L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{j},j}^{t})\min(\left|R_{i,j}^{t}\right|,\left|L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{i},j}^{t}\right|) \\
R_{i+n/2^{j},j+1}^{t} &= R_{i+n/2^{j},j}^{t} + sign(L_{i,j+1}^{t-1})sign(R_{i,j}^{t})\min(\left|L_{i,j+1}^{t-1}\right|,\left|R_{i,j}^{t}\right|).
\end{aligned}
\tag{5.1}
$$

Based on (5.1), the likelihood messages can be propagated and updated iteratively in the factor graph. After the decoder reaches maximum iteration number (*max_iter*), node (*i*, 1) will output the decoded bits based on hard decision of messages.

It should be noted that (5.1) represents the min-sum approximation of the BP algorithm. Compared with the original BP algorithm, this approximated version is more suitable for hardware implementation [54]. However, its error-correcting performance is degraded due to the approximation. In next section, this problem is addressed further.



(a)



(b)

Fig. 5.1. Structure of BP decoding
(a) Factor graph of (8, 4) polar code. (b) Diagram of PE.

5.3 Optimized BP Decoder

5.3.1 Scaled Min-Sum Technique

As mentioned in Section 5.2, the min-sum algorithm described by (5.1) has some inherent performance disadvantages due to the approximation (see Fig. 5.2). In order to avoid performance loss, similar to the approach used in LDPC decoding [55], we propose to introduce a scaling parameter *s* to offset the approximation error: For each time of min-sum operation, the output will be scaled by *s*. Accordingly, the original non-scaled MS algorithm described by (5.1) is modified to a scaled min-sum (SMS) version described by (5.2).

$$\begin{aligned}
L_{i,j}^{t} &= s * sign(L_{i,j+1}^{t-1}) sign(L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{j},j}^{t}) \min(\left|L_{i,j+1}^{t-1}\right|, \left|L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{j},j}^{t}\right|) \\
L_{i+n/2^{j},j}^{t} &= L_{i+n/2^{j},j+1}^{t-1} + s * sign(L_{i,j+1}^{t-1}) sign(R_{i,j}^{t}) \min(\left|L_{i,j+1}^{t-1}\right|, \left|R_{i,j}^{t}\right|) \\
R_{i,j+1}^{t} &= s * sign(R_{i,j}^{t}) sign(L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{j},j}^{t}) \min(\left|R_{i,j}^{t}\right|, \left|L_{i+n/2^{j},j+1}^{t-1} + R_{i+n/2^{i},j}^{t}\right|) \\
R_{i+n/2^{j},j+1}^{t} &= R_{i+n/2^{j},j}^{t} + s * sign(L_{i,j+1}^{t-1}) sign(R_{i,j}^{t}) \min(\left|L_{i,j+1}^{t-1}\right|, \left|R_{i,j}^{t}\right|).
\end{aligned} \tag{5.2}$$

As shown in Fig. 5.2, the introduction of scaling parameter helps improve the decoding performance greatly. For the example (1024, 512) polar code with *max_iter*=60, the proposed SMS algorithm with *s*=0.9375 can obtain an extra 0.5 dB decoding gain over its non-scaled counterpart. In that case, the SMS algorithm can achieve a similar error-correcting performance with the original BP and SC algorithms. Notice that since $s=0.9375=1-2^{-4}$, the scaling operation can be implemented with a simple shift-addition circuit.

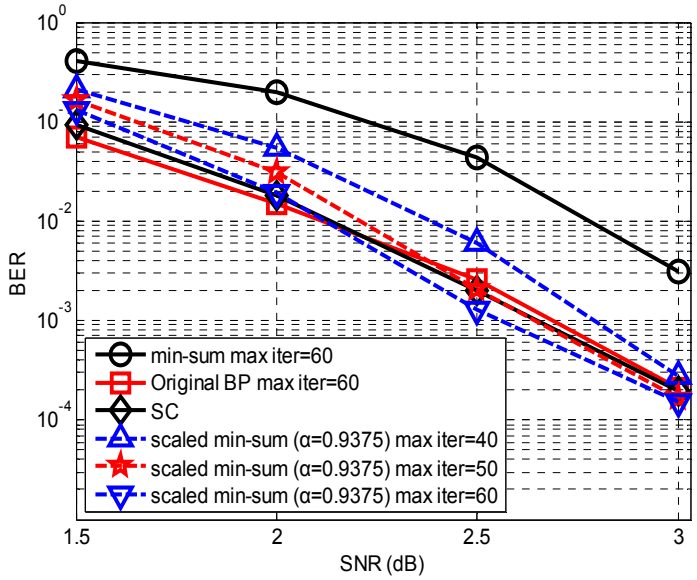


Fig. 5.2. Performance of BP decoding for (1024, 512) polar codes.

5.3.2 Cross-level Overlapping

Low hardware utilization of original polar BP decoder

As discussed in Section 5.2, the BP algorithm for (n, k) polar code is processed over m-stage factor graph (see Fig. 5.1(a)). Therefore, a systolic (n, k) polar BP decoder can be developed in a straightforward manner based on the corresponding factor graph. Fig. 5.3 shows the overall architecture of such systolic polar BP decoder presented. Here each stage contains $n/2$ PEs to update LLR messages. Between adjacent stages register-based pipelines are inserted to store propagating messages.

The problem of the systolic architecture in Fig. 5.3 is its low hardware utilization. According to the decoding procedure of BP algorithm, PEs are activated stage-by-stage from left to right in each iteration. Fig. 5.4 shows an example of this decoding scheme of (16, 8) polar code for 3 iterations. Here $C_{i,j}^{p}$ indicates that, the propagating messages that belong to the i-th received codeword are updated in the j-th stage of PEs during the p-th iteration. For example, $C_{1,3}^{2}$ in clock

cycle-7 represents that, in order to decode the 1st received codeword, the stage3 is activated and processes the propagating messages in the 2nd iteration. In addition, the arrow in Fig. 5.4 describes the data dependency in (5.2). Here the black arrow indicates the dependency within the same iteration, and the red arrow indicates the dependency between consecutive iterations. For example, only after stage 1 and stage 3 finish processing $C_{1,1}^{2}$ and $C_{1,3}^{1}$, respectively, their output is sent to stage 2 as its inputs, and then stage 2 is allowed to process $C_{1,2}^{2}$. From Fig. 5.4 it can be seen that in each cycle, only one stage is activated while other stages are always idle. For a (n, k) polar BP decoder, this yields a low hardware utilization rate of only 1/m.

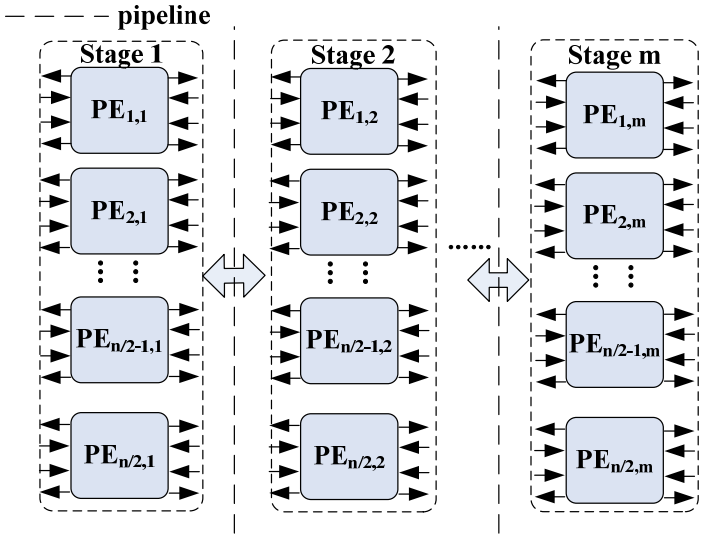


Fig. 5.3. Architecture of systolic polar BP decoder.

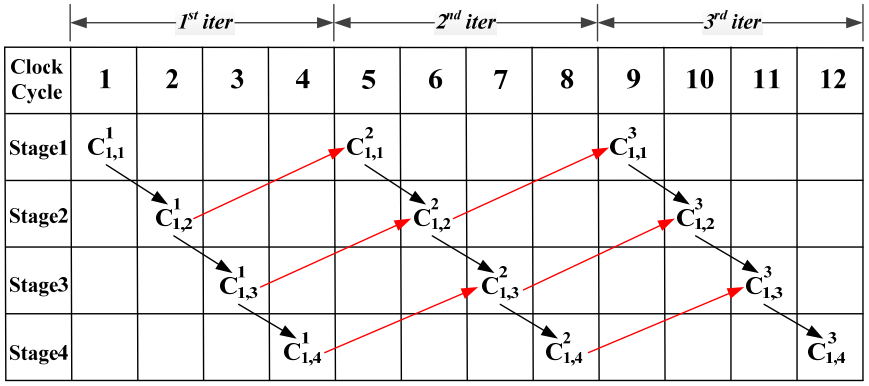


Fig. 5.4. Original decoding scheme of systlic n=16 BP decoder.

Overlapping at iteration level

Fig. 5.4 shows that *m*-1 stages of PEs are idle in each cycle of polar BP decoding. In order to avoid this under-utilization, overlapped scheduling technique [43] can be used to fully utilize those idle resource. In this section, we discuss the iteration-level overlapping first.

After a careful examination of Fig. 5.4, we find that in each cycle multiple stages can be activated at the same time. For example, consider $C_{1,1}^{2}$ in cycle-5 at stage 1. This computation is dependent on output from $C_{1,2}^{1}$ in cycle-2 at stage 2. Since stage 2 can process $C_{1,2}^{1}$ in cycle-2, stage 1 can process $C_{1,1}^{2}$ at the beginning of cycle-3. Therefore, instead of being activated in cycle-5 in the original scheme (see Fig.5.4), stage1 can now be enabled in cycle-3 to process propagating messages for the 2nd iteration ($C_{1,1}^{2}$) without any timing conflict (see Fig. 5.5). As a result, one clock cycle can be saved by applying this re-scheduling approach. Similarly, Fig. 5.5 shows that the other stages can also be activated earlier. Therefore, this re-scheduling approach can lead to reduction of 4 cycles in latency.

By exploiting overlapped-scheduling, from Fig. 5.5, it can be seen that, from cycle-3 to cycle-6, some computations belong to either 2nd or 3rd iteration. This overlapped-rescheduling approach leads to great reduction in decoding latency. In general, for an (*n*, *k*) polar BP decoder with m-stages of PEs, the proposed iteration-level overlapped scheduling approach reduces the decoding latency from *m***max_iter* to 2**max_iter*+*m*-2 clock cycles, where *max_iter* is the pre-set maximum number of iterations. Considering *m* is usually larger than 10 for practical use, the latency can be reduced by approximately 10/2=5 times.

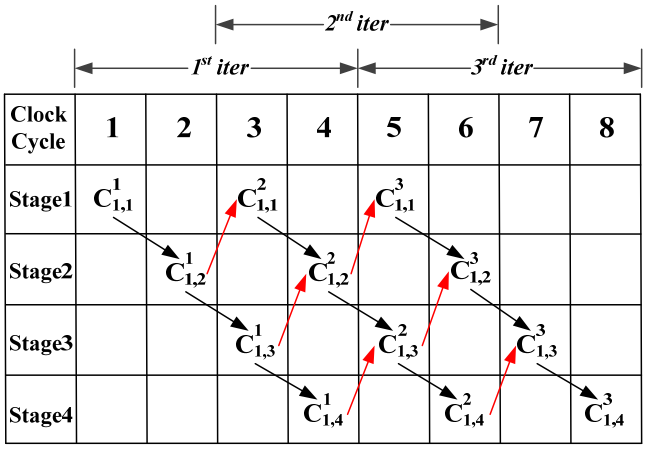


Fig. 5.5. Overlapped-scheduling at iteration level.

Overlapping at codeword level

Although the above iteration-level overlapped scheduling method can greatly improve the hardware utilization, the schedule in Fig. 5.5 cannot achieve 100% hardware utilization since some stages still remain idle. In general, due to the data dependency between successive iterations, the maximum hardware utilization rate of iteration-level overlapped scheme is limited to be less than 50%.

Different from iteration-level overlapping, codeword-level overlapping, as a common technique used in SC decoder designs, can make hardware utilization close to 100%. In this section, we apply this technique for BP decoder optimization. Fig. 5.6 shows an example of a 4-level-codeword-overlapping scheme. Here different colors represent propagating messages belonging to different received codewords. Compared with original scheme with iteration-level overlapping (see Fig. 5.4 and Fig. 5.5), the codeword-level overlapping scheme fully utilizes those idle stages to process multiple independent received codewords. As a result, the hardware utilization rate approaches 100%. In general, for an (n, k) polar BP decoder, maximum m independent received codewords can be input to the decoder for overlapped processing. As a result, the processing throughput increases by approximately m times at the expense of an extra m-1 clock cycles, and (m-1) times more memory for storing the codewords than that in Fig. 5.4

Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Stage1	$C^1_{1,1}$	$C^1_{2,1}$	$C^1_{3,1}$	$C^1_{4,1}$	$C^2_{1,1}$	$C^2_{2,1}$	$C^2_{3,1}$	$C^2_{4,1}$	$C^3_{1,1}$	$C^3_{2,1}$	$C^3_{3,1}$	$C^3_{4,1}$			
Stage2		$C^1_{1,2}$	$C^1_{2,2}$	$C^1_{3,2}$	$C^1_{4,2}$	$C^2_{1,2}$	$C^2_{2,2}$	$C^2_{3,2}$	$C^2_{4,2}$	$C^3_{1,2}$	$C^3_{2,2}$	$C^3_{3,2}$	$C^3_{4,2}$		
Stage3			$C^1_{1,3}$	$C^1_{2,3}$	$C^1_{3,3}$	$C^1_{4,3}$	$C^2_{1,3}$	$C^2_{2,3}$	$C^2_{3,3}$	$C^2_{4,3}$	$C^3_{1,3}$	$C^3_{2,3}$	$C^3_{3,3}$	$C^3_{4,3}$	
Stage4				$C^1_{1,4}$	$C^1_{2,4}$	$C^1_{3,4}$	$C^1_{4,4}$	$C^2_{1,4}$	$C^2_{2,4}$	$C^2_{3,4}$	$C^2_{4,4}$	$C^3_{1,4}$	$C^3_{2,4}$	$C^3_{3,4}$	$C^3_{4,4}$

Fig. 5.6. 4-level-overlapping at codeword level.

Joint overlapping at both codeword and iteration level

The above two subsections illustrated two types of overlapping methods at iteration level and codeword level, respectively. Next we show that these methods can be unified in a general framework. Recall that in Fig. 5.4 the processed propagating messages are denoted as $C^p_{i,j}$, where i and p represent the indices of the codeword and iteration, respectively. After observing Fig. 5.5 and Fig. 5.6 we can find that these two overlapping strategies are only different with respect to the choice of the overlap variable. In Fig. 5.5, p, as the index of iteration, is used for overlap (for example $C^2_{1,1}$ and $C^1_{1,3}$ in cycle-3), while in Fig. 5.6, i, the index of codeword is used for overlap (for example $C^1_{3,1}$, $C^1_{2,2}$ and $C^1_{1,3}$ in cycle-3). Therefore, Fig. 5.5 and Fig 5.6 can be viewed as two special types of overlapping strategies for $C^p_{i,j}$. Furthermore, we can develop other types of overlapping strategies where both i and p are used for overlap. Fig. 5.7 shows an example of such a joint overlapping method. From this table it can be seen that from cycle-3 to cycle-7 both iterations and codewords overlap. As a result, the hardware utilization rate increases to 100% at the expense of 100% increase in memory.

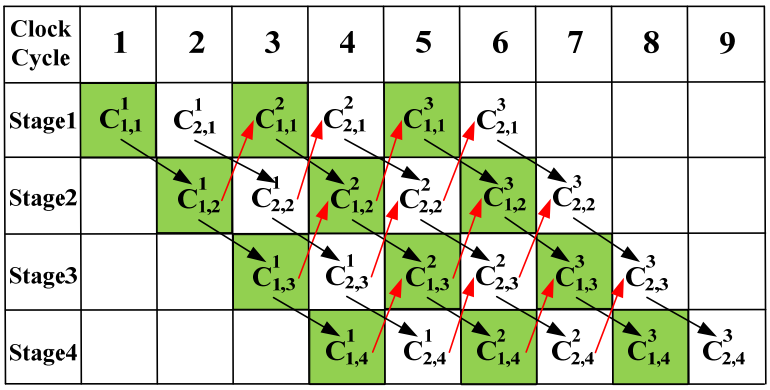


Fig. 5.7. Joint overlapping at both iteration and codeword level.

5.3.3 Folded Architecture

Section 5.3.2 presented two methods that can increase hardware utilization of polar BP decoders to 100%. However, both these methods need linear increase of memory. In this subsection, we propose to use folding technique to improve hardware utilization without increase of hardware.

Recall that in Fig. 5.4 m-1 stages remain idle in each cycle. Since the inner architecture of PEs in different stages is exactly the same, we can use folding technique [56] with fewer stages of PEs. Fig. 5.8 illustrates a folded decoding scheme with folding factor as 4. In cycles-2,6,10, the propagating messages that were previously processed by stage 2, are now routed to be processed by stage 1. Similarly, in cycles-3,7,11 and cycles-4,8,12, the propagating messages that were processed by stage 3 and stage 4 are now processed by stage 1 as well. As a result, the hardware utilization is 100% and the hardware complexity is reduced by a factor of 4. However, the penalty of this method is the extra routing network for routing the messages to the corresponding PEs.

Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12
Stage1	$C_{1,1}^{1}$	$C_{1,2}^{1}$	$C_{1,3}^{1}$	$C_{1,4}^{1}$	$C_{1,1}^{2}$	$C_{1,2}^{2}$	$C_{1,3}^{2}$	$C_{1,4}^{2}$	$C_{1,1}^{3}$	$C_{1,2}^{3}$	$C_{1,3}^{3}$	$C_{1,4}^{3}$

Fig. 5.8. 4-level folded decoding scheme.

The above folding technique can also be incorporated with iteration-level overlapping to simultaneously increase hardware utilization and reduce latency. Fig. 5.9 shows joint use of iteration-level overlapping and folding for an n=16 polar code. Compared with the original 4-stage-scheme of Fig. 5.5, it can be seen that only 2 stages are needed in Fig. 5.9 to process the same codeword with the help of routing network. In addition, the overall latency remains the same with that in Fig. 5.5. In general, for the original m-stage iteration-overlapping BP decoding scheme, we can develop a class of folded architectures based on different folding factors f, where $f \leq m$. For example, for the 6-stage iteration-overlapping BP decoding scheme with $n=2^6=64$, we can fold it as either 3-stage (f=2) architecture (see Fig. 9) or 2-stage (f=3) architecture (see Fig. 11). In general, smaller f can lead to less latency (f=2 in Fig. 10), and larger f can lead to fewer number of stages of PEs (f=3 in Fig. 11).

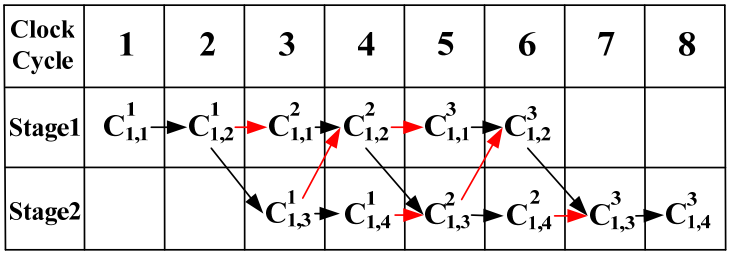


Fig. 5.9. Joint folded and iteration-level overlapping scheme.

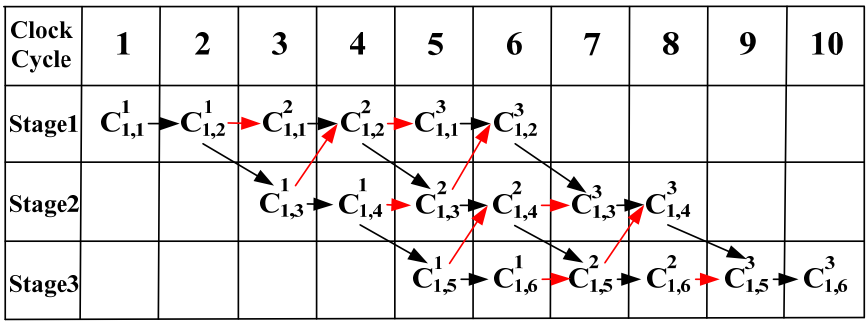


Fig. 5.10. 2-level folded iteration-level overlapping with n=64.

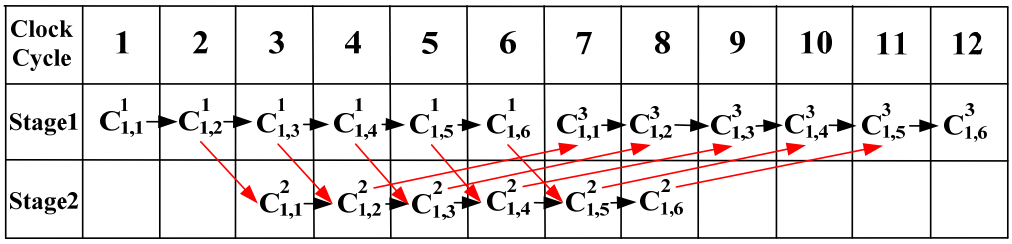


Fig. 5.11. 3-level folded iteration-level overlapping with n=64.

5.4 Early-Stopping Criteria

Early stopping criteria have been extensively explored in many prior iterative decoders, such as LDPC and Turbo decoding [57-62]. Based on the different targeted SNR regions, early stopping criteria can be grouped into different types. Among them the most important type is the *detection-type*, which is used for detecting whether the valid output *has been* already decoded or not. If so, the decoder will stop at an early iteration due to decoding success; this is illustrated in Fig. 5.12. This detection-type stopping criterion is very useful in high SNR regions, since valid output can be usually obtained after few iterations. Notice that the well-known ***H***-matrix-based method [63], which uses parity ***H*** matrix to check the output of decoder is a valid codeword or not, can be viewed as a detection-type stopping criterion for block codes (such as LDPC codes).

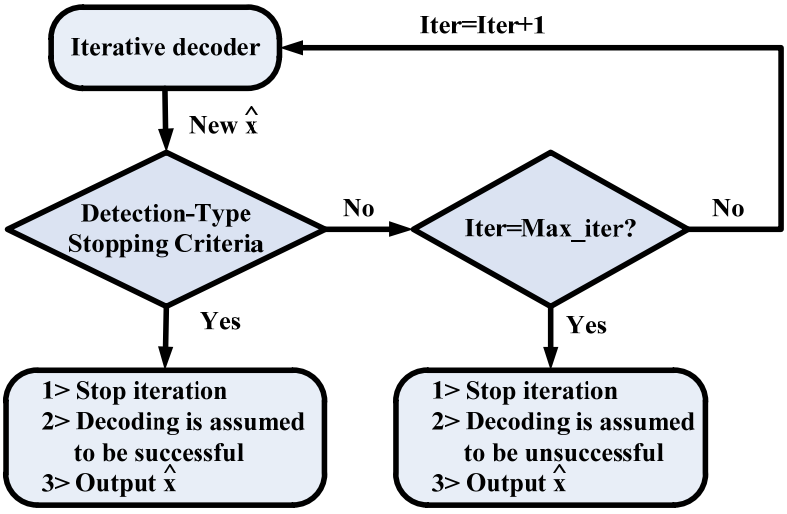


Fig. 5.12. Iterative decoder with detection-type stopping criteria.

5.4.1 Inability of H-matrix-based approach for polar codes

For most block codes (such as LDPC codes), $\boldsymbol{H}$ matrix is commonly used for codeword detection. According to coding theory [63], given generator matrix $\boldsymbol{G}$ and parity matrix $\boldsymbol{H}$, $\boldsymbol{xH^T=0}$ always holds for any codeword $\boldsymbol{x}$. Therefore, if the output of the decoder, referred as $\hat{\boldsymbol{x}}$ (the estimate of $\boldsymbol{x}$), satisfies $\hat{\boldsymbol{x}}\boldsymbol{H}^T=\boldsymbol{0}$, then $\hat{\boldsymbol{x}}$ is also the codeword generated by $\boldsymbol{G}$. In that case, $\hat{\boldsymbol{x}}$ is equivalent to $\boldsymbol{x}$ with high probability. Therefore, the decoder can immediately terminate the iterations and output $\hat{\boldsymbol{x}}$ as the valid estimate of $\boldsymbol{x}$ (see Fig. 5.13(a)). In general, the above $\boldsymbol{H}$-matrix-based approach has very high detection accuracy with very small hardware overhead.

As a type of block code, polar codes have the similar property of $\boldsymbol{xH^T=0}$ [29]. However, the $\boldsymbol{H}$ matrix of polar codes cannot be used for codeword detection. This is because the output after each iteration of polar decoder is always $\hat{\boldsymbol{u}}$ (the estimate of $\boldsymbol{u}$) instead of $\hat{\boldsymbol{x}}$ (see Fig. 5.13(b)). Next we prove that $\hat{\boldsymbol{u}}\boldsymbol{H}^T$ is generally not equivalent to 0 even for decodable cases.

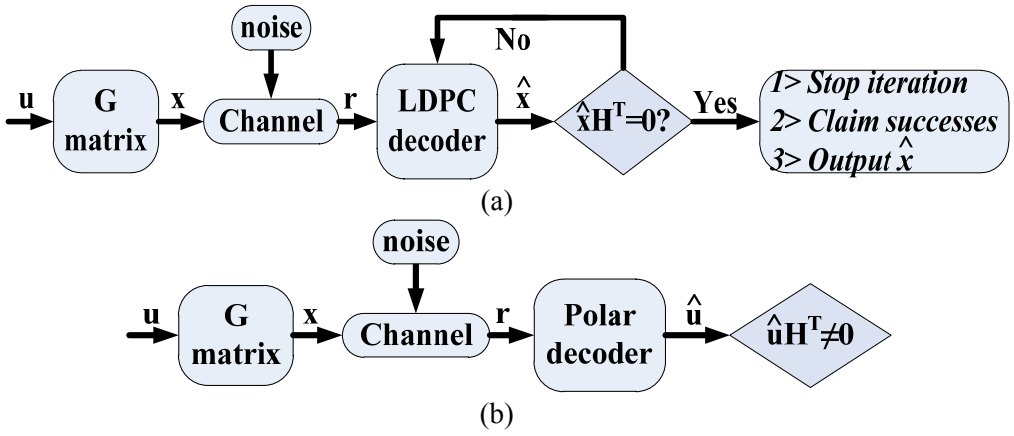


Fig. 5.13. $\boldsymbol{H}$-matrix –based detection
(a) $\boldsymbol{H}$-matrix can detect valid output of LDPC decoder. (b) $\boldsymbol{H}$-matrix cannot detect valid output of polar decoder.

Assume $\hat{\boldsymbol{x}}$ is the valid codeword $\boldsymbol{x}$, then $\hat{\boldsymbol{x}}\boldsymbol{H}^T=0$. Because $\hat{\boldsymbol{u}}=\hat{\boldsymbol{x}}\boldsymbol{G}$, we then have $\hat{\boldsymbol{u}}\boldsymbol{H}^T=\hat{\boldsymbol{x}}\boldsymbol{G}\boldsymbol{H}^T=\hat{\boldsymbol{x}}(\boldsymbol{G}\boldsymbol{H}^T)$. Consider in general $\boldsymbol{G}\boldsymbol{H}^T\neq\boldsymbol{H}^T$, then $\hat{\boldsymbol{u}}\boldsymbol{H}^T=\hat{\boldsymbol{x}}(\boldsymbol{G}\boldsymbol{H}^T)\neq\hat{\boldsymbol{x}}\boldsymbol{H}^T$. Hence $\hat{\boldsymbol{u}}\boldsymbol{H}^T\neq 0$ even if $\hat{\boldsymbol{x}}=\boldsymbol{x}$. As a result, $\boldsymbol{H}$ matrix cannot be used to examine the validity of the output

of polar decoder.

5.4.2 G-matrix-based Stopping Criterion for Detection

Fig. 5.13 shows that the absence of $\hat{\boldsymbol{x}}$ causes ***H***-matrix-based approach to fail in detecting a valid output for polar decoders. In this section, we propose a novel ***G***-matrix-based approach to solve this problem. By utilizing $\hat{\boldsymbol{u}}$ and ***G*** matrix, this ***G***-matrix-based approach achieves good performance in detecting a valid output. As a result, this method can be used as an efficient detection-type early stopping criterion for polar codes.

As introduced in Section 5.2, the BP algorithm is performed over the factor graph of polar codes. Due to the special auto-duality property of the generator matrix $\boldsymbol{G} = \boldsymbol{F}^{\otimes m}$, the factor graph of BP decoder is just the edge-to-edge estimation of the encoder (***G*** matrix). As a result, the node $q_{i,j}$ in factor graph (see Fig. 5.1(a)) is the estimation of corresponding bit in the encoder, where $q_{i,j} \triangleq \mathbf{sign}(L_{i,j}^t + R_{i,j}^t)$. Consider $b_{i,1}=u_i$ and $b_{i,m+1}=x_i$, hence for decodable cases, after certain rounds of iterations, $q_{i,1}$ and $q_{i,m+1}$ are very likely equal to u_i and x_i, respectively. As a result, the two vectors which consist of $q_{i,1}$ and $q_{i,m+1}$ can be viewed as the estimates of ***u*** and ***x***, referred as $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{x}}$, respectively. Here $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{x}}$ are defined as:

$$\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, ..., \hat{u}_n) \triangleq (q_{1,1}, q_{2,1}, ..., q_{n,1}) \text{ and } \hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \triangleq (q_{1,m+1}, q_{2,m+1}, ..., q_{n,m+1}). \quad (5.3)$$

Recall that for any polar codes $\boldsymbol{x}=\boldsymbol{u}\boldsymbol{G}$, hence if $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{x}}$ are the valid estimates, $\hat{\boldsymbol{x}} = \hat{\boldsymbol{u}}\boldsymbol{G}$ must also hold. Therefore, $\hat{\boldsymbol{x}} = \hat{\boldsymbol{u}}\boldsymbol{G}$ can be used to detect valid $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{x}}$ as follows:

G-matrix-based stopping criterion for detecting valid output (G-matrix criterion): *If* $\hat{\boldsymbol{x}} = \hat{\boldsymbol{u}}\boldsymbol{G}$, *then the* $\hat{\boldsymbol{u}}$ *is assumed as a valid estimate of* ***u***. *The decoder will output* $\hat{\boldsymbol{u}}$ *and stop further iterations.*

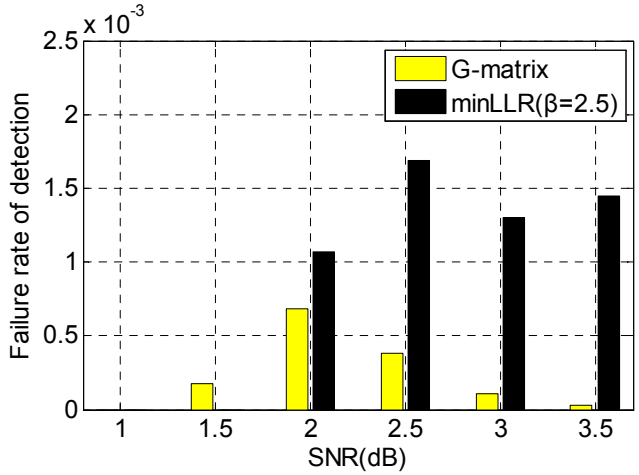


Fig. 5.14. Failure rate of stopping criteria with *max_iter*=40.

Accordingly, a polar BP decoder with the above ***G-matrix*** early stopping criterion is developed as shown in Scheme 5.1.

Scheme 5.1 (n, k) Polar BP decoder with G-matrix criterion:

1: Input:

2: **Log-Likelihood ratios** $LLR(r_i)$ **from channel output**

3: **Frozen positions set:** **frz** $= \{frz_1, frz_2, ...frz_{n-k}\}$

4: Initialization:

5: **Set scale value** α, **maximum iteration number** $\max\_iter$

6: **For the propagating messages** $L_{i,j}^t$ **and** $R_{i,j}^t$ **of each node**(i, j)**:**

7: **if** $(j == 1)$ **&** $(i \in$ **frz**$)$ $R_{i,1}^t = \infty$ **for** $t = 0,1...\max\_iter$

8: **else if** $(j == 1 + m)$ $L_{i,m+1}^t = LLR(r_i)$ **for** $t = 0,1...\max\_iter$

9: **else** $L_{i,j}^0 = R_{i,j}^0 = 0$

10: Iteration process:

11: **While** $t < \max\_iter$ **do**

12: **Update** $L_{i,j}^t$ **and** $R_{i,j}^t$ **for each node based on equations (3)**

13: **Update** $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2 ..., \hat{u}_n)$ **and** $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2 ..., \hat{x}_n)$**:**

14: **if** $(L_{i,m+1}^t + R_{i,m+1}^t \geq 0)$ $\hat{x}_i = 0$ **else** $\hat{x}_i = 1$

15: **if** $(L_{i,1}^t + R_{i,1}^t \geq 0)$ $\hat{u}_i = 0$ **else** $\hat{u}_i = 1$

16: **G-matrix-I early stopping criterion for decodable cases:**

17: **if** $(\hat{\mathbf{u}}\mathbf{G} == \hat{\mathbf{x}})$ **1> Decoding is assumed to be successful**

18: **2> Stop iteration & Output** $\hat{\mathbf{u}}$

19: **else** $t = t + 1$ **& Begin next iteration**

20: Output : $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2 ..., \hat{u}_n)$

Fig. 5.14 shows the failure rate of the proposed ***G-matrix*** stopping criterion for polar (1024, 512) code. Here the failure is counted when the stopping criterion causes an invalid decoded output. From this figure it can be seen that the proposed method shows very low failure rate (<0.1%) at both low and high SNR regions. As a result, compared to the BP decoder with constant number of iterations, the BP decoder with the ***G-matrix*** stopping criterion can achieve the same decoding performance (see Fig. 5.15) with less number of iterations (see Fig. 5.16). In particular, as seen in Fig. 5.16, in high SNR region the benefit of this approach on reducing the number of iterations is very significant. For example, when SNR is 3.5dB, the average number of required iterations can be reduced by 42.5%.

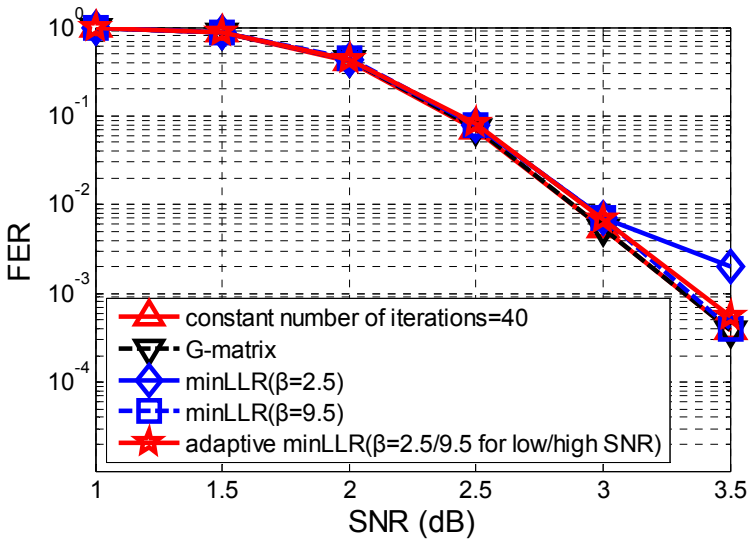


Fig. 5.15. Performance of polar BP decoding with stopping criteria.

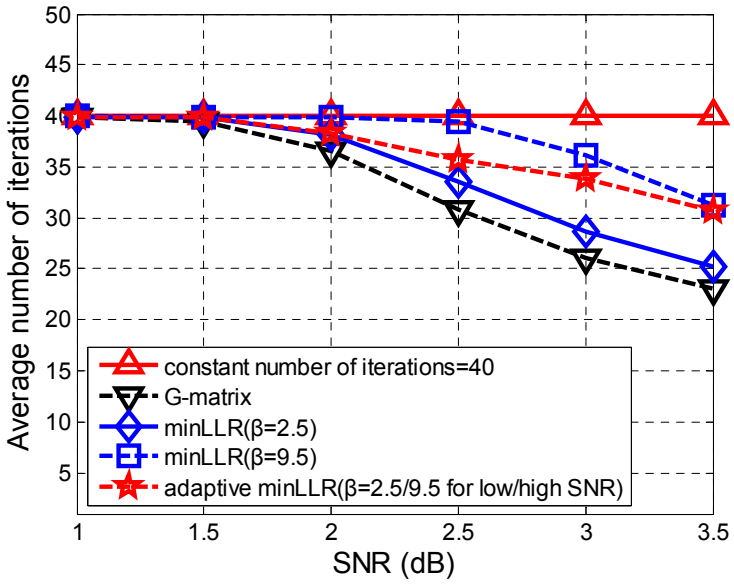


Fig. 5.16. Average number of iterations with stopping criteria.

5.4.3 minLLR-based Stopping Criterion for Detection

Section 5.4.2 introduced a ***G***-matrix-based early stopping criterion, which utilizes $q_{i,1}$ and $q_{i,m+1}$, as the hard-decision values of $LLR_{i,1}^t$ and $LLR_{i,m+1}^t$, to detect the valid $\hat{\boldsymbol{u}}$. In this section, another efficient detection-type stopping criterion, which only uses the information of $LLR_{i,1}^t$ is presented. Recall that $\hat{u}_i$ is determined as $\hat{u}_i = \mathbf{sign}(LLR_{i,1}^t)$. This hard-decision procedure only utilizes the information in sign part of $LLR_{i,1}^t$, while the information in magnitude part, denoted as $\left|LLR_{i,1}^t\right|$, is not used at all. However, since $LLR_{i,1}^t$ is related to the probability of $\hat{u}_i$ being 0 or 1 (where $LLR_{i,1}^t = \ln(\Pr(\hat{u}_i = 0)/\Pr(\hat{u}_i = 1))$), intuitively the larger $\left|LLR_{i,1}^t\right|$ means the corresponding hard-decision value is more reliable. As a result, $\left|LLR_{i,1}^t\right|$ can be used to measure the reliability of $\hat{u}_i$. Therefore, we propose to use the minimum $\left|LLR_{i,1}^t\right|$ (*minLLR*) for all i=1,2,…n to detect the valid $\hat{\boldsymbol{u}}$:

minLLR-based stopping criterion for detecting valid output (minLLR criterion): *If minLLR is larger than threshold value β, then the corresponding* $\hat{\boldsymbol{u}}$ *can be assumed as a valid estimate of* $\boldsymbol{u}$*. The decoder will output* $\hat{\boldsymbol{u}}$ *and stop further iterations.*

This criterion can be understood in an intuitive way. When *minLLR* is larger than β, that means all $\left|LLR_{i,1}^{t}\right|$ values are larger than β. Considering β is typically larger than 2.5, it indicates that the probability of each corresponding hard-decision $\hat{u}_i$ being 0 or 1 is at least $e^{2.5}\approx 12$ times larger than that of it being 1 or 0. In that case, all the decoded $\hat{u}_i$ are highly reliable. Hence, $\hat{\boldsymbol{u}}$ is most likely a valid estimate of $\boldsymbol{u}$.

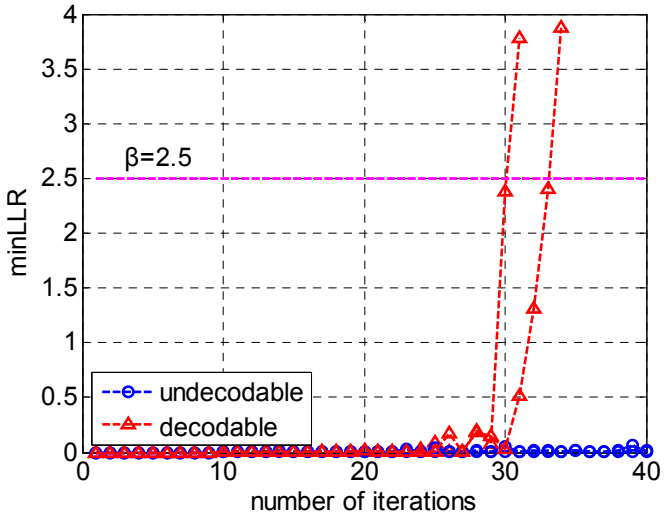


Fig. 5.17. *minLLR* trend with polar BP decoding at SNR=2.5dB.

The validity of the proposed approach is illustrated in Fig. 5.17. Fig. 5.17 shows the *minLLR*-vs-number of iterations curve for two undecodable and two decodable cases with SNR=2.5dB. Here polar (1024, 512) scaled min-sum decoder with α=0.9375 is used. From this figure, it can be seen that, for undecodable cases *minLLR* is always smaller than the threshold value β=2.5 during the whole iteration process. On the other hand, for decodable cases *minLLR* always exceeds β at an earlier time.

Fig. 5.18 shows the cumulative distribution function (CDF) of successful detection for decodable cases when using *minLLR* criterion. With β=2.5, the (1024, 512) BP decoder can always decode the valid codewords earlier than reaching the pre-set maximum iteration number. From this figure it is also found that with higher SNR value the probability that finding valid codewords with less number of iterations become higher.

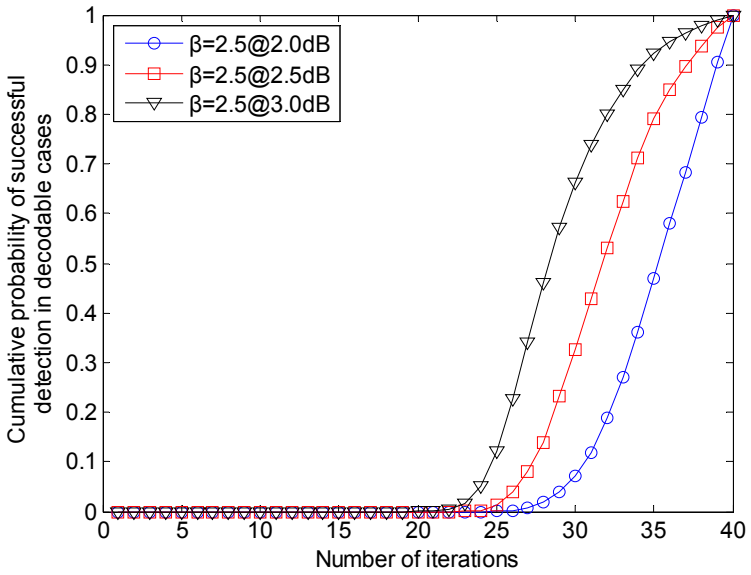


Fig. 5.18. CDF of successful detection for decodable cases with β=2.5.

Accordingly, a polar BP decoder with the *minLLR*-based early stopping criterion can be summarized in Scheme 5.2.

The failure rate of *minLLR* criterion for polar (1024, 512) codes is shown in Fig. 5.14. From this figure it can be seen that, for low and medium SNR scenarios that allow higher FER rate (10^{-1}~10^{-2}), the failure rate of *minLLR* criterion is low enough to guarantee decoding performance. However, for high SNR regions that require lower FER rate (such as 10^{-3} for SNR=3.5dB), the failure rate is relatively large to cause performance loss (see Fig. 5.15). In that case, a larger β is needed to avoid performance degradation. For example, simulation results show that the choice of β=9.5 can totally avoid the performance loss for polar (1024, 512) code (see Fig. 5.15).

Scheme 5.2 (n, k) Polar BP decoder with minLLR criterion:

1: Input: Same with Scheme-A, threshold parameter β

2: Initialization: Same with Scheme-A

3: Iteration process:

4: While $t < \max\_iter$ **do**

5: Update $L_{i,j}^t$ **and** $R_{i,j}^t$ **for each node based on equations (3)**

6: Update $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2 ..., \hat{u}_n)$ **: if** $(L_{i,1}^t + R_{i,1}^t \geq 0)$ $\hat{u}_i = 0$ **else** $\hat{u}_i = 1$

7: Find minimum magnitude LLR ($\min LLR$**) among all** $\left| L_{i,1}^t + R_{i,1}^t \right|$

8: minLLR early stopping criterion for decodable cases

9: if $(\min LLR \geq \beta)$ **1> Decoding is assumed to be successful**

10: 2> Output $\hat{\mathbf{u}}$ **& Stop iteration**

11: else $t = t + 1$ **& Begin next iteration**

12: Output : $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2 ..., \hat{u}_n)$

The decoding performance and average number of iterations for (1024, 512) BP decoder with the *minLLR* stopping criterion are show in Fig. 5.15 and Fig. 5.16, respectively. As seen from these two figures, after using *minLLR* criterion the required number of iterations for the BP decoder at low and medium SNR regions can be reduced by 17.5%~37.5% without performance loss. For the performance loss at high SNR region, since it is caused by the relatively high failure rate with smaller β, the use of larger β (for example β=9.5) can avoid this performance degradation in high SNR regions (see Fig. 5.15). In that case, the average number of iterations at 3.5dB can be reduced by 32.5%. Notice that compared to using a smaller β, the use of larger β leads to a relatively larger number of iterations (see Fig. 5.16). This is the penalty for avoiding performance loss.

5.4.4 Adaptive Stopping Criterion

The proposed channel condition estimation method

For practical applications, the SNR information from channel output is very useful for efficient polar decoding. Recall that in Section 5.4.4, the required value of threshold in the high SNR regions (for example β=9.5 at SNR=3.5dB) is always larger than that in low SNR regions (β=2.5

when SNR≤3dB). This difference of β in different SNR scenarios leads to the dilemma of choosing β. If smaller β is selected, decoding performance in high SNR scenario cannot be guaranteed (see Fig. 5.15); however, if a larger β is chosen, it leads to extra iterations in low SNR cases since smaller β is sufficient for these cases (see Fig. 5.16). Fortunately, this dilemma can be easily solved if the channel SNR level is known. In that case, an adaptive strategy can be applied by selecting different values of β based on different channel SNR regions.

Although knowing SNR information can offer great benefit to polar decoding, in many practical applications, channel SNR information is unknown at the decoder end; therefore, a simple channel condition estimation approach, which can roughly estimate the SNR level of channel, is useful and needs to be explored for efficient polar decoding.

In this section, a novel channel condition estimation approach is presented. This method is based on measuring λ, which is defined as the Hamming distance between $\hat{\boldsymbol{u}}\boldsymbol{G}$ and $\hat{\boldsymbol{x}}$. Recall that in Section 5.4.1 $\hat{\boldsymbol{u}}\boldsymbol{G} = \hat{\boldsymbol{x}}$, which corresponds to λ=0, indicates that it is very likely that the valid output $\hat{\boldsymbol{u}}$ has been found. Therefore, λ, as the difference between $\hat{\boldsymbol{u}}\boldsymbol{G}$ and $\hat{\boldsymbol{x}}$, can be viewed as *the number of unsatisfied constraints for valid polar decoding*. Based on this property, λ can be further used as the *metric to measure channel noise*. The relationship between λ and channel noise can be understood in an intuitive way. For low SNR case, because a large portion of transmitted bits is corrupted due to the stronger noise, many unsatisfied constraints still remain even after certain iterations of decoding; for high SNR case, because many transmitted bits are not corrupted due to good channel condition, all the constraints can be quickly satisfied after few iterations. As a result, the value of λ after certain rounds of iterations can be used to roughly estimate channel noise condition. This leads to the following channel condition estimation approach described next.

Channel condition estimation approach for polar codes: *If λ at the 2m-th iteration, denoted as*

λ(2m), is larger than a threshold value μ, then the channel condition can be assumed to be in low SNR region, otherwise in high SNR region.

Note that in the above approach λ is selected to be measured at the $2m$-th iteration. This is because the polar BP decoder typically needs $2m$ iterations to fully propagate initial channel LLRs in the decoder.

Simulation results verify the different distributions of $\lambda(2m)$ in low and high SNR regions. Fig. 5.19 shows the distributions of $\lambda(2m)$ at SNR=1.0 dB and 3.5dB for (1024, 512) polar codes, respectively. Here $m=\log_2 1024=10$. From this figure it can be seen that μ=100~200 can distinguish low and high SNR regions very well. As a result, the different distributions of λ in different SNR regions show that $\lambda(2m)$ is a good metric to roughly estimate channel SNR level.

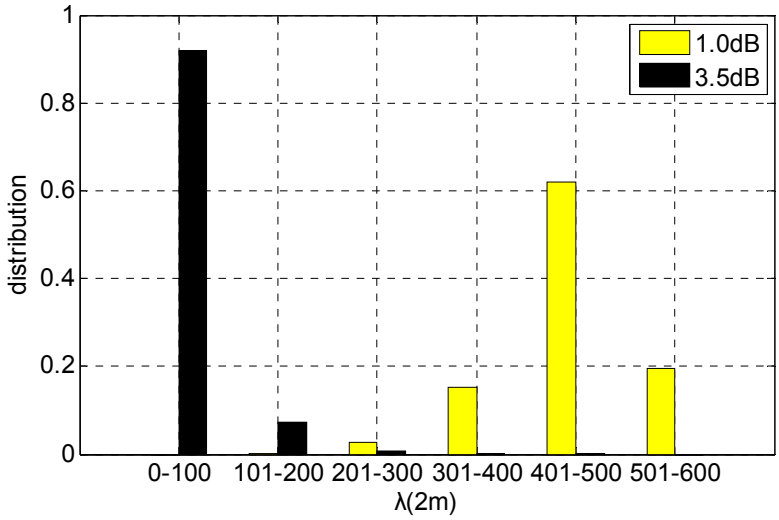


Fig. 5.19. The distribution of λ(2m) of different channel SNR values.

The proposed adaptive early stopping criterion

Based on the proposed channel condition estimation approach, a *minLLR*-based adaptive stopping criterion can be developed. In the proposed adaptive strategy, the selection of β depends on the channel SNR condition, which is determined by the value of $\lambda(2m)$. As a result, a polar BP decoder with the adaptive stopping criterion is summarized in Scheme 5.3.

Scheme 5.3 (n, k) Polar BP decoder with adaptive stopping criterion:

1: Input: Same with Scheme-A, threshold parameter μ
2: Iteration process:
3: While $t < \max\_iter$ **do**
4: Update $L_{i,j}^t$ **and** $R_{i,j}^t$ **for each node based on equation (4)**
5: Update $\hat{\mathbf{u}}, \hat{\mathbf{x}}, \min LLR$
6: if t=2m **Calculate Hamming distance** $\lambda(2m)$ **between** $\hat{\mathbf{u}}\mathbf{G}$ **and** $\hat{\mathbf{x}}$.
7: if $\lambda(2m) < \mu$ **(high SNR) larger** β **is chosen for minLLR criterion**
8: else (low SNR) smaller β **is chosen for minLLR criterion**
9: Check minLLR criterion(Scheme-B)
10: if (Satisified) 1> Decoding is assumed to be successful
11: 2> Output $\hat{\mathbf{u}}$ **& Stop iteration**
12: else $t = t + 1$ **& Begin next iteration**
13: Output : $\hat{\mathbf{u}}$

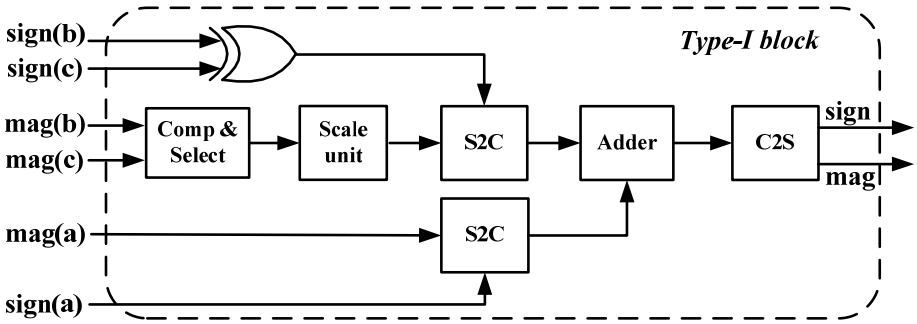
The decoding performance for polar (1024, 512) codes after using the proposed adaptive stopping criterion is shown in Fig. 5.15. It can be seen that, compared to the *minLLR* stopping criterion with fixed β, the adaptive minLLR-based stopping criterion, referred as *adaptive minLLR*, can achieve the same decoding performance with an additional 10% reduction in the number of iterations in low SNR regions (see Fig. 5.16).

5.5 Hardware Architecture

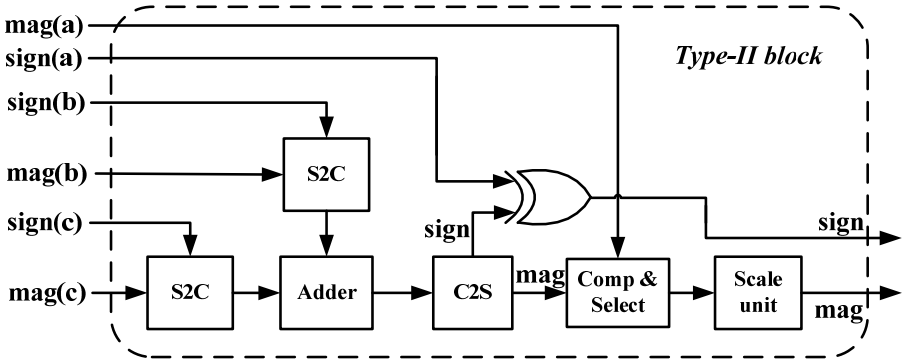
In this section, hardware architectures of BP decoders with stopping criteria are presented. Due to the generality of stopping criteria, they can be applied to any BP decoders. In this section the folded iteration-level overlapping decoders in Section 5.4 are used as the reference design.

5.5.1 Computation Element

(5.2) describes the LLR-based SMS algorithm is described by (1). In general, the four equations in (5.2) can be generalized as: Type I d=a+s***sign**(b)**sign**(c)**min**(|b|,|c|) and Type II d=s***sign**(a)**sign**(b+c)**min**(|a|,|b+c|) (see Fig. 5.20). Based on these two kinds of computation, the basic processing element (PE) of polar BP decoder [64] can be developed as Fig. 5.21.



(a)



(b)

Fig. 5.20 Architecture of computation blocks
(a) Type-I block (b) Type-II block.

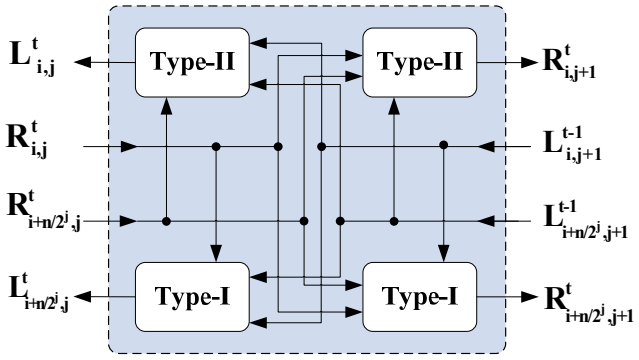


Fig. 5.21. Architecture of PE.

5.5.2 Early Stopping Block

Architecture of G-matrix stopping criterion

The lines 13~19 in Scheme 3.1 describe ***G-matrix*** stopping criterion. Fig. 5.22 shows its corresponding hardware architecture. Here ***G*** matrix block performs multiplication with ***G*** matrix. Besides, the equality detector, which contains XNORs and AND trees, is used to determine whether $\hat{\boldsymbol{x}}' = \hat{\boldsymbol{u}}\boldsymbol{G}$ is equivalent to $\hat{\boldsymbol{x}}$ or not.

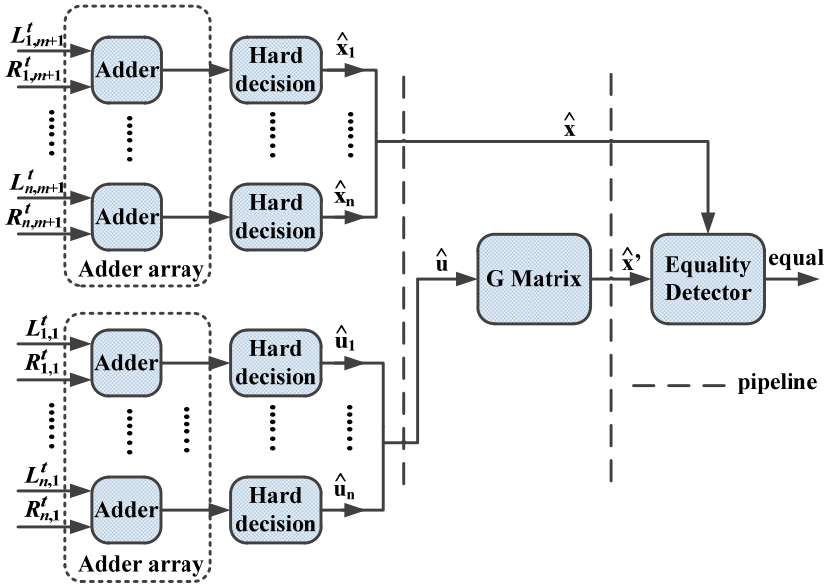


Fig. 5.22. Hardware architecture of ***G-matrix*** stopping criterion.

Timing analysis of G-matrix stopping criterion

Fig. 5.23 illustrates the worst-case decoding scheme after applying ***G-matrix*** stopping criterion to Fig. 5.9. Notice since only one codeword is processed, the subscription of 1 is omitted. Here the dashed arrows in Fig. 5.23 represent the data dependencies between stages of PEs and components of stopping criterion. In the first cycle of the *t*-th iteration, stage 1 updates the propagating messages C_1^t and outputs $L_{i,1}^t$ and $R_{i,1}^t$. In the next cycle, $L_{i,1}^t$ and $R_{i,1}^t$ are added by the adder array and then $\hat{u}_i$ is determined. Next, $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, ..., \hat{u}_{16})$ is sent to ***G*** matrix block to calculate $\hat{\boldsymbol{x}}' = \hat{\boldsymbol{u}}\boldsymbol{G}$. Similarly, after the last cycle of the *t*-th iteration, since C_4^t have been

processed by stage 2, $L^{t}_{i,m+1}$ and $R^{t}_{i,m+1}$, as the left-to-right and right-to-left LLR values of $\hat{x}_i$, are output. Then in the next two cycles, $\hat{x}_i$ is calculated and $\hat{\boldsymbol{x}}=(\hat{x}_1,\hat{x}_2,...,\hat{x}_{16})$ is sent to an equality detector.

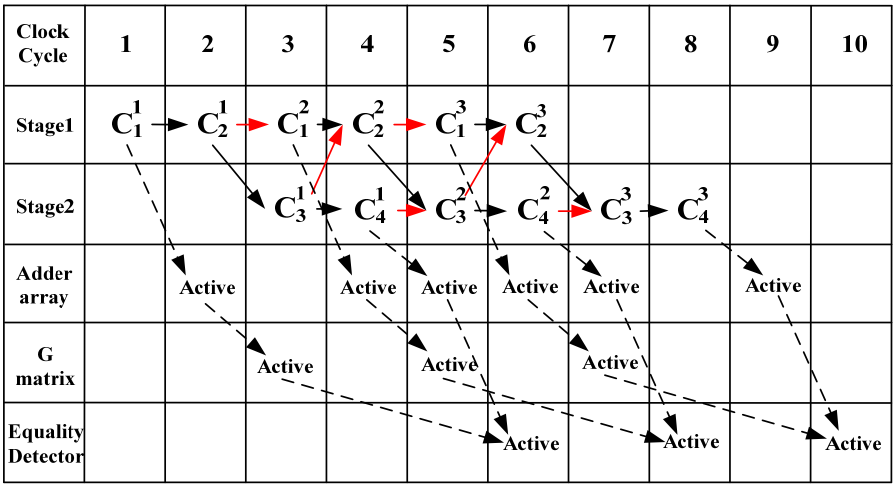


Fig. 5.23. Worst-case decoding scheme after applying ***G**-matrix* to Fig. 5.9.

For most polar codes with practical length (n≤10000), the critical path delays of the adder array, ***G*** matrix block and equality detector are always less than that of PE. Therefore, the use of ***G**-matrix* stopping criterion does not increase the critical path delay of the whole decoder. In addition, as illustrated in Fig. 5.23, compared to the case in Fig. 5.9, the use of ***G**-matrix* criterion only leads to an additional latency of two clock cycles. If we assume the whole decoding procedure is terminated at the v-th iteration, then the overall latency after using ***G**-matrix* criterion is $2v+m-2+2=2v+m$ cycles.

Architecture of minLLR stopping criterion

The lines 6~11 in Scheme 5.2 describe *minLLR* stopping criterion. Fig. 5.24 shows the corresponding hardware architecture. Here the ABS block is used to obtain the absolute value of input, and threshold comparator, which contains comparators and AND tree, is used to compare the values of *minLLR* and threshold β.

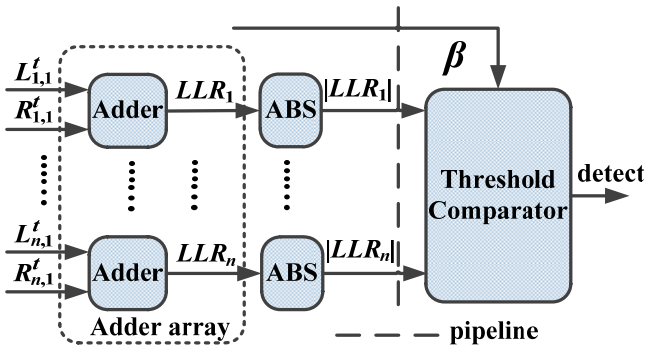


Fig. 5.24. Hardware architecture of *minLLR* stopping criterion.

Timing analysis of minLLR stopping criterion

Fig. 5.25 illustrates the worst-case decoding scheme after applying *minLLR* stopping criterion to Fig. 9. Since the critical path delays of adder array and threshold comparator are also less than that of PE, the use of *minLLR* stopping criterion does not change the overall critical path delay. Furthermore, as illustrated in Fig. 16, the use of *minLLR* criterion does not change the overall latency as well. In general, if the decoding procedure is terminated at the v-th iteration, then the overall latency after using the *minLLR* criterion is $2v+m-2$ cycles.

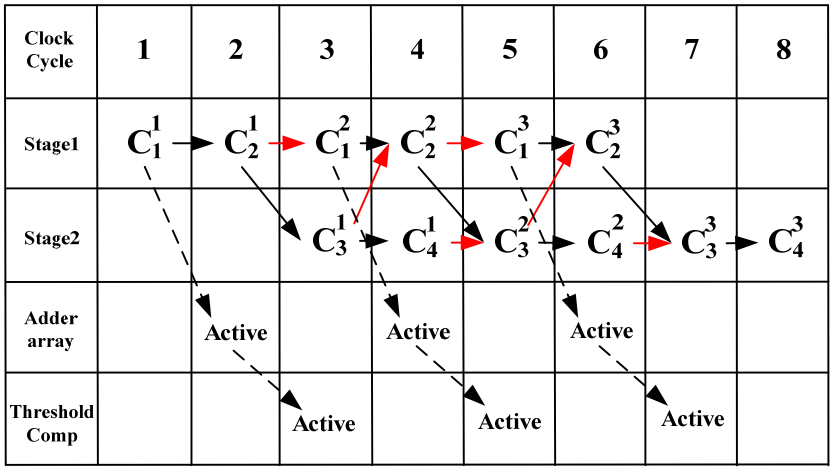


Fig. 5.25. Worst-case decoding scheme after applying *minLLR* to Fig. 5.12(b).

Architecture of channel condition estimation approach

In Section 5.4.5, channel condition estimation approach calculates the Hamming distance (HD) between $\hat{\boldsymbol{u}}\boldsymbol{G}$ and $\hat{\boldsymbol{x}}$. This computation is very similar to that in ***G-matrix*** criterion. As a result,

the architecture of channel condition estimator can be directly developed by just replacing equality detector in Fig. 5.22 with a new Hamming distance (HD) measurement block. Notice that because the HD block contains XORs and adder tree, a one-stage pipeline is needed for high-speed design, which leads to extra one cycle to the entire computation procedure (see Fig. 5.26).

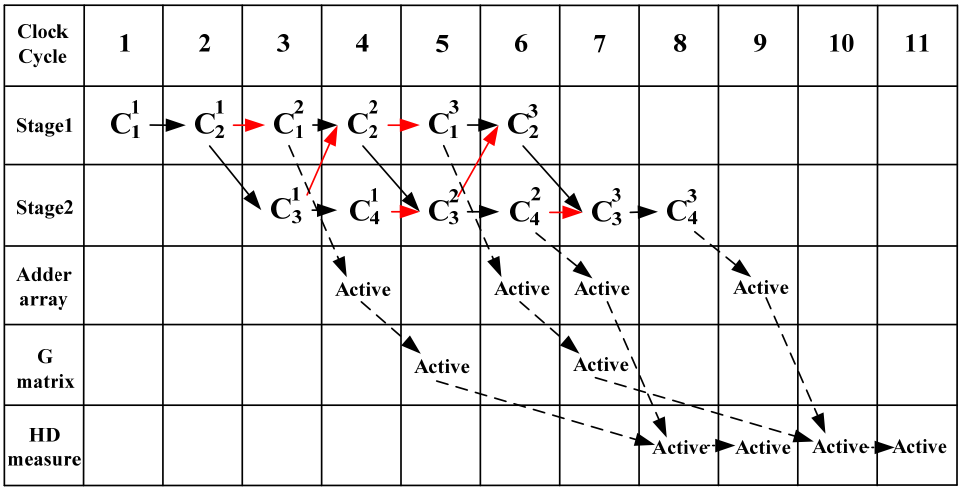


Fig. 5.26. Computing scheme of 1-stage pipelined channel condition estimator.

Timing analysis for the case with very large n

It should be noted that with n increases, the critical path delay of PE will not increase but all the other components of stopping criteria will. That means for a very large n, it is possible that the critical path delay of components of stopping criteria, such as $\boldsymbol{G}$ matrix, equality detector, threshold comparator and Hamming distance measure blocks, would be larger than that of PE. However, this problem can be solved by performing 1 or 2 stages pipelining to those components with only 1 or 2 extra clock cycles for the overall latency. Notice that in current design, 1-stage pipelining has been applied to Hamming distance measure block to reduce its critical path delay, and it shows that the use of pipelining only leads to 1 extra clock cycle to the overall latency (see Fig. 5.26). In summary, for the large n case, 1 or 2 stage pipelining to the key components of stopping criteria can guarantee the critical path of the overall decoder always locates in the PE. As a result, the use of stopping criterion will not affect the maximum clock frequency, but will

only slightly increase the latency.

5.6 Hardware Performance

In this section, the improvement on hardware performance of polar BP decoders with the use of the proposed early stopping criteria is analyzed. Table 5.1 shows the reduction in the average number of iterations and performance loss for different early stopping criteria in different SNR regions. Here (1024, 512) polar BP decoder with *max_iter*=40 is used as the original referenced decoder. From this table it can be seen that, the proposed criteria are very useful for reducing the number of iterations with keeping the same decoding performance. For ***G-matrix*** criterion it can save 23.0%~42.5% iterations for SNR in the range of 2.5dB to 3.5dB. For adaptive *minLLR* criterion, it can save 10.7%~23.2% iterations with negligible performance loss.

Table 5.1. Average number of iterations and performance loss

Stopping criterion	**The proposed *G-matrix***			**The proposed adaptive *minLLR* *(β=2.5/9.5 for low/high SNR)***		
SNR	**Average iterations**	**Iteration reduction**	**Perform degradation**	**Average iterations**	**Iteration reduction**	**Perform degradation**
2.5dB	30.8	23.0%	No	35.7	10.7%	No
3.0dB	26.1	34.7%	No	33.9	15.2%	No
3.5dB	23.0	42.5%	No	30.7	23.2%	<0.05dB

The RTL models of the proposed polar (1024, 512) BP decoders are developed with Verilog HDL. Here the I/O buffers and memories that store channel LLRs, decoded bits and hard/soft information are included in the hardware design. Then the designs are synthesized by Synopsys Design Compiler with FreePDK CMOS 45nm library. The supply voltage is 1.1 volts with typical timing model at 27 C. Table 5.2 list the critical path delays of key components in the proposed designs. From this table it can be seen that the longest path is still in the PE, hence the use of stopping criterion does not affect the maximum clock frequency.

Table 5.2. The critical path delay of key components

Block	**PE**	**G matrix**	**Equality detector**	**Threshold Comparator**	**Hamming Distance Measure (pipelined)**
Location	PE	*G-matrix Channel condition estimator*	*G-matrix*	*minLLR*	*Channel condition estimator*
Critical path delay(*ns*)	1.9337	1.3868	0.7666	1.8381	0.9795

Table 5.3 compares the hardware performance of the polar (1024, 512) BP decoder before and after using the proposed early stopping criteria. It can be seen that, after using the stopping criteria, the average decoding latency is reduced greatly with very small overhead (2% and 5% for ***G**-matrix* and adaptive *minLLR*, respectively). More importantly, the use of stopping criteria leads to great reduction in energy dissipation. Here energy per bit (EPB) [61] is used as the metric to evaluate required energy consumption for decoding process. The value of EPB is calculated as:

$$Energy\ per\ bit(EPB) = \frac{power \times decoding\ latency}{clock\ frequency \times n},$$

where the power of each design is reported by Design Compiler and n is 1024. Notice the unit of latency is clock cycle.

From Table 5.3 it can be seen that, the proposed ***G**-matrix* and adaptive *minLLR* stopping criteria can reduce EPB at 3.5dB by 32.9% and 11.2%, respectively. This means for decoding the same codeword, the decoder with the stopping criteria can save 11%~30% energy without performance loss, as compared to the decoder with fixed number of iterations. As a result, the use of stopping criteria is a powerful and low-complexity solution for saving the energy of polar BP decoders.

In addition, because the average decoding latency is reduced significantly, the average decoding throughput increases. Here similar to the case for iterative LDPC decoder, the decoding throughput for iterative polar decoder is calculated as:

$$Decoding\ throughput = \frac{clock\ frequency \times k}{decoding\ latency}.$$

Compared to the original decoder with fixed number of iterations, the ***G-matrix*** and adaptive *minLLR* stopping criteria can improve the average throughput of polar BP decoder by 55.1% and 20.6% at 3.5dB, respectively.

Table 5.3. Hardware performance of (1024, 512) polar BP decoders

Design	**Decoder without use of stopping criteria (fixed number of iter=40)**	**Decoder with *G-matrix***	**Decoder with adaptive *minLLR***
Hardware Architecture	5-stage folded iteration-level overlapping architecture		
CMOS Technology	45nm		
Maximum Clock Frequency (MHz)	500		
Total Gate Counts	1920500	1961584	2018993
Average number of iterations @3.5dB	40	23.0	30.7
Average Latency (cycles) @3.5dB	88	56	73
Energy per bit (pJ/bit) @3.5dB	328	220	291
Average Throughput (Gbps) @3.5dB	2.9	4.5	3.5

In addition, Fig. 5.27 shows the distribution of throughput of BP decoders operated at 3.5dB. Here the distribution of throughput is derived from the distribution of number of iterations. It can be seen that, for the decoder with adaptive *minLLR* stopping criterion, the distribution on the range of 3.3~4.4Gbps is much larger than other cases, therefore the actual throughput is located in this range, which is consistent with Table III. For the decoders with ***G-matrix*** stopping criterion, the probability of being larger than 4.4Gbps is much larger than other cases. This indicates that the codewords are always decoded in a short time. As a result, the actual throughput is rarely degraded by the long decoding time of the successive codewords. In general, for the BP decoders with the stopping criteria, their high throughput can be guaranteed.

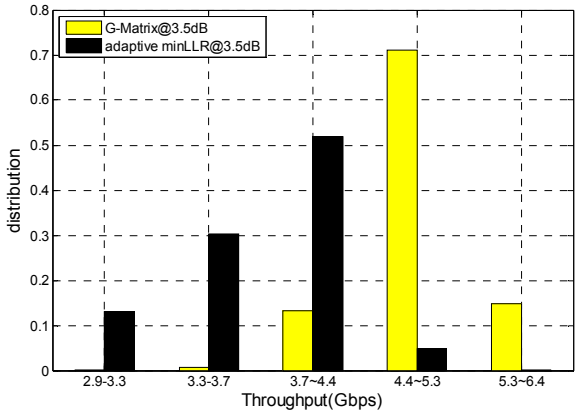


Fig. 5.27. Distribution of throughput with the use of stopping criteria.

5.7 Conclusion

In this section, the hardware architecture of polar BP decoders is optimized to reduce decoding latency and hardware complexity. Then, several novel stopping criteria and a channel condition estimation approach are proposed. Based on the proposed optimizing techniques and stopping criteria, energy-efficient low-latency BP decoders are developed. Analysis shows that the use of stopping criteria can lead to significant improvements in hardware performance with small overhead.

Chapter 6

6 STOCHASTIC SC DECODER

In this chapter, we present the stochastic SC decoder. Section 6.2 reviews the stochastic computation. In Section 6.3, the stochastic SC algorithm is derived. Section 6.4 presents several techniques that can improve the error-correcting performance of stochastic SC algorithm.

6.1 Introduction

Proposed in [34], stochastic computation had a short prosperous history in 1970's. With the development of semi-conduct industry and computer science, the long latency of stochastic computing became unsuitable for the practical use and hence was inferior to the deterministic computation, which became the dominated number representation in any digital systems.

However, in the emerging nanoscale CMOS era, stochastic computation is re-gaining the attention from researchers due to its inherent error-tolerant capability. To date, stochastic computation has been applied to various application systems, including image processing [37][65-68], signal processing [36][69-91], industrial control [92], and fundamental arithmetic function [93-102].

6.2 Stochastic Computation

Different from conventional deterministic computation, stochastic computation uses bit-stream to encode number. Here the portion of "1" in the entire bit-stream represents the required real number. Based on this representation scheme, the same real value within the range of [0, 1] can be represented by different bit streams. For example, 0.4 can be represented by 3 different length-10 bit streams (see Fig. 6.1(a)).

A big advantage of stochastic computing system is its low hardware cost. For an instance, the multiplication in stochastic computing system needs only one AND gate (see Fig. 6.1(b)), which has much less hardware complexity than the implementation in the deterministic computing scenario. In addition, the maximum clock frequency that this stochastic multiplier can achieve is also much higher than the traditional multiplier. As a result, the stochastic computing system may be very suitable for area-efficient high-speed applications.

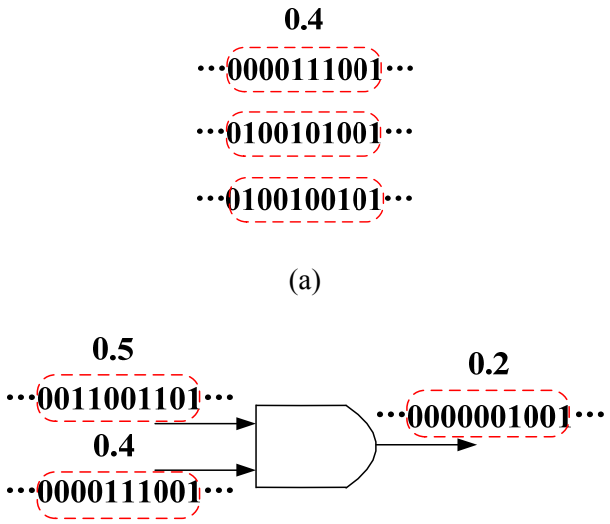


Fig. 6.1. Example of Stochastic computation
F (a) stochastic bit streams (b) stochastic multiplier.

More importantly, the stochastic computation is inherently error resilient, which is extremely valuable in naonscale CMOS era. Since the flipping of some bits in the entire bit-stream does not affect the represented number significantly, the fault tolerance of the stochastic computation is much stronger than its deterministic counterpart. This unique feature is very attractive and beneficial to the design of fault-tolerant computing architecture.

6.3 Stochastic SC Algorithm

6.3.1 Channel Message Conversion

In order to design a stochastic SC decoder, we need to first convert the original deterministic channel output iinto stochastic form. Since these channel messages are based on LR form, we can derive the following likelihood information:

$$\Pr(y_i = 1) = \frac{e^{LR(y_i)}}{e^{LR(y_i)} + 1}. \tag{6.1}$$

Notice that $\Pr(y_i=1)$ is within range [0, 1]; hence it can be represented by a bit-stream. As a result, for the stochastic SC decoder, we use the bit-stream that represents $\Pr(y_i=1)$ as the input instead of $LR(y_i)$. This choice is based on the convenient transformation from $\Pr(y_i=1)$ to $\Pr(y_i=0)$ in stochastic computing (only using a NOT gate), and it is also consistent with the case of other stochastic channel decoders [72].

Fig. 6.2 shows the architecture of an input bit-stream generator, which consists of a comparator, a lookup table (LUT) and a pseudo-random number generator.

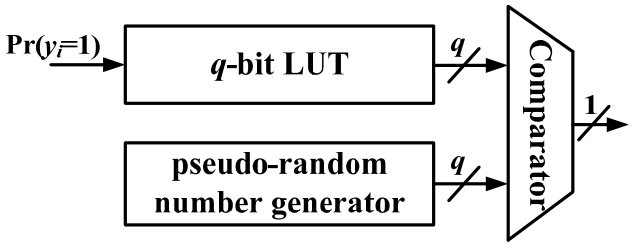


Fig. 6.2. The architecture of an input bit-stream generator.

6.3.2 Stochastic f Node

To be compatible with stochastic bit-stream, the original deterministic **f** and **g** nodes in SC decoder need to be converted to stochastic forms as well. In this section we first consider the reformulation of **f** node.

Recall that the function of **f** node is described in (2.5), which has two LR-based inputs a and b. According to the definition of LR, we have $a = \frac{\Pr(a=0)}{\Pr(a=1)}$ and $b = \frac{\Pr(b=0)}{\Pr(b=1)}$.

Based on the above denotation, we can re-write (2.5) as:

$$c = \frac{\Pr(c=0)}{\Pr(c=1)} = f(a,b) = \frac{1+ab}{a+b} = \frac{1+\frac{\Pr(a=0)}{\Pr(a=1)}\frac{\Pr(b=0)}{\Pr(b=1)}}{\frac{\Pr(a=0)}{\Pr(a=1)}+\frac{\Pr(b=0)}{\Pr(b=1)}}$$
$$= \frac{\Pr(a=1)\Pr(b=1)+\Pr(a=0)\Pr(b=0)}{\Pr(a=0)\Pr(b=1)+\Pr(a=1)\Pr(b=0)}. \quad (6.2)$$

From (6.2) it can be noted that the output of **f** node is the ratio of numerator and denominator, whose sum equals 1. As a result, we have:

$$P_c \triangleq \Pr(c=1) = \Pr(a=0)\Pr(b=1)+\Pr(a=1)\Pr(b=0) = P_a(1-P_b)+P_b(1-P_a), \quad (6.3)$$

where $P_a \triangleq \Pr(a=1)$ and $P_b \triangleq \Pr(b=1)$.

The function of stochastic **f** node is described by (6.3). Here we use P_c=Pr(c=1) as the output of **f** node, which is consistent with the choice in Section 6.3.1.

Based on (6.3), the stochastic **f** node can be implemented using a XOR gate as shown in Fig. 6.3.

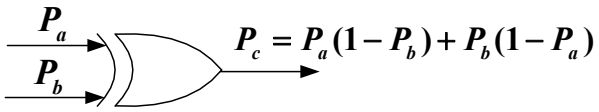


Fig. 6.3. Architecture of stochastic **f** node.

6.3.3 Stochastic g Node

(2.6) describes the function of deterministic **g** node. Again, we need to reformulate this LR-based function to likelihood form. Considering the case when $\hat{u}_{sum} = 0$, then we have:

$$c = \frac{\Pr(c=0)}{\Pr(c=1)} = g(a,b,0) = ab = \frac{\Pr(a=0)}{\Pr(a=1)}\frac{\Pr(b=0)}{\Pr(b=1)}. \tag{6.4}$$

Notice that the sum of numerator and denominator in (6.4) is not 1; hence we need to scale it and have Pr(c=1) as follows:

$$\begin{aligned} P_c &= \Pr(c=1) = \frac{\Pr(a=1)\Pr(b=1)}{\Pr(a=1)\Pr(b=1)+\Pr(a=0)\Pr(b=0)} \\ &= \frac{P_a P_b}{P_a P_b + (1-P_a)(1-P_b)}. \end{aligned} \tag{6.5}$$

where P_a and P_b are defined in Section 6.3.2.

For the case when $\hat{u}_{sum} = 1$, we have:

$$c = \frac{\Pr(c=0)}{\Pr(c=1)} = g(a,b,1) = \frac{b}{a} = \frac{\Pr(a=1)}{\Pr(a=0)}\frac{\Pr(b=0)}{\Pr(b=1)}. \tag{6.6}$$

Similarly, Pr(c=1) is derived as follows:

$$\begin{aligned} P_c &= P(c=1) = \frac{P(a=0)P(b=1)}{P(a=0)P(b=1)+P(a=1)P(b=0)} \\ &= \frac{(1-P_a)P_b}{(1-P_a)P_b + P_a(1-P_b)}. \end{aligned} \tag{6.7}$$

In summary, (6.5) and (6.7) depict the function of a stochastic **g** node. Accordingly, its hardware architecture is shown in Fig. 6.4.

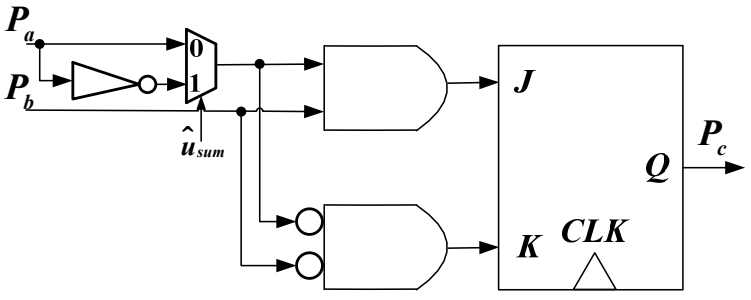


Fig. 6.4. Architecture of a stochastic **g** node.

6.4 Improving Decoding Performance

In Section 6.3 the original deterministic SC decoder has been reformulated to stochastic form. However, due to the approximation in stochastic computing, straightforward use of stochastic **f** and **g** nodes in Fig. 6.3 and Fig. 6.4 will cause performance loss. As a result, in this section we analyze several approaches that can potentially improve the decoding performance of stochastic SC decoder.

6.4.1 Channel Message Scaling

In [72], channel message scaling technique was proposed to improve error-correcting performance of stochastic LDPC decoder. In this approach, $LR(y_i)'=\alpha N_0 \mathrm{LR}(y_i)$, instead of the original channel message $\mathrm{LR}(y_i)$, is used for generating input bit-streams. Here N_0 is the single-sided noise power density. Accordingly, (6.1) is re-written as:

$$\Pr(y_i = 1) \approx \frac{e^{LR(y_i)'}}{e^{LR(y_i)'}+1} = \frac{e^{\alpha N_0 LR(y_i)}}{e^{\alpha N_0 LR(y_i)}+1}. \tag{6.8}$$

In [72], it was reported that approximation in (6.8) can partially compensate the performance loss caused by stochastic computation in LDPC decoding. For stochastic SC decoder, a similar phenomenon is observed as well. As shown in Fig. 6.5, with α=0.5, different rate-1/2 stochastic SC decoders achieve significant coding gain over the ones without channel message scaling.

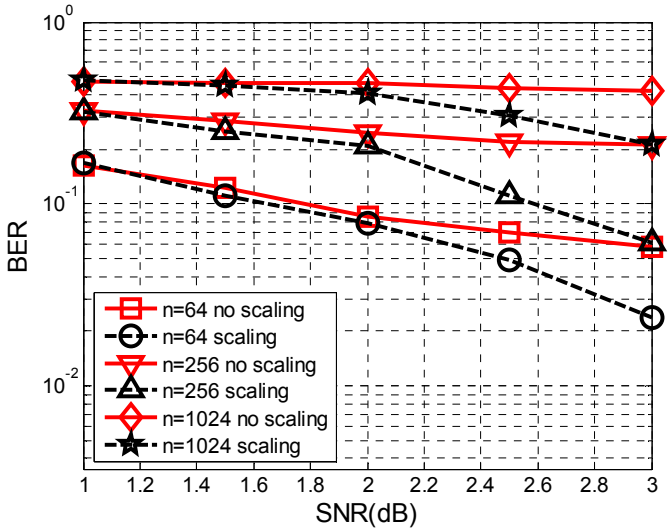


Fig. 6.5. Simulation results for stochastic SC decoders with length-128 bit-stream.

6.4.2 Increasing Length of Bit-stream

In general, the accuracy of stochastic computing is improved with the increase of length of the bit-stream. This is due to the improved precision for representation scheme. For the scheme that uses length-2^s bit-stream, the precision is $1/2^s$.

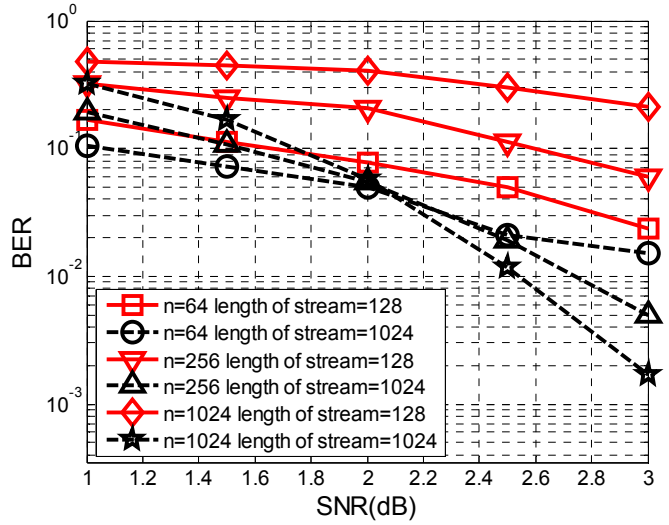


Fig. 6.6. Simulation results for stochastic SC decoders with channel message scaling.

In [36], length-128 bit-stream was used for stochastic LDPC decoding. However, as shown in Fig. 6.6, a longer bit-stream is required to overcome precision loss in stochastic SC decoding. A possible reason is that polar codes can be approximately viewed as "high-density parity-codes (HDPC)", which has more severe error propagation than LDPC codes. As a result, in the proposed stochastic SC decoding the length of bit-stream is selected as 1024.

6.4.3 Re-randomizing Bit-stream

In stochastic computing system, the randomness of generated bit-stream is gradually lost. In that case, re-randomizing the bit-stream is needed. As seen in Fig. 6.7, with the re-randomizing technique, only negligible performance loss is observed for different rate-1/2 stochastic SC decoders as compared to the conventional deterministic decoders.

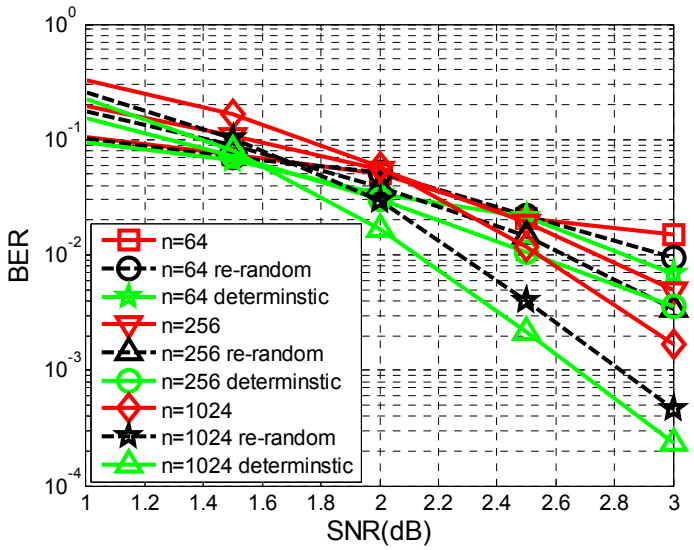


Fig. 6.7. Simulation results for stochastic SC decoders with length-1024 bit-stream.

6.5 Conclusion

In this section, the performance of stochastic SC polar code decoder is investigated. Various potential approaches that can improve decoding performance are analyzed and discussed. It is shown that the stochastic SC decoder achieves similar error-correcting performance to its

deterministic counterpart, and this paves the way for future VLSI design of stochastic polar decoders.

Chapter 7

7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

This thesis consider several parts of algorithm and VLSI co-design for polar codes decoder, including SC decoder, SCL decoder, BP decoder and stochastic SC decoder with the goal to achieve significant improvement on hardware performance with negligible error-correcting performance loss.

A reduced-latency SC decoding algorithm that can output 2 bits in one cycle was proposed. As a result, the entire decoding latency can be reduced from (2n-2) cycles to (1.5n-2) cycles without any performance loss. Then, the 2b-SC-Overlapped-scheduling decoding architecture was proposed that can be further reduced the decoding latency to n-1. Beyond that, the use of precomputation and look-ahead techniques further reduces the latency from (n-1) cycles to (3n/4-1) cycles. Synthesis results showed that the proposed decoder architecture for example (1024, 512) polar codes has significant improvement on throughput and hardware efficiency than the prior works.

Based on the idea of 2 bit decoding for SC algorithm, we further derived a general multi-bit decision algorithm for SC list decoding, referred as 2^Kb-SCL algorithm, which can determine 2^K bits at the same time without any performance loss. Then, hardware architecture of the 2^Kb-SCL decoder was developed. In particular, data path balancing technique was presented to reduce the overall critical path. Synthesis results showed that the proposed (1024, 512) 2b-rSCL and 4b-rSCL decoders have significant reduction in latency and throughput as compared to the existing SCL decoders.

Besides addressing the challenge of long latency, this thesis also presented the LLR-based SCL

decoder to reduce the overall silicon area. The proposed new algorithm, namely LLR-2^Kb-SCL algorithm, can determine 2^K bits in one cycle for arbitrary K with the use of LLR messages. As a result, it can achieve both low-complexity and short latency. Based on this new algorithm, a VLSI architecture of the SCL decoder was developed. Synthesis results showed that the proposed example (1024, 512) LLR-4b-SCL decoder achieves great reduction in both area and latency as compared to the prior works, respectively.

BP algorithm is another commonly used approach for polar codes decoding. This thesis performed systematic investigation of optimizing BP decoders. A scaled min-sum algorithm was proposed to improve the error-correcting performance of BP algorithm. In addition, several VLSI DSP optimizing techniques, including folding and overlapped-scheduling, were applied to BP architecture to improve hardware resource utilization. Furthermore, several novel early-stopping criteria were developed to reduce the number of required iterations; hence the entire energy dissipation and decoding latency can be reduced linearly. Synthesis results showed that the proposed BP decoder for (1024, 512) polar codes achieve significant reduction in energy consumption and decoding latency as compared to the prior BP decoder designs.

Stochastic computation is an old but re-surging computing paradigm. This thesis investigated the behavior of SC decoder in the scenario of stochastic computation. The original deterministic SC algorithm was reformulated with the stochastic form. Then, several techniques that can improve the error-correcting performance of stochastic SC decoders were analyzed and discussed. Simulation results showed that the stochastic SC decoders can achieve the similar decoding performance with their deterministic counterparts.

7.2 Future Work

Starting from their invention in 2008, the investigations of polar codes have gained numerous achievements, such as systematic encoding and SC list decoding. However, there are still several challenges that need to be solved before the practical use of polar codes.

First, code construction is an important aspect for practical use of channel codes. To date the relationship between frozen bit positions and performance over non-BEC channel is still an open problem. Future works will investigate efficient selection of frozen bit positions over many practical channel models.

Second, in order to achieve beyond-LDPC performance, the list size of SCL decoders are usually very large, which leads to huge cost on silicon area and power consumption. Future research will be directed towards the efficient algorithm and architecture design for small list-size polar codes decoder with the outstanding error-correcting performance.

Third, to date the BP algorithm is not comparable to SCL algorithm in term of error-correcting performance. Because list decoding strategy cannot be directly applied to the BP algorithm, how to improve the error-correcting performance of BP algorithm is an interesting topic. Future investigation will be performed on the design of beyond-LDPC BP decoders.

Reference

[1] C. E. Shannon, “A mathematical theory of communication,” Bell Syst. Tech. J., vol. 27, pp. 379-423 and 623-656, 1948.

[2] C. Berrou, A. Glavieus, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo codes,” in *Proc. 1993 Int. Conf. Commun.*, pp. 1064-1070, May 1993.

[3] R. G. Gallager, *Low-Density Parity-Check Codes. Cambridge*, MA: MIT Press, 1963.

[4] D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low-density parity-check codes,” *Elect. Lett.*, vol. 32, pp. 1645-1646, Aug. 1996.

[5] E. Arıkan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051-3073, 2009.

[6] I. Tal and A. Vardy, “List decoding of polar codes,” arXiv:1206.0050, May 2012.

[7] I. Tal and A. Vardy, “How to construct polar codes,” arXiv: 1105.6164v1, May 2011.

[8] A. Alamdar-Yazdi and F. R. Kschischang, “A simplified successive-cancellation decoder for polar codes,” *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378-1380, 2011.

[9] K. Chen, K. Niu, and J. Lin, “Improved successive cancellation decoding of polar codes,” *IEEE Trans. Commun.*, vol. 61, no. 8, pp. 3100–3107, Aug, 2013.

[10] K. Niu and K. Chen, “Stack decoding of polar codes,” *Elect. Lett.*, vol. 48, no. 12, pp. 695-696, 2012.

[11] K. Chen, K. Niu and J. R. Lin, “List successive cancellation decoding of polar codes,” *Elect. Lett.*, vol. 48, no. 9, pp. 500-501, 2012.

[12] A. Eslami and H. Pishro-Nik, “On bit error rate performance of polar codes in finite regime,” in *Proc. 48th Annual Allerton Conference on Communication, Control, and Computing*, Sept. 2010.

[13] E. Arıkan, “Polar codes: A pipelined implementation,” in *Proc. 4th Int. Symp. on Broad. Commun. ISBC 2010*, pp. 11-14, July 2010.

[14] E. Arikan, “A performance comparison of polar codes and Reed-Muller codes,” *IEEE Commun. Lett.*, vol. 12, no. 6, pp. 447-449, June 2008.

[15] E. Arikan, and E. Telatar, “On the rate of channel polarization,” in Proc. *IEEE Int. Symp. Inform. Theory (ISIT)*, pp. 1493-1495, July 2009.

[16] R. Mori and T. Tanaka, “Performance of polar codes with the construction using density evolution,” *IEEE Commun. Lett.*, vol. 13, no. 7, pp. 519-521, July 2009.

[17] S. B. Korada, E. Sasoglu, and R. Urbanke, “Polar Codes: Characterization of Exponent, Bounds, and Constructions,” *IEEE Trans. Inf. Theory*, vol. 56, no. 12, pp. 6253-6264, 2010.

[18] N. Hussami, S. B. Korada, and R. Urbanke, “Performance of polar codes for channel and source coding,” in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, pp. 1488-1492, July 2009.

[19] S. B. Korada, and R. L. Urbanke, “Polar codes are optimal for lossy source coding,” *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1751-1768, 2010.

[20] R. Pedarsani, S. H. Hassani, I. Tal, and E. Telatar, “On the construction of polar codes,” in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, pp. 11-15, July 2011.

[21] Z. Huang, C. Diao, J. Dai, C. Duanmu, X. Wu and M. Chen, “An improvement of modified successive-cancellation decoder for polar codes,” *IEEE Commun. Lett.*, vol. 7, no. 12, pp. 957-961, July 2013.

[22] E. Hof and S. Shamai, “Secrecy-achieving polar-coding,” in *Proc. of IEEE Information Theory Workshop (ITW)*, pp. 1-5, 2010.

[23] E. Koyluoglu and H. El Gamal, “Polar coding for secure transmission and key agreement,” in *Proc. IEEE Int. Symp. Personal, Indoor and Mobile Radio Commun. Conf (PIMRC)*, pp. 2698-2703, 2010.

[24] D. M. Shin, S. C. Lim, and K. Yang, “Mapping selection and code construction for 2^m-ary polar-coded modulation,” *IEEE Commun. Lett.*, vol. 16, no. 6, pp. 905-908, 2012.

[25] S. Cayci, O. Arikan, and E. Arikan, “Polar code construction for non-binary source alphabets,” in *Proc. 20th Signal Processing and Commu. Applications Conference (SIU)*, pp. 1-4, 2012.

[26] B. Li, H. Shen, and D. Tse, “An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check,” *IEEE Comm. Letters*, vol. 16, pp. 2044–2047, Dec. 2012.

[27] B. Li, H. Shen and D. Tse, “Parallel decoders of polar codes,” arXiv: 1309.1026v1, Sep. 2013.

[28] E. Abbe, and E. Telatar, “MAC polar codes and matroids,” *Information Theory and Applications Workshop (ITA)*, pp. 1-8, 2010.

[29] N. Goela, S. B. Korada, and M. Gastpar, “On LP decoding of polar codes,” in *Proc. IEEE Information Theory Workshop (ITW)*, pp. 1-5, Aug. 2010.

[30] E. Arikan, “Systematic polar coding,” *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 860-862, 2011.

[31] B. Yuan and K.K. Parhi, "Successive cancellation list polar decoder using Log-likelihood ratios," in *Proc. of Asilomar Conf. on Signal, Systems and Computers*, pp. 548-552, 2014.

[32] A. Balatsoukas-Stimming, M. Bastani Parizi and A. Burg, “LLR-based successive cancellation list decoding of polar codes,” in *Proc. of 2014 IEEE Intl. Conf. on Acoustics, Speech and Signal Procesing (ICASSP)*, pp. 3903–3907, May 2014.

[33] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 498-519, Feb. 2001.

[34] B. Gaines, "Stochastic computing systems," *Advances in Information Systems Science*, vol. 2, no. 2, pp. 37–172, 1969.

[35] B. Brown and H. Card, "Stochastic neural computation I: computational elements," *IEEE Trans. Comput.*, vol. 50, pp. 891-905, Sept. 2001.

[36] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. on Signal Processing*, vol. 56, no. 11, pp. 5692–5703, Nov.2008.

[37] P. Li and D. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Proc. of 29th International Conference on Computer Design (ICCD)*, pp. 154–161, 2011.

[38] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders", *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725-728, Apr. 2013.

[39] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: algorithm and implementation", arXiv:1307.7154v2.

[40] K.K. Parhi, and D.G. Messerschmitt, "Static rate-optimal scheduling of iterative data flow programs via optimum unfolding," *IEEE Trans. on Computers*, vol. 40, no. 2, pp. 178-195, Feb. 1991.

[41] K.K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 7, pp. 745-754, July 1991.

[42] K.K. Parhi, "Design of multi-gigabit multiplexer loop based decision feedback equalizers," *IEEE Trans. on VLSI Systems*, vol. 13, no. 4, pp. 489-493, Apr. 2005.

[43] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, New York, NY: John Wiley & Sons Inc., 1999.

[44] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware Architectures for Successive Cancellation Decoding of Polar Codes," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1665-1668, May 2011.

[45] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Processing*, vol. 61, no. 2, pp. 289-299, Jan. 2013.

[46] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency SC polar decoder architectures," in *Proc. Int. Conf. Commun.*, pp. 3471-3475, June 2012.

[47] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Trans. Signal Processing*, vol. 61, no. 10, pp. 2429-2441, May, 2013.

[48] Z. Cui and Z. Wang, "A 170 Mbps (8176, 7156) quasi-cyclic LDPC decoder implementation with FPGA," in *Proc. of IEEE Int. Symp. on Circuits and Systems*, pp. 5095-5098, May 2006.

[49] D. Oh and K. K. Parhi, "Minsum decoder architecture with reduced word-length for LDPC codes", *IEEE Trans. Circuits and Systems-I: Regular Papers*, vol. 57, no. 1, pp. 105-115, Jan. 2010.

[50] A. Mishra, A. Raymond, L. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. J. Gross, "A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS," *IEEE Asian Solid-State Circuits Conference(A-SSCC)*, 2012.

[51] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross and A. Burg, "Hardware architecture for list SC decoding of polar codes", arXiv:1303.7127v3.

[52] D.E. Knuth, *The Art of Computer Programming*, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1998.

[53] B. Yuan and K.K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multi-bit decision," accepted by *IEEE Trans. on VLSI Systems*, 2015.

[54] A. Pamuk, “An FPGA implementation architecture for decoding of polar codes,” in *Proc. 8th Int. Symp. on Wireless Commun. Syst. (ICWCS)*, pp. 437-441, Nov. 2011.

[55] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X. Y. Hu, “Reduced complexity decoding of LDPC codes,” *IEEE Trans. on Communications*, vol. 53, no. 8, pp. 1288-1299, Aug. 2005.

[56] K.K. Parhi, C.Y. Wang, A.P. Brown, "Synthesis of control circuits in folded pipelined DSP Architectures," *IEEE Journal of Solid State circuits*, vol. 27, no. 1, pp. 29-43, Jan. 1992,

[57] R. Y. Shao, S. Lin, and M. P. C. Fossorier, “Two simple stopping criteria for Turbo decoding,” *IEEE Trans. on Commun*., vol. 47, no. 8, pp. 1117-1120, Aug. 1999.

[58] J. Li, X-h You, and J. Li, “Early stopping for LDPC decoding: convergence of mean magnitude (CMM),” *IEEE Commun. Lett.*, vol. 10, no. 9, pp. 667-669, Sep. 2006.

[59] W. J. Ebel, Y. Wu, and B. D. Woerner, “A simple stopping criterion for Turbo decoding,” *IEEE Commun. Lett.*, vol. 4, no. 8, pp. 258-260, 2000.

[60] Z. Wang and K. K. Parhi, “On-line extraction of soft decoding information and applications in VLSI Turbo decoding,” *IEEE Trans. on Circuits and Systems- II: Analog and digital signal processing*, vol. 49, no. 12, pp. 760-769, Dec. 2002.

[61] T. Mohsenin, H. Shirani-mehr, and B. Baas, “Low power LDPC decoder with efficient stopping scheme for undecodable block,” in *Proc. of 2011 IEEE Int. Symp. on Circuits and Systems*, pp. 1780-1783, May 2011.

[62] Z. Cui, L Chen and Z. Wang, “An efficient early stopping scheme for LDPC decoding,” in *Proc. of 13th NASA Symp. on VLSI design*, 2007.

[63] R. E. Blahut, *Theory and Practice of Error-Control Codes*. Reading,MA: Addison-Wesley, 1983.

[64] B. Yuan and K. K. Parhi, "Architecture optimizations for BP polar decoders," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2013.

[65] P. Li and D. Lilja, "A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm," in *Proc. of IEEE International Conf. on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 161-168, 2011.

[66] Peng Li, David J. Lilja, Weikang Qian, and Kia Bazargan, "Computation on stochastic bit streams: Digital image processing case studies," *IEEE Trans. on Very Large Scale Integrated (VLSI) Systems*, vol. 22, no. 3, pp. 449-462, April 2013.

[67] A. Alaghi, Cheng Li and John P. Hayes. "Stochastic circuits for real-time image processing applications," in *Proc. Design Automation Conference (DAC)*, pp. 1-6, 2013.

[68] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Proc. Design, Automation and Test Conference in Europe (DATE)*, pp. 1-4, 2014.

[69] A. Rapley, C. Winstead, V. Gaudet and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Proc. of the 3rd Int. Symp. on Turbo Codes and Related Topics*, pp. 507–510, 2003.

[70] W. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Proc. of the 39th Asilomar Conference on Signals, Systems and Computers*, pp. 713–717, Nov. 2005.

[71] S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor and W. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, Sept. 2010.

[72] S. Sharifi Tehrani, W. Gross and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct. 2006.

[73] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, “Delayed stochastic decoding of LDPC codes,” *IEEE Trans. Signal Processing*, vol. 59, no. 11, pp. 5617-5626, Nov. 2011.

[74] A. Ciobanu, S. Hemati, and W. J. Gross, “Adaptive multiset stochastic decoding of non-binary LDPC codes,” *IEEE Transactions on Signal Processing*, vol. 61, no. 16, pp. 4100-4113, 2013,

[75] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, “Stochastic decoding of LDPC codes over GF(q)”, *IEEE Trans. on Communications*, vol. 61, no. 3, pp. 939-950, 2013.

[76] G. Sarkis and W. J. Gross, “Efficient stochastic decoding of non-binary LDPC codes with degree-two variable nodes”, *IEEE Commun. Letters*, vol. 16, no. 3, pp. 389-391, March 2012.

[77] A. Naderi, S. Mannor, M. Sawan and W. J. Gross, “Delayed stochastic decoding of LDPC Codes,” *IEEE Trans.on Signal Processing*, vol. 59, no. 11, pp. 5617-5626, Nov. 2011.

[78] M. Arzel, C. Lahuec, C. Jego, W. J. Gross, and Y. Bruned, “Stochastic multiple stream decoding of cortex codes”, *IEEE Trans. on Signal Processing*, vol. 59, no. 7, pp. 3486-3491, July 2011.

[79] Q. T. Dong, M. Arzel, C. Jego, and W. J. Gross, “Stochastic decoding of turbo codes”, *IEEE Trans. on Signal Processing*, vol. 58, no. 12, pp. 6421-6425, Dec. 2010.

[80] C. Winstead, S. Sharifi Tehrani, W. J. Gross, S. Mannor, S. Howard, and V. C. Gaudet, “Relaxation dynamics in stochastic iterative decoders”, *IEEE Trans. on Signal Processing*, vol. 58, no. 11, pp. 5955-5961, Nov. 2010.

[81] C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, “Stochastic chase decoding of Reed-Solomon codes”, *IEEE Commu. Letters*, vol. 14, no. 9, pp. 863-865, Sep. 2010.

[82] S. Sharifi Tehrani, C. Jego, B. Zhu, and W. J. Gross, “Stochastic decoding of linear block codes with high-density parity-check matrices”, *IEEE Trans. on Signal Processing*, vol. 56, no. 11, pp. 5733-5739, Nov. 2008.

[83] N. Onizawa, W. J. Gross, and T. Hanyu, “Lowering error floors in stochastic decoding of LDPC codes based on wire-delay dependent asynchronous updating”, in *Proc. of IEEE 43rd International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 254-259, May 2013.

[84] S. Sharifi Tehrani, P. Siegel, S. Mannor, and W. J. Gross, “Joint stochastic decoding of LDPC codes and partial response channels”, in *Proc. of IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 13-18, Oct. 2012.

[85] N. Onizawa, W. J. Gross, T. Hanyu, and V. Gaudet, “Clockless stochastic decoding of low-density parity-check codes”, in *Proc. of IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 143-148, Oct. 2012.

[86] R. Heloir, C. Leroux, S. Hemati, M. Arzel, and W. J. Gross, “Stochastic chase decoder for Reed-Solomon codes”, in *Proc. of 10th IEEE International NEWCAS Conference (NEWCAS)*.

[87] N. Onizawa, V. C. Gaudet, T. Hanyu, and W. J. Gross, “Asynchronous stochastic decoding of low-density parity-check codes”, in *Proc. of 2012 IEEE 42nd International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 92-97, May 2012.

[88] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, “Relaxed half-stochastic decoding of LDPC codes over GF(q)”, in *Proc. of 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 36-41, Sep. 2010.

[89] V. C. Gaudet and W. J. Gross, “Switching activity in stochastic decoders”, in *Proc. of 40th IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 167-172, May 2010.

[90] G. Sarkis and W. J. Gross, “Reduced-latency stochastic decoding of LDPC codes over GF(q)”, in *Proc. of European Wireless Conference (EW)*, pp. 994-998, April, 2010.

[91] F. Leduc-Primeau, S. Hemati, W. J. Gross, and S. Mannor, “A relaxed half-stochastic iterative decoder for LDPC codes”, in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 1-6, Nov. 2009.

[92] D. Zhang and H. Li, “A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms,” *IEEE Trans. Industrial Electronics*, vol. 55, no. 2, pp. 551–561, 2008.

[93] P. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, “The synthesis of linear finite state machine-based stochastic computational elements,” in *Proc. of 17th Asia and South Pacific, Design Automation Conference (ASP-DAC)*, pp. 757 –762, Jan. 2012.

[94] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, “The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic,” in *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pp. 480–487, Nov. 2012.

[95] W. Qian, M. D. Riedel, H. Zhou, and J. Bruck, “Transforming probabilities with combinational logic,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1279-1292, 2011.

[96] W. Qian, X. Li, Marc D. Riedel, K. Bazargan, and D. J. Lilja, “An architecture for fault-tolerant computation with stochastic logic,” *IEEE Trans. on Computers*, vol. 60, no. 1, pp. 93-105, 2011.

[97] W. Qian, C. Wang, P. Li, D. J. Lilja, K. Bazarga, and M. D. Riedel, “An efficient implementation of numerical integration using logical computation on stochastic bit streams,” in *Proc. of the 2012 IEEE/ACM International Conference on Computer-Aided Design*, pp. 156-162, 2012.

[98] W. Qian and M. D. Riedel, “Two-level logic synthesis for probabilistic computation,” in *Proc. of the 19th International Workshop on Logic and Synthesis*, pp. 95-102, 2010.

[99] W. Qian, M. D. Riedel, K. Bazargan, and D. J. Lilja, "The synthesis of combinational logic to generate probabilities," in *Proc. of the 2009 IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 367-374, 2009.

[100] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Proc. of the 45th ACM/IEEE Design Automation Conference*, pp. 648-653, 2008.

[101] W. Qian and M. D. Riedel, "The synthesis of stochastic logic to perform multivariate polynomial arithmetic," in *Proc. of the 17th International Workshop on Logic and Synthesis*, pp. 79-86, 2008.

[102] W. Qian, J. Backes, and M. D. Riedel, "The synthesis of stochastic circuits for nanoscale computation," in *Proc. of the 16th International Workshop on Logic and Synthesis*, pp. 176-183, 2007.

Appendix

To prove $\hat{u}_{2i-1} = \alpha$ and $\hat{u}_{2i} = \beta$, we show that $\mathrm{P}(\hat{u}_{2i-1}\hat{u}_{2i})$ corresponds to the largest probability among P(00), P(01), P(10) and P(11). Since $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ can be either 0 or 1, we discuss four possible cases:

Case A-1: $\hat{u}_{2i-1} = 0$ and $\hat{u}_{2i} = 0$

Recall that $\hat{u}_{2i-1}$ and $\hat{u}_{2i}$ are the outputs from the SC algorithm. Therefore, according to (2.2), when $\hat{u}_{2i-1} = 0$, $L(2i-1, m) = LR(\hat{u}_{2i-1}) \geq 1$.

According to (2.3) (2.16), we have

$$\begin{aligned} & L(2i-1, m) = LR(\hat{u}_{2i-1}) \\ & = \frac{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0) + \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1)}{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) + \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1)} \geq 1. \end{aligned}$$

Thus,

$$\begin{aligned} & \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0) + \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1) \\ & \geq \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) + \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1). \end{aligned} \tag{A.1}$$

Now we show that the largest probability $\mathrm{P}(\alpha\beta)$ must be $\mathrm{P}(00) = \mathrm{P}(c=0)\,\mathrm{P}(d=0)$ or $\mathrm{P}(01) = \mathrm{P}(c=1)\,\mathrm{P}(d=1)$.

***Proposition*-A.1:** Given equation (A.1), among P(00), P(01), P(10) and P(11), the largest probability $\mathrm{P}(\alpha\beta)$ must be P(00) or P(01).

***Proof*:** If $\mathrm{P}(\alpha\beta)$ is not P(00) or P(01), without loss of generality, assume $\mathrm{P}(\alpha\beta)$ is $\mathrm{P}(10) = \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)$. Since $\mathrm{P}(\alpha\beta) = \mathrm{P}(10)$ is the largest probability, and the sum of $\mathrm{P}(\hat{c}=1)$ and $\mathrm{P}(\hat{c}=0)$ is equal to some non-negative value *x*, then we have:

$$
\begin{aligned}
\mathrm{P}(10) > \mathrm{P}(00) &\Rightarrow \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) > \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0) \\
&\Rightarrow x - \mathrm{P}(\hat{c}=0) > \mathrm{P}(\hat{c}=0) \\
&\Rightarrow \mathrm{P}(\hat{c}=0) < x/2.
\end{aligned} \tag{A.2}
$$

Similarly, we can get:

$$
\begin{aligned}
\mathrm{P}(10) > \mathrm{P}(01) &\Rightarrow \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) > \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1) \\
&\Rightarrow \mathrm{P}(\hat{d}=0) > y - \mathrm{P}(\hat{d}=0) \\
&\Rightarrow \mathrm{P}(\hat{d}=0) > y/2,
\end{aligned} \tag{A.3}
$$

where y is the non-negative sum of $\mathrm{P}(\hat{d}=1)$ and $\mathrm{P}(\hat{d}=0)$.

Recall that for equation (A.1):

$$
\begin{aligned}
&\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0) + \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1) \\
&\geq \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) + \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1) \\
&\Rightarrow \mathrm{P}(\hat{c}=0)(\mathrm{P}(\hat{d}=0) - \mathrm{P}(\hat{d}=1)) \geq \mathrm{P}(\hat{c}=1)(\mathrm{P}(\hat{d}=0) - \mathrm{P}(\hat{d}=1)) \\
&\Rightarrow (\mathrm{P}(\hat{c}=0) - \mathrm{P}(\hat{c}=1))(\mathrm{P}(\hat{d}=0) - \mathrm{P}(\hat{d}=1)) \geq 0 \\
&\Rightarrow (2\,\mathrm{P}(\hat{c}=0) - x)(2\,\mathrm{P}(\hat{d}=0) - y) \geq 0.
\end{aligned} \tag{A.4}
$$

However, with (A.2) and (A.3) we know that $(2\,\mathrm{P}(\hat{c}=0) - x)(2\,\mathrm{P}(\hat{d}=0) - y) < 0,$ which contradicts (A.4). Therefore, $\mathrm{P}(\alpha\beta)$ can not be P(10). Similarly, it can be proved that $\mathrm{P}(\alpha\beta)$ can not be P(11). Therefore, $\mathrm{P}(\alpha\beta)$ must be P(00) or P(01). □

After proving the above proposition-A1, we now show P(00) must be larger than P(01). Since $\hat{u}_{2i-1} = 0$ and $\hat{u}_{2i} = 0,$ according to (2.3) and (2.23), we can get

$$
\begin{aligned}
&LR(\hat{u}_{2i}) = LR(\hat{c})^{1-2\hat{u}_{2i-1}}\, LR(\hat{d}) = LR(\hat{c})LR(\hat{d}) \geq 1 \\
&\Rightarrow \frac{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0)}{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1)} \geq 1 \\
&\Rightarrow \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0) \geq \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1) \\
&\Rightarrow \mathrm{P}(00) \geq \mathrm{P}(01).
\end{aligned} \tag{A.5}
$$

Since it has been proved that $\mathrm{P}(\alpha\beta)$ must be P(00) or P(01), then with (A.5), we have $\mathrm{P}(\alpha\beta) = \mathrm{P}(00) = \mathrm{P}(\hat{u}_{2i-1}\hat{u}_{2i})$.

Case A-2: $\hat{u}_{2i\text{-}1}=0$ and $\hat{u}_{2i}=1$

Similar to the case A-1, when $\hat{u}_{2i\text{-}1}=0$, $\mathrm{P}(\alpha\beta)$ must be P(00) or P(01).

For $\hat{u}_{2i-1}=0$ and $\hat{u}_{2i}=1$, according to (2.3) and (2.23), we can obtain

$$\begin{aligned} & LR(\hat{u}_{2i}) = LR(\hat{c})^{1-2\hat{u}_{2i-1}}\, LR(\hat{d}) = LR(\hat{c})LR(\hat{d}) < 1 \\ & \Rightarrow \frac{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0)}{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1)} < 1 \\ & \Rightarrow \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0) < \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1) \\ & \Rightarrow \mathrm{P}(00) < \mathrm{P}(01). \end{aligned} \tag{A.6}$$

Since $\mathrm{P}(\alpha\beta)$ must be P(00) or P(01), in this case, $\mathrm{P}(\alpha\beta)=\mathrm{P}(01)=\mathrm{P}(\hat{u}_{2i\text{-}1}\hat{u}_{2i})$.

Case A-3: $\hat{u}_{2i\text{-}1}=1$ and $\hat{u}_{2i}=0$

When $\hat{u}_{2i\text{-}1}=1$, according to (2.3) and (2.16), we have

$$\begin{aligned} & L(2i-1,m) = LR(\hat{u}_{2i\text{-}1}) < 1 \\ & \Rightarrow LR(\hat{u}_{2i\text{-}1}) = \frac{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0)+\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1)}{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)+\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1)} < 1 \\ & \Rightarrow \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=0)+\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=1) \\ & \quad < \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)+\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1). \end{aligned} \tag{A.7}$$

Similar to the proof of proposition-A1, it is easy to prove: From (A.7) the $\mathrm{P}(\alpha\beta)$ must be $\mathrm{P}(10)=\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)$ or $\mathrm{P}(11)=\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1)$.

Then, consider $\hat{u}_{2i}=0$, with (2.23), we can obtain that

$$\begin{aligned} & LR(\hat{u}_{2i}) = LR(c)^{1-2\hat{u}_{2i\text{-}1}}\, LR(d) = LR(c)^{-1} LR(d) \geq 1 \\ & \Rightarrow \frac{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)}{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1)} \geq 1 \\ & \Rightarrow \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) > \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1) \Rightarrow \mathrm{P}(10) > \mathrm{P}(11). \end{aligned}$$

Therefore, $\mathrm{P}(\alpha\beta)=\mathrm{P}(10)=\mathrm{P}(\hat{u}_{2i\text{-}1}\hat{u}_{2i})$.

Case A-4: $\hat{u}_{2i\text{-}1}=1$ and $\hat{u}_{2i}=1$

Similar to the case A-3, when $\hat{u}_{2i\text{-}1}=1$, $\mathrm{P}(\alpha\beta)$ must be P(10) or P(11).

For $\hat{u}_{2i-1}=1$ and $\hat{u}_{2i}=1$, according to (2.3) and (2.23), we can obtain

$$\begin{aligned} & LR(\hat{u}_{2i}) = LR(c)^{1-2\hat{u}_{2i-1}} LR(d) = LR(c)^{-1} LR(d) < 1 \\ \Rightarrow & \frac{\mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0)}{\mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1)} < 1 \\ \Rightarrow & \mathrm{P}(\hat{c}=1)\,\mathrm{P}(\hat{d}=0) < \mathrm{P}(\hat{c}=0)\,\mathrm{P}(\hat{d}=1) \Rightarrow \mathrm{P}(10) < \mathrm{P}(11). \end{aligned}$$

Therefore, the largest probability is: $\mathrm{P}(\alpha\beta) = \mathrm{P}(11) = \mathrm{P}(\hat{u}_{2i-1}\hat{u}_{2i})$.

Summarizing the above four cases, we can conclude that $\mathrm{P}(\alpha\beta) = \mathrm{P}(\hat{u}_{2i-1}\hat{u}_{2i})$ holds all the time.

Therefore, $\hat{u}_{2i-1}=\alpha$ and $\hat{u}_{2i}=\beta$. Thus, proposition 1 is proved. □