LOW-LATENCY LOW-COMPLEXITY CHANNEL DECODER ARCHITECTURES FOR MODERN COMMUNICATION SYSTEMS

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

Chuan Zhang

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

Professor Keshab K. Parhi, Advisor

December 2012

Acknowledgements

First of all, I wish to thank my advisor, Professor Keshab K. Parhi, for his continuing and fatherly encouragement, guidance, and financial support throughout my entire Ph.D. study at the University of Minnesota. Before I got the admission to University of Minnesota, I was always dreaming to join this group. I am very lucky and grateful that I have had research opportunities under Professor Parhi's supervision in the area of VLSI digital signal processing. Professor Parhi will still be my mentor even after my graduation.

I would like to thank Professor Gerald E. Sobelman, Professor Marc Riedel, and Professor Victor Reiner for their support as members of my Ph.D. committee. I would like to thank the Graduate School of University of Minnesota for their financial support by the Three-Year Graduate School Fellowship and Doctoral Dissertation Fellowship.

My thanks also go to current and former members of our research group. I would like to thank Bo Yuan for our numerous discussions on a variety of research topics. I would like to thank Yun Sang Park for his consistent engagement and help even after his graduation. I am lucky and happy to have Sohini Roy Chowdhury as one of my best friends. Also, I am grateful to Tingting Xu, Yingbo Hu, Sayed Ahmad Salehi, Te-Lung Kung, Manohar Ayinala, Yingjie Lao, Zisheng Zhang, and Lan Luo, for their kind support in my Ph.D. life.

I would like to also thank my friends at University of Minnesota and in my country China, especially Professor Andreas Stein, Zhenzhen Jiang, Ying Zhu, Jianfeng Zheng, Yao Wang, Fei Zheng, Lian Huai, Juan Du, Zhe Zhang, Jieming Yin, Huan Li, Meng Yang, and Changjiang Liu, for their assistance and engagement to continue my studies.

Lastly, I am forever grateful to my parents, my parents in law, and especially my lovely wife, Xiaoqing Chen, for their love, support, and encouragement throughout the years. Without them, I would not have completed my Ph.D. successfully, and my whole life could be of no meanings.

Abstract

Nowadays, along with the economic and technical progress, modern communication industry is playing a more and more important role in people's lives. The rapid growth of communication industry is benefiting and gradually changing our work, learning, and life styles. It is difficult to imagine what life will be like without smartphones, HDTVs, high-speed networks, and Wi-Fi hotspots. On the other hand, the ever-increasing users' demands force the modern communication systems to be faster, more portable, more reliable, and safer. As an indispensable and important part of modern communication systems, channel decoders are expected to be low-latency, low-complexity, low-error, and wiretap-free. However, developing channel decoders to meet those requirements is quite a struggle. Fortunately, VLSI digital signal processing techniques offer us great facilities to enable channel decoders to advance to new generations.

This thesis commits itself to the efficient VLSI implementation of low-latency low-complexity channel decoders. In order to make our approaches more applicable for variant real-time communication applications, formal design methodologies are proposed. Novel non-binary QC-LDPC decoders with efficient switch networks are presented. For the newly invented polar codes, a family of latency-reduced decoder architectures is also proposed. Comparisons with prior works have demonstrated that the proposed designs show advantages in both decoding throughput and hardware efficiency.

First, a novel design methodology to design low-complexity VLSI architectures for non-binary LDPC decoders is presented. By exploiting the intrinsic shifting and symmetry properties of non-binary quasi-cyclic LDPC (QC-LDPC) codes, significant

reduction of memory size and routing complexity can be achieved. These unique features lead to two network-efficient decoder architectures for Class-I and Class-II non-binary QC-LDPC codes, respectively. Comparison results with the state-of-the-art designs show that for the code example of the 64-ary (1260, 630) rate-0.5 Class-I code, the proposed scheme can save up to 70.6% hardware required by switch network, which demonstrates the efficiency of the proposed technique. The proposed design for the 32-ary (992, 496) rate-0.5 Class-II code can achieve a 93.8% switch network complexity reduction compared with conventional approaches. Furthermore, with the help of a generator for possible solution sequences, both forward and backward steps can be eliminated to offer processing convenience of check node unit (CNU) blocks. Results show that the proposed 32-ary (992, 496) rate-0.5 Class-II decoder can achieve 4.47 Mb/s decoding throughput.

Second, the low-latency sequential SC polar decoder is proposed based on the DFG analysis. The complete gate-level decoder architecture is proposed. The *feedback part* is proposed to generate control signals on-the-fly. The proposed design method is universal and can be employed to design the low-latency sequential SC polar decoder for any code-length. Compared with prior works, this design can achieve twice throughput with similar hardware consumption.

Third, in order to meet the requirements of high-throughput communication systems, both *time-constrained* (TC) and *resource-constrained* (RC) interleaved SC polar decoders are proposed. Analysis shows that the TC interleaved decoders can multiply the throughput and achieve much higher utilization. Also, the RC interleaved decoders can improve the decoding throughput while keeping the hardware complexity low. Compared

with our pre-computation sequential polar decoder design, the RC 2-interleaved decoder given here can achieve 200% throughput with only 50% hardware consumption.

Finally, the decoder design issue of the newly proposed *simplified* SC (SSC) decoding algorithm for polar codes is investigated. Since the decoding latency for SSC algorithm changes with the choice of codes, a systematic way to determine the decoding latency is derived. By following a simple equation, we can calculate the decoding latency for any given polar code easily. A formal DFG-based design flow for the SSC decoder architecture is developed also. Furthermore, in order to always achieve a lower decoding latency than previous works, a novel pre-computation SSC decoder architecture is also proposed. A (1024, 512) decoder example is employed to demonstrated the advantages of the proposed approaches.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Along with the emergence and rapid development of modern communication technologies, people's lives have been enormously changed by a list of new concepts such as streaming media, cloud storage, smart phone, and so on. The corresponding markets are huge and of great potential. Take the mobile phone market as an example. China Mobile is the world's largest individual mobile operator by subscribers [1]. Illustrated in Figure 1.1, China Mobile has over 859 million mobile phone subscribers by the end of the year 2010 [2]. By the end of 2009, more than 50 mobile operators have over 10 million subscribers each. And more than 150 operators had at least one million subscribers [2]. Not only the number of subscribers has expanded drastically, the data transmission rate of a single mobile phone has increased a lot. The data-optimized 4th-generation technologies such as the WiMAX standard [3] and the LTE standard [4] can achieve up to 10-fold speed improvements over 3G technologies [5]. And still researchers are now working towards the 5G systems [6], which would like to be implemented around the year 2020 [7]. Therefore, it becomes very challenging for modern communication systems to handle those increasing heavy tasks.

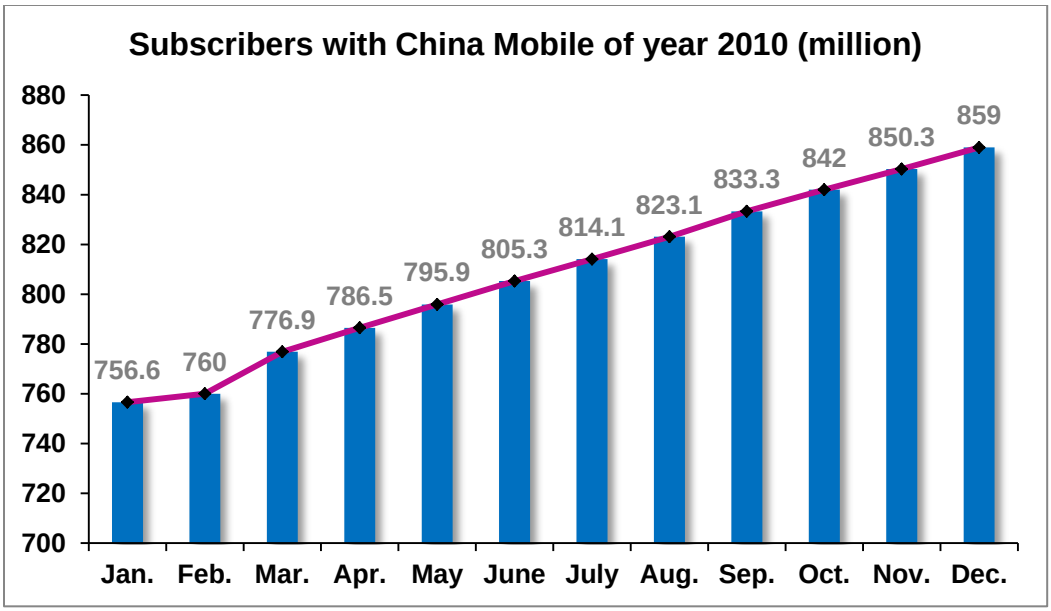


Figure 1.1 Subscribers with China Mobile of the year 2010 [2].

As illustrated in Figure 1.2, any modern communication system can be decomposed into five parts, which are the source, the transmitter, the noisy channel, the receiver, and the destination [8]. In order to protect the data transmitted over channels, we need to effectively suppress the influence of the noise. Therefore, channel codes are always of high necessity [9, 10]. Ever since the notation of channel capacity was defined by Claude E. Shannon during World War II [11], channel codes have experienced a rapid development [12]. From Gray codes [13], we had at the very beginning, nowadays we have much more choices: Hamming codes [14], Reed-Solomon (RS) codes [15], Bose-Chaudhuri-Hocquenghem (BCH) codes [16, 17], Turbo codes [18], fountain codes [19], and low-density parity-check (LDPC) codes [20]. The history of channel codes is actually the annals of modern communication systems' development. This reminds us of the emergence of Turbo codes, LDPC codes, and the iterative decoding methodology in 1990's. These channel coding techniques have promoted the growth of modern communication systems such as 10 Giga-bit/s Ethernet [21], Digital video broadcasting [22], Wi-Fi [23], and 3G wireless communications [5]. On the other hand, the always-

existing pressing need to develop “the next generation” communication systems drives the development of channel codes move forward. Now, we are at the new turn again. Here comes the first question: what error channel codes can we expect for the next generation modern communication systems?

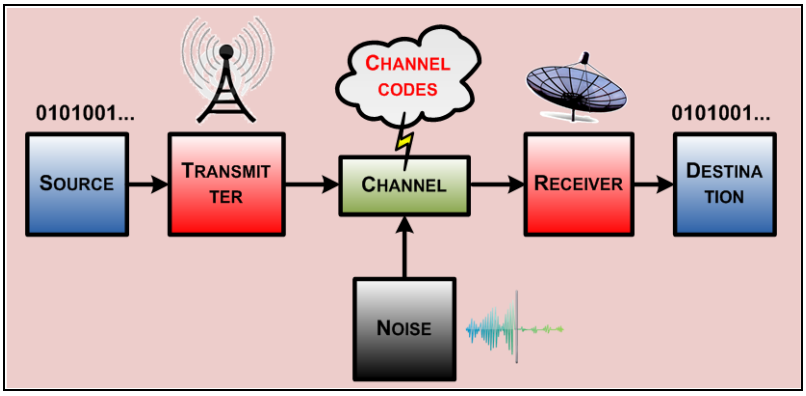


Figure 1.2 Simple block diagram of modern communication systems.

Aside from the issue of developing new channel codes, the efficient implementation of corresponding decoders is of equal importance [24]. For modern communication systems, to fulfill the long-distance high-quality information exchanges among a large number of people, the corresponding decoders for channel codes are required to be fast, portable, reliable, and safe [25]. This means that the channel decoder implementation should be low-latency, low-complexity, low-error rate, and wiretap-free. Therefore, the second question is: can we design low-complexity low-latency channel decoders to meet the requirements of modern communication systems?

In order to answer the first question, recently two kinds of error correction codes have been proposed by the coding society, which are non-binary LDPC codes [26, 27] and polar codes [28-34], respectively. Compared with their binary counterparts, non-binary LDPC codes are defined over finite field $\boldsymbol{GF}(q)$ with $q>2$. Previous literatures such as [26] have shown that non-binary LDPC codes can show better decoding performance over

their binary counterparts with proper encoding approaches and code lengths. This advantage makes the non-binary LDPC codes very attractive for real-time applications. However, the straightforward implementation of non-binary LDPC decoders results in the computation complexity of (q^2). This high hardware complexity makes it difficult to adopt non-binary LDPC codes for modern communication applications. Therefore, further methodologies which can reduce the complexity of non-binary LDPC decoders to an acceptable level are in need. For polar codes, now they are considered as the most favorable capacity-approaching channel codes due to the low encoding complexity and good secrecy [35]. But the disadvantage is also obvious. Since the successive cancellation (SC) polar decoders are only able to produce decoded bits in a serial manner, the corresponding decoding latency turns out to be 2(N-1) clock cycles, where N is the length of the codeword [36]. Considering N is always set to be greater than 2^{10} [28], the resulting latency becomes impractical. How to design low-latency low-complexity SC polar decoder is still challenging.

VLSI digital signal processing (DSP) design techniques can be widely applied to the implementation of different application scenarios [37]. We believe that with proper VLSI DSP design techniques, practical decoder designs for modern communication systems can be obtained. This thesis is devoted to developing design methodologies for feasible low-complexity low-latency channel decoders in modern communication systems, especially on non-binary LDPC decoders and SC polar decoders. Our contributions are listed in the next section as follows.

1.2 Summary of Contributions

1.2.1 Non-Binary QC-LDPC Decoders with Efficient Networks

As mentioned in Section 1.1, non-binary LDPC codes are of great interest due to their better performance over binary ones when the code length is moderate. However, the cost of decoder implementation for these non-binary LDPC codes is still very high [27]. Generally speaking, the hardware consumption for an LDPC decoder usually comes from two parts: the processing units, which include both the check node units (CNUs) and the variable node units (VNUs), and the switch networks connecting those processing units. Previous literatures [38-44] mainly focused on the low-complexity design of processing units, especially on the hardware-efficient implementation of CNUs. However, the methodologies on how to reduce the complexity of switch networks have not been well addressed yet.

We have proposed a low-complexity VLSI architecture for non-binary LDPC decoders [45]. It should be mentioned that the specific non-binary LDPC codes we are dealing with are called non-binary quasi-cyclic LDPC (QC-LDPC) codes [46-56]. Like their binary counterparts, non-binary QC-LDPC codes are hardware-friendly and can also achieve promising decoding performance. According to Figure 2 of [49], with 50 decoding iterations, the 64-ary (1260, 630) non-binary QC-LDPC code can attain 3.78 dB code gain over the (1260, 630, 631) shortened RS code.

The non-binary QC-LDPC codes introduced in [49] can be categorized into to families. The first one is called Class-I non-binary QC-LDPC codes or Class-I codes for short. Their algebraic construction is mainly based on cyclic subgroups of the multiplicative group of $\boldsymbol{GF}(q)$. The other family is named as Class-II non-binary QC-LDPC codes or Class-II codes for short. Their construction is similar to the first one but

based on additive subgroups of the finite field. We will show that by exploiting the intrinsic shifting and symmetry properties of non-binary QC-LDPC codes, significant reduction of memory size and routing complexity can be achieved. These unique features directly lead to two different network-efficient decoder architectures for Class-I and Class-II codes, respectively.

Comparison results with the state-of-the-art designs show that for the code example of the 64-ary (1260, 630) rate-0.5 Class-I code, the proposed scheme can save up to 70.6% hardware required by the switch network, which demonstrates the efficiency of the proposed design methodology. The proposed design for the 32-ary (992, 496) rate-0.5 Class-II code can achieve a 93.8% switch network complexity reduction compared with conventional approaches. Those comparison results have demonstrated that the proposed approaches are feasible and efficient.

Furthermore, we also try to reduce the hardware complexity of the CNUs. With the help of a generator for possible solution sequences, both forward and backward steps can be eliminated to offer processing convenience of the CNU blocks. Results show that the proposed 32-ary (992, 496) rate-0.5 Class-II decoder can achieve 4.47 Mb/s decoding throughput.

1.2.2 Low-Latency Successive Cancellation Polar Decoders

Polar codes have recently emerged as one of the most favorable capacity-achieving error correction codes due to their low encoding and decoding complexity [28]. Polar codes are constructed with a method called channel polarization to achieve the symmetric capacity of any given binary-input discrete memoryless channel (B-DMC). It has been reported that polar codes under list decoding with CRC are competitive with the best LDPC codes at lengths as short as $N = 2^{11}$ [57].

However, because of the large code length required by practical applications ($N \geq 2^{10}$), the few existing SC decoder implementations still suffer from not only high hardware cost but also long decoding latency. Therefore, SC polar decoders with less decoding latency are required by modern communication systems. In this thesis, a data-flow graph (DFG) for the SC decoder is derived. Based on the DFG analysis, a family of low-latency SC polar decoders is derived formally [36, 58].

Low-Latency Sequential SC Polar Decoder

According to the DFG analysis, a low-latency sequential SC polar decoder architecture is proposed to reduce the achievable minimum decoding latency. Pre-computation look-ahead techniques are employed to halve the latency. The *feedback part* is presented for the first time. Sub-structure sharing is used to design a *merged* processing element (PE) for higher hardware utilization.

TC Interleaved SC Polar Decoder

In order to meet throughput requirements for a diverse set of application scenarios, a systematic approach to construct different TC interleaved SC polar decoder architectures is also presented. Compared with the conventional N-bit tree SC decoder, the proposed TC interleaved architectures can achieve as high as (N-1) times speedup with only 50% decoding latency and ($N \cdot \log_2 N$)/2 *merged* PEs.

RC 2-Interleaved SC Polar Decoder

Another approach to meet the high-speed requirements for modern communication system applications is introducing RC interleaved processing. However, the design of an RC interleaved SC decoder is challenging due to the inherent serial decoding schedule. Straightforward RC interleaved designs usually introduce significant decoding latency

overhead. To this end, in this paper a low-latency low-complexity RC 2-interleaved SC decoder is proposed with the folding technique [59]. Compared with similar designs in previous literatures, the proposed design can achieve 4 times speed-up while only consumes similar hardware.

1.2.3 Simplified SC Polar Decoder Architecture

Although the low-latency SC polar decoder family could reduce the latency by 50%, we believe that we do better. Recently, a low-latency decoding scheme referred as the *simplified* successive cancellation (SSC) algorithm has been proposed for the decoding of polar codes [60]. It is claimed that significant latency reduction is achieved over a wide range of code rates. However, since this approach highly depends on the specific code it is dealing with, the corresponding latency is not easy to predict.

In this thesis, we present the first systematic approach to formally derive the SSC decoding latency for any given polar code [61]. The method to derive various SSC polar decoder architectures for any specific code is also presented. Moreover, it is shown that with the pre-computation technique, the decoding latency can be further reduced. Similarly, the latency-reduced SSC decoder's latency can also be calculated with a simple equation. Compared with the state-of-the-art SC decoder designs, the two SSC polar decoders can save up to 39.6% decoding latency with the same hardware cost.

1.3 Outline of the Thesis

The thesis is organized as follows. Chapter 2 gives a brief review of non-binary QC-LDPC codes. Based on the geometry properties of Class-II and Class-II codes, two different switch networks are proposed. A generator for possible solution sequences is also introduced to further reduce the complexity of CNUs.

Chapter 3 introduces the design of low-latency sequential SC polar decoder. The DFG of SC decoding process is proposed for the first time. Then the novel pre-computation look-ahead SC decoder architecture is described in detail.

Chapter 4 presents two kinds of SC polar decoder architectures towards high-speed applications. First, a systematic methodology for designing the TC interleaved SC polar decoders is described. Afterwards, the RC 2-interleaved SC polar decoder architecture is constructed based on the folding technique.

Chapter 5 proposes a method to determine the decoding latency of SSC algorithm. A design approach for corresponding SSC polar decoder is given also.

Finally, Chapter 6 summarized of the contribution of the entire thesis and provides future research directions.

Chapter 2

Non-Binary LDPC Decoders with Efficient Networks

In this chapter, we present two network-efficient decoder architectures for non-binary QC-LDPC codes [45]. Section 2.1 provides a brief introduction of non-binary LDPC codes and their sub-class, non-binary QC-LDPC codes. Section 2.2 briefly reviews prior works on non-binary LDPC decoder designs. Section 2.3 investigates the geometry properties of both Class-I and Class-II codes. Decoding schemes with different choices of layers are evaluated in Section 2.4. Section 2.5 presents message passing schedules via proposed networks and corresponding low complexity non-binary QC-LDPC decoder architectures. The hardware cost estimations and comparisons are presented in Section 2.6. Section 2.7 concludes this chapter finally.

2.1 Introduction

Rediscovered by MacKay [62], binary LDPC codes have shown near-Shannon limit performance [22, 63-66]. They have been extensively adopted in next-generation communication system standards. Recently, LDPC codes over $\boldsymbol{GF}(q)$ with $q>2$ are

reported to show even better decoding performance over the binary ones [26] when encoding approach and code length are proper. However, the introduced high decoding complexity is also significant.

To this end, a sub-class of non-binary LDPC codes is proposed in [46-55]. Non-binary QC-LDPC codes are architecture-aware and can achieve good performance. In [49], two algebraic construction methods based on *array dispersions* of matrices over non-binary subgroups are presented and referred as Class-I and Class-II, respectively.

2.1.1 Construction Method of Class-I Codes

Assume the *Galois* field ***GF***(q) has a primitive element α. In this case, elements within ***GF***(q) can be represented by powers of the primitive element α: $\alpha^{-\infty} = 0$, $\alpha^0 = 1$, α^1, …, α^{q-2}. Define $\mathbf{z}(\alpha^i) = (z_0, z_1, \ldots, z_{q-2})$ as a (q-1)-ary location-vector. Here the i-th component $z_i = \alpha^i$, and all the other ones are 0's. Specially, $\mathbf{z}(0)$ is defined as the all-zero (q-1)-tuple. The *circulant permutation matrix* (CPM) of δ is defined as $(\mathbf{z}(\delta), \mathbf{z}(\alpha\delta), \ldots, \mathbf{z}(\alpha^{q-2}\delta))^{\mathrm{T}}$, where δ can be any element in ***GF***(q). The construction method of Class-I codes is described as follows:

Construction of Class-I Non-Binary QC-LDPC Codes

1: Factorization: $q-1 = c \times n, \gcd(c, n) = 1$;

2: Element definition: $\beta = \alpha^c$, $\delta = \alpha^n$;

3: Subgroup expansion: $\begin{cases} G_1 = \{\beta^0 = 1, \beta, \ldots, \beta^{n-1}\}, \\ G_2 = \{\delta^0 = 1, \delta, \ldots, \delta^{c-1}\}; \end{cases}$

4: Matrix formation: $\begin{cases} \mathbf{W}^{(1)} = [\mathbf{W}_{i,j}^{(1)}]_{0 \le i < c, 0 \le j < c}, \\ \mathbf{W}_{i,j}^{(1)} = [\delta^{j-i}\beta^k - \beta^l]_{0 \le k < n, 0 \le l < n}; \end{cases}$

5: Substitution: Replace each entry of $\mathbf{W}^{(1)}$ by its CPM to get $\mathbf{A}^{(1)} = [\mathbf{A}_{i,j}^{(1)}]_{0 \le i,j < q}$;

6: Truncation: $\mathbf{H}^{(1)} = \mathbf{A}^{(1)}(\gamma, \rho) = [\mathbf{A}_{i,j}^{(1)}]_{0 \le i < \gamma, 0 \le j < \rho}$;

7: Output: $\mathbf{H}^{(1)}$.

2.1.2 Construction Method of Class-II Codes

Using the additive subgroups instead of the cyclic ones, we have the construction method of Class-II codes as follows.

Construction of Class-II Non-Binary QC-LDPC Codes

1: Factorization: $q = 2^m$, $c = 2^{m-t}$, and $n = 2^t$;

2: Elements definition: $\begin{cases} f_t' = \{\alpha^0, \alpha^1, \ldots, \alpha^{t-1}\}, \\ f_{m-t}'' = \{\alpha^t, \alpha^{t+1}, \ldots, \alpha^{m-1}\}; \end{cases}$

3: Subgroup expansion: $\begin{cases} F_t' = \{0, \beta_1, \ldots, \beta_{2^t-1}\} \text{ spanned by } f_t', \\ F_{m-t}'' = \{0, \delta_1, \ldots, \delta_{2^{m-t}-1}\} \text{ spanned by } f_{m-t}''; \end{cases}$

4: Matrix formation: $\begin{cases} \mathbf{W}^{(2)} = [\mathbf{W}_{i,j}^{(2)}]_{0 \le i,j < c}, \\ \mathbf{W}_{i,j}^{(2)} = [(\delta_i - \delta_j) + (\beta_k - \beta_l)]_{0 \le k,l < n}; \end{cases}$

5: Substitution: Replace each entry of $\mathbf{W}^{(2)}$ by its CPM to get $\mathbf{A}^{(2)} = [\mathbf{A}_{i,j}^{(2)}]_{0 \le i,j < q}$;

6: Truncation: $\mathbf{H}^{(2)} = \mathbf{A}^{(2)}(\gamma, \rho) = [\mathbf{A}_{i,j}^{(2)}]_{0 \le i < \gamma, 0 \le j < \rho}$;

7: Output: $\mathbf{H}^{(2)}$.

2.2 Prior Works on Non-Binary LDPC Decoders

Much research has been carried out on non-binary LDPC decoder designs. A brief review of previous works on non-binary LDPC decoding algorithms and decoder architectures is given as follows.

2.2.1 Decoding Algorithms for Non-Binary LDPC Codes

The *belief propagation* (BP) algorithm is the locally optimal, yet the most complex, iterative decoding algorithm of non-binary LDPC codes. Because the size of messages is *q*, the straightforward implementation of BP algorithm has the complexity of (q^2), which is very high for hardware designers. To this end, several revised decoding algorithms

have been proposed. The log-domain decoding scheme [67] is mathematically equivalent to the BP algorithm. It has shown advantages in both decoding complexity and numerical robustness. By employing a p-dimensional two-point fast Fourier transform (FFT), the computation complexity can be further reduced, where $p = \log_2 q$. However, the FFT algorithm is not suitable for the log domain. Then, a mixed-domain implementation has been presented in [68]. In this algorithm, FFT operation is carried out in real-domain. And the operations of CNU and VNU are carried out in log-domain. To implement the exponential and logarithm computations, the look-up table (LUT) is employed for the data conversion between log-domain and real-domain.

Unfortunately, for high-order field applications, all the above approaches are of limited interests. This is because the number of LUT accesses grows with a complexity of (qp) for a single message. To solve this problem, a complexity-reduced variant of the Min-Sum (MS) decoding, called Extended Min-Sum (EMS), was proposed in [27, 69, 70]. In this algorithm, the CNUs only deal with a *selective* part of the incoming messages. Moreover, some other low-complexity quasi-optimal iterative algorithms are proposed in [71]. The Min-Max algorithm is the most attractive one of them. It reduces the total number of operations with minimum decoding degradation.

2.2.2 Existing Non-Binary LDPC Decoder Architectures

Although in the past few years, the research on binary LDPC decoder design has experienced a significant growth [72-74], very few publications on non-binary LDPC decoder implementations have appeared. A straightforward implementation of the EMS decoding algorithm was proposed in [41]. This is the first implementation of a non-binary LDPC decoder with $q \geq 64$. [68] presented a mixed-domain non-binary LDPC decoder for small codes. But the decoding throughput is only 1 Mb/s, which is not enough for modern communication systems. In [42-44], several non-binary QC-LDPC decoder architectures

using semi-parallel processing scheme are proposed. Another two efficient non-binary LDPC decoders have been proposed by [38] and [39]. Moreover, a flexible decoder which is suitable for both binary and non-binary LDPC codes has been given in [40]. However, all those architectures suffer from high-complexity networks. This is because they use either a bi-directional network or two full-size switch networks for shuffling and reshuffling messages.

In this chapter, we present novel non-binary LDPC decoder architectures with both high network efficiency and low hardware complexity. The layered decoding algorithm is employed for the proposed decoders. By investigating the geometry properties of the corresponding **H** matrices, two kinds of *local switch networks* for VNUs are introduced for Class-I and Class-II codes, respectively. The proposed decoders are memory efficient, highly parallel, and have low routing complexity. Comparison results have shown that 70.6% switch network hardware can be reduced compared with the state-of-the-art design for decoding Class-I codes. And 93.8% can be reduced for Class-II decoders. Finally, by uncovering the actual identity of parity check equations for different layers, low-complexity CNUs are implemented.

2.3 Geometry Properties of Non-Binary QC-LDPC Codes

The parity-check matrix of non-binary QC-LDPC codes is simply composed of square sub-matrices. [42-44] have addressed some straightforward properties of them. However, for more efficient decoder architectures, more thorough investigations on the geometry properties of the check matrix **H** are required now.

2.3.1 Shifting Properties of Class-I Codes

According to the construction method of Class-I codes, the identity of $\mathbf{W}_{i,j}^{(1)}$ and its neighbor $\mathbf{W}_{(i-1)\bmod c,(j-1)\bmod c}^{(1)}$ can be verified by the following equation:

$$\begin{aligned}\mathbf{W}_{i,j}^{(1)} &= [\delta^{j-i}\beta^{k} - \beta^{l}] \\ &= [\delta^{((j\text{-}1)\bmod c)-((i\text{-}1)\bmod c)}\beta^{k} - \beta^{l}] \\ &= \mathbf{W}_{(i-1)\bmod c,(j-1)\bmod c}^{(1)}.\end{aligned} \tag{2-1}$$

Therefore, each row of the base matrix $\mathbf{W}^{(1)}$, except for the first row, is the 1-step right cyclic-shift of the row above it. The first row is the 1-step right cyclic-shift of the last row.

Moreover, further exploitation of sub-matrix $\mathbf{W}_{i,j}^{(1)}$ can show that similar permutation property holds at a lower level. Let $\mathbf{w}_{(i,j)(k,l)}^{(1)}$ be the entry of sub-matrix $\mathbf{W}_{i,j}^{(1)}$ located at the *k*-th row and *l*-th column. The property can be expressed as follows,

$$\begin{aligned}\mathbf{w}_{(i,j)(k,l)}^{(1)} &= \delta^{j-i}\beta^{k} - \beta^{l} \\ &= \beta(\delta^{j-i}\beta^{(k-1)\bmod n} - \beta^{(l-1)\bmod n}) \\ &= \beta\mathbf{w}_{(i,j)((k-1)\bmod n,(l-1)\bmod n)}^{(1)}.\end{aligned} \tag{2-2}$$

Note that except for the permutation operation, one multiplication with β is also required.

Combined with the definition of CPM, the geometry properties of Class-I codes can be summarized as follows:

Proposition 1 The Class-I non-binary QC-LDPC codes satisfy the shifting properties at three different levels:

1. The *i*-th row of the base matrix $\mathbf{W}^{(1)}$ is exactly the 1-step right cyclic-shift of the [(*i*-1)mod*c*]-th row. Therefore, $\mathbf{W}_{i,j}^{(1)} = \mathbf{W}_{(i-1)\bmod c,(j-1)\bmod c}^{(1)}$;
2. The *k*-th row of the sub-matrix $\mathbf{W}_{i,j}^{(1)}$ is exactly the 1-step right cyclic-shift of the [(*k*-1)mod*n*]-th row multiplied by β, that is, $\mathbf{w}_{(i,j)(k,l)}^{(1)} = \beta\mathbf{w}_{(i,j)(k-1,l-1)}^{(1)}$.;

3. The m-th row of the CPM corresponding to $\mathbf{w}_{(i,j)(k,l)}^{(1)}$ is the 1-step right cyclic-shift of the [(m-1)mod$n$]-th row multiplied by α, which is given by the definition of CPM.

Here is a simple example. ***GF***(2^4) has minimal polynomial $p(x)=1+x+x^4$ Following the construction steps, we can determine that $\beta=\alpha^c=\alpha^3$ and $\delta=\alpha^n=\alpha^5$. Therefore, the sub-groups can be obtained as follows:

$$\begin{cases} G_1=\{\beta^0=1,\beta,\ldots,\beta^{n-1}\}=\{\alpha^0=1,\alpha^3,\ldots,\alpha^{12}\}, \\ G_2=\{\delta^0=1,\delta,\ldots,\delta^{c-1}\}=\{\alpha^0=1,\alpha^5,\ldots,\alpha^{10}\}. \end{cases} \tag{2-3}$$

Accordingly, a 3×3 array of 5×5 sub-matrices can be constructed. **Proposition 1.1** and **1.3** can be verified. Now we only check the validation of **Proposition 1.2**. For the entry $\mathbf{W}_{0,0}^{(1)}$, we can find out that **Proposition 1.1** holds:

$$\mathbf{W}_{0,0}^{(1)}=\begin{bmatrix} 0 & \alpha^{14} & \alpha^{13} & \alpha^{7} & \alpha^{11} \\ \alpha^{14} & 0 & \alpha^{2} & \alpha & \alpha^{10} \\ \alpha^{13} & \alpha^{2} & 0 & \alpha^{5} & \alpha^{4} \\ \alpha^{7} & \alpha & \alpha^{5} & 0 & \alpha^{8} \\ \alpha^{11} & \alpha^{10} & \alpha^{4} & \alpha^{8} & 0 \end{bmatrix}. \tag{2-4}$$

Here we have $\mathbf{w}_{(0,0)(i,j)}^{(1)}=\alpha^3\mathbf{w}_{(0,0)(i-1,j-1)}^{(1)}=\beta\mathbf{w}_{(0,0)(i-1,j-1)}^{(1)}$.

2.3.2 Symmetry Properties of Class-II Codes

Compared with the Class-I codes, uncovering geometry properties of Class-II codes is non-trivial. This is because the construction method of Class-II codes does not specify the exact surjective function from the elements of subgroups F_t' and F_{m-t}'' to powers of α.

Given this degree of design freedom, we develop one specific surjective function to construct Class-II codes with symmetry properties. Without loss of generality, the construction of subgroup F_t' is illustrated here as an example.

Index Assignment of Surjective Function

1: Suppose $\beta_i = \sum_{m=m_0}^{m_p} \alpha^m, \beta_j = \sum_{n=n_0}^{n_q} \alpha^n$, with $0 \le m, n < t$;
2: if $p < q$ **then** $i < j$;
3: elseif $p > q$ **then** $i > j$;
4: else
5: **for** $l = 0, l++, l \le p$ **do**
6: **if** $m_l < n_l$ **then** $i < j$ **break**;
7: **else** $m_l > n_l$ **then** $i > j$ **break**;
8: **endfor**
9: endif

Table 2.1 denotes an example with t = 4. Here the minimal polynomial is given as $p(x) = 1 + x + x^4$. Based on the construction steps, the subgroup F_t' is spanned by the t elements within the set of $f_t' = \{\alpha^0, \alpha^1, \alpha^2, \alpha^3\}$. According to the proposed surjective function, we can check that for any element β_i within sub-group F_t', the following equation holds,

$$\beta_i + \beta_{(2^t-1)-i} = \beta_{2^t-1} \tag{2-5}$$

Similarly, for the other sub-group F_{m-t}'', the same symmetry property holds,

$$\delta_i + \delta_{2^{m-t}-1-i} = \delta_{2^{m-t}-1}. \tag{2-6}$$

Therefore, the symmetry property for each sub-matrix of $\mathbf{W}^{(2)}$ is given as follows,

$$\mathbf{W}_{i,j}^{(2)} = \mathbf{W}_{c-j-1,c-i-1}^{(2)}, \tag{2-7}$$

which indicates that each sub-matrix $\mathbf{W}_{i,j}^{(2)}$ is identical with its mirror about the anti-diagonal.

According to the construction method of non-binary QC-LDPC codes, every sub-matrix $\mathbf{W}_{i,j}^{(2)}$ is also self-symmetric about its own anti-diagonal:

$$\mathbf{w}_{(i,j)(k,l)}^{(2)} = \mathbf{w}_{(i,j)(n-l-1,n-k-1)}^{(2)}. \tag{2-8}$$

Table 2.1 Proposed surjective function example for F_t'.

Polynomial form				Power form	Element
0				0	β_0
1				1	β_1
	α			α	β_2
		α^2		α^2	β_3
			α^3	α^3	β_4
1	$+\alpha$			α^4	β_5
1		$+\alpha^2$		α^8	β_6
1			$+\alpha^3$	α^{14}	β_7
	α	$+\alpha^2$		α^5	β_8
	α		$+\alpha^3$	α^9	β_9
		α^2	$+\alpha^3$	α^6	β_{10}
1	$+\alpha$	$+\alpha^2$		α^{10}	β_{11}
1	$+\alpha$		$+\alpha^3$	α^7	β_{12}
1		$+\alpha^2$	$+\alpha^3$	α^{13}	β_{13}
	α	$+\alpha^2$	$+\alpha^3$	α^{11}	β_{14}
1	$+\alpha$	$+\alpha^2$	$+\alpha^3$	α^{12}	β_{15}

On the other hand, for Class-II codes, both the base matrix $\mathbf{W}^{(2)}$ and its sub-matrix $\mathbf{W}_{i,j}^{(2)}$ are self-symmetric about their diagonals. This can be verified according to the 4th step of the construction method of Class-II codes:

$$\begin{aligned}\mathbf{W}_{i,j}^{(2)} &= [(\delta_i - \delta_j) + (\beta_k - \beta_l)] \\ &= [(\delta_j - \delta_i) + (\beta_k - \beta_l)] \\ &= \mathbf{W}_{j,i}^{(2)},\end{aligned} \tag{2-9}$$

$$\begin{aligned}\mathbf{w}_{(i,j)(k,l)}^{(2)} &= (\delta_i - \delta_j) + (\beta_k - \beta_l) \\ &= (\delta_i - \delta_j) + (\beta_l - \beta_k) \\ &= \mathbf{w}_{(i,j)(l,k)}^{(2)}.\end{aligned} \tag{2-10}$$

In what follows, we will summarize all the proposed geometry properties of Class-II codes in **Proposition 2**:

Proposition 2 The Class-II non-binary QC-LDPC codes satisfy the geometry properties at three different levels:

1. The base matrix $\mathbf{W}^{(2)}$ is symmetric about its diagonal and anti-diagonal, i.e., $\mathbf{W}_{i,j}^{(2)} = \mathbf{W}_{j,i}^{(2)}$ and $\mathbf{W}_{i,j}^{(2)} = \mathbf{W}_{c-j-1,c-i-1}^{(2)}$;
2. The sub-matrix $\mathbf{W}_{i,j}^{(2)}$ is also symmetric about its diagonal and anti-diagonal, i.e., we have $\mathbf{w}_{(i,j)(k,l)}^{(2)} = \mathbf{w}_{(i,j)(n-l-1,n-k-1)}^{(2)}$ and $\mathbf{w}_{(i,j)(k,l)}^{(2)} = \mathbf{w}_{(i,j)(l,k)}^{(2)}$;
3. Each row of one CPM $\mathbf{w}_{(i,j)(k,l)}^{(2)}$ is the right cyclic-shift of the row above it multiplied by α and the first row is the right cyclic-shift of the last row multiplied by α.

Since **Proposition 2.1** and **2.2** are similar, without loss of generality, we only give an example of the latter one. Suppose t = 3 and the minimal polynomial $p(x) = 1 + x + x^3$. According to the proposed index assignment scheme, $\mathbf{W}_{0,0}^{(2)}$ can be constructed as follows:

$$\mathbf{W}_{0,0}^{(2)} = \begin{bmatrix} 0 & 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^6 & \alpha^4 & \alpha^5 \\ 1 & 0 & \alpha^3 & \alpha^6 & \alpha & \alpha^2 & \alpha^5 & \alpha^4 \\ \alpha & \alpha^3 & 0 & \alpha^4 & 1 & \alpha^5 & \alpha^2 & \alpha^6 \\ \alpha^2 & \alpha^6 & \alpha^4 & 0 & \alpha^5 & 1 & \alpha & \alpha^3 \\ \alpha^3 & \alpha & 1 & \alpha^5 & 0 & \alpha^4 & \alpha^6 & \alpha^2 \\ \alpha^6 & \alpha^2 & \alpha^5 & 1 & \alpha^4 & 0 & \alpha^3 & \alpha \\ \alpha^4 & \alpha^5 & \alpha^2 & \alpha & \alpha^6 & \alpha^3 & 0 & 1 \\ \alpha^5 & \alpha^4 & \alpha^6 & \alpha^3 & \alpha^2 & \alpha & 1 & 0 \end{bmatrix}. \quad (2\text{-}11)$$

It can be observed that $\mathbf{W}_{0,0}^{(2)}$ is symmetric about its diagonal and anti-diagonal. For **Proposition 2.3**, which is similar to **Proposition 2.1**, similar conclusion can be drawn.

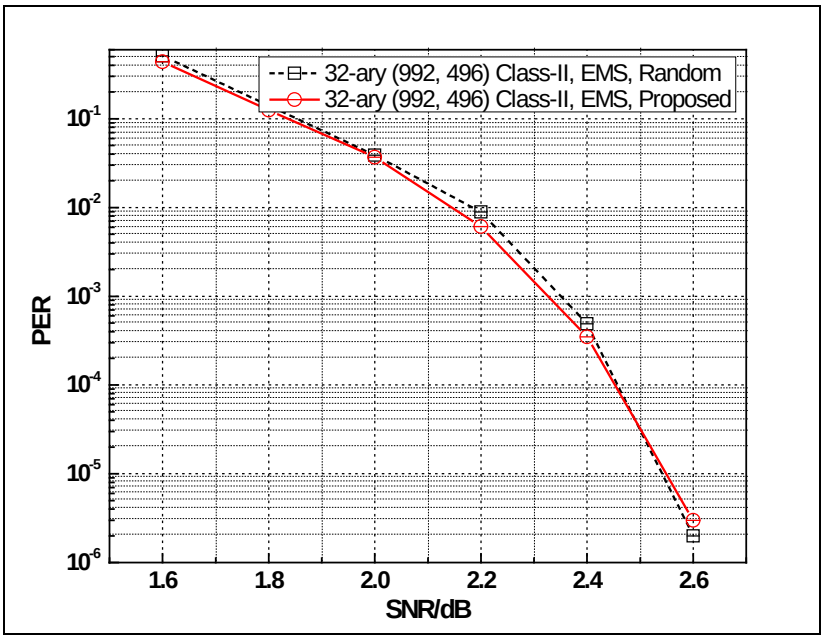


Figure 2.1 Performances of codes with different surjective functions.

We can also check that rows of the base matrix $\mathbf{W}^{(2)}$ constructed according to the proposed surjective function satisfy the α-multiplied row-column constraints [49]. Assume $m = 5$ and $t = 2$. With the code construction steps and surjective function, we can construct a 32-ary (992, 496) rate-0.5 Class-II code. In order to guarantee the decoding performance of codes generated with the proposed surjective function, another Class-II code with random surjective function is employed for comparison. Figure 2.1 illustrates the decoding performances of this code and its random counterpart over AWGN channel with BPSK signaling. The conventional EMS decoding algorithm with maximum iteration number of 10 is used for both codes.

Shown in Figure 2.1, it is observed that the packet error rate (PER) performance of Class-II code with the proposed surjective function is similar as that of the one with random scheme. Therefore, the introduced geometry properties do not affect the algebraic architecture and the decoding advantage of Class-II codes.

2.4 Layer Partition Choice for Layered Decoding Algorithm

2.4.1 Review of Layered Decoding Algorithm

The layered decoding approach [75-78] partition the check matrix **H** into *l* layers:

$$\mathbf{H}^{\mathrm{T}} = [\mathbf{H}_0^{\mathrm{T}} \ \mathbf{H}_1^{\mathrm{T}} \cdots \mathbf{H}_{l-1}^{\mathrm{T}}]. \tag{2-12}$$

Each layer is associated with one super-code C_i, and the original code C can be treated as the intersection of all *l* super-codes [76]:

$$C = C_0 \bigcap C_1 \bigcap \cdots \bigcap C_{l-1}. \tag{2-13}$$

It is required that the column weight of each layer is equal or less than 1.

The layered decoding message passing schedule with the Min-Max algorithm in the *k*-th iteration for layer *t* can be formulated as follows:

Layered Decoding for Min-Max Algorithm

1: $L_{cv}^{k,t}(a) = L_v^{k,(t-1)}(a) - R_{cv}^{(k-1),t}(a)$;

2: $R_{cv}^{k,t}(a) = \min_{\substack{(a_{v'})_{v' \in \mathcal{H}(c)\setminus(v)} \\ \in \mathcal{L}(c|a_v = a)}} \left(\max_{v' \in \mathcal{H}(c)\setminus(v)} L_{cv'}^{k,t}(a_{v'}) \right)$;

3: $L_v^{k,t}(a) = L_{cv}^{k,t}(a) + R_{cv}^{k,t}(a)$.

Here, $L_{cv}^{k,t}(a)$ is the *variable to check* message from layer *t* to the next layer during the *k*-th iteration which is associated with finite field element *a*. And $R_{cv}^{k,t}(a)$ is the *check to variable* message. The message $L_v^{k,t}(a)$ is the LLRs from layer *t* to the next layer during the *k*-th iteration. Define $\mathcal{H}(c)$ be the set of variable nodes participating in check node *c*, and $\mathcal{H}(c)\setminus(v)$ be the set excluded the variable node *v*. $\mathcal{L}(c \mid a_v = a)$ denotes the set of finite sequences which satisfy check node *c*, given the value of the variable node *v* equals *a*. As mentioned above, each layer carries out its own decoding process with both channel inputs and the *extrinsic* output of last layer. Because of this novel updating schedule, the layered decoding propagates much faster than the conventional ones such as the two-

phase message-passing (TPMP) decoding algorithm [26]. Therefore, compared with conventional message passing algorithms, layered decoding performance is better within the same number of decoding iterations.

2.4.2 Layer Partition and Related Decoding Performances

According to **Proposition 1** and **2**, nice algebraic construction enables both classes of non-binary QC-LDPC codes accommodated with the layered decoding algorithm. Inherently, their check matrix **H** can be split into layers. And each layer can naturally serve as a super LDPC code. Actually, we have two layer partition options listed as follows:

1. Choose each sub-block row of $\mathbf{W}_{i,j}^{(1)}$ or $\mathbf{W}_{i,j}^{(2)}$ as one layer, which consists of (q-1) rows. This option is defined as the *Layer-I* choice;
2. Choose each row of CPM within $\mathbf{w}_{(i,j)(k,l)}^{(1)}$ or $\mathbf{w}_{(i,j)(k,l)}^{(2)}$ as one layer, which consists of only one row. This option is defined as the *Layer-II* choice.

It can be observed that the constraint of at most 1 column weight within each layer is satisfied in both options. To demonstrate the advantages of the layered scheme for non-binary QC-LDPC codes, one decoding example is given as follows. For a 64-ary (1260, 630) rate-0.5 Class-I code, performances of three decoding approaches are compared in Figure 2.2. The maximum number of iterations is set to 10. Decoding performances of the conventional Min-Max algorithm and Min-Max algorithm with two different layer choices are illustrated. According to Figure 2.2, it can be seen that the layered decoding variations can attain more than 0.08 dB decoding gain than the conventional Min-Max algorithm. For the two different layer partition choices, the fewer rows in each layer, the better performance can be achieved. This is because compared with the *Layer-I* choice, more inter-layer *extrinsic* messages are utilized in each iteration of the *Layer-II* choice.

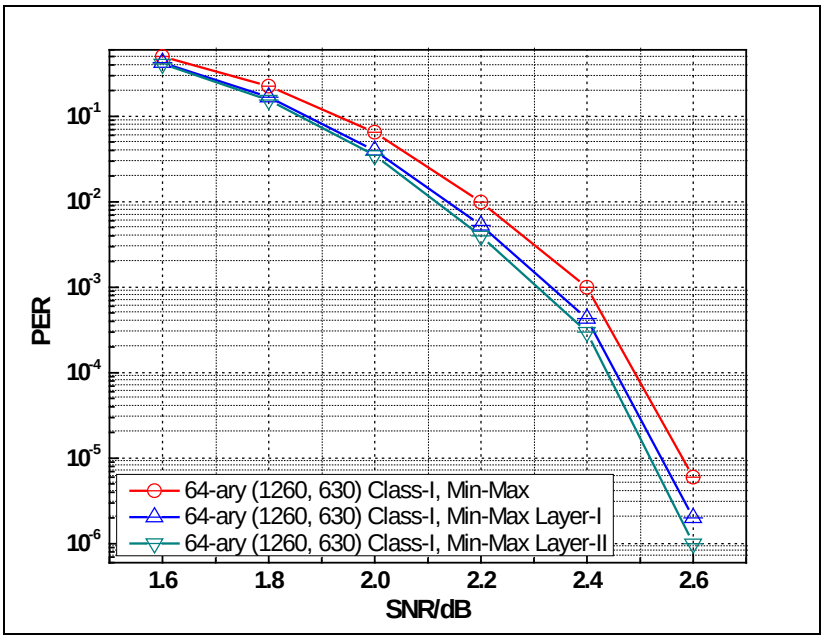


Figure 2.2 PER comparisons between different algorithms for a Class-I code.

2.5 A Reduced-Complexity Decoder Architecture

Although non-binary LDPC codes outperform their equivalent binary counterparts, the efficient implementation of non-binary LDPC decoders still remains challenging. Since the routing complexity and control memory size increase drastically with order of $\boldsymbol{GF}(q)$, how to implement low-complexity switch network connecting various processing nodes becomes a big problem. For the (u, v) non-binary QC-LDPC decoder, the straightforward implementation requires $\rho p(q\text{-}1)\gamma$-bit memory to store all the control signals. Here, $u = \rho(q\text{-}1)$ is the code length, and $u\text{-}v = \gamma(q\text{-}1)$ is the number of check bits. In the following part, we present a reduced-complexity decoder architecture based on the proposed geometry properties of non-binary QC-LDPC codes. The decoder is suitable for both serial and semi-parallel approaches. In addition, systematic algorithms to generate *local switch network* of VNUs are proposed for both classes of non-binary QC-LDPC codes.

2.5.1 Overall Architecture of Reduced-Complexity Decoder

The overall block diagram of the proposed semi-parallel decoder is shown in Figure 2.3. We assume the code-word length and layer height of **H** to be $\rho(q\text{-}1)$ and w, respectively. Illustrated in Figure 2.3, the proposed decoder architecture is composed of an array of $\rho(q\text{-}1)$ VNUs with a *local switch network*, a set of l CNUs, a *global shuffle network*, and a permutation/de-permutation block which implements multiplication/division operation in *Galois* fields. All l rows in each layer are updated in parallel, and a total of $\gamma(q\text{-}1)/w = l$ clock cycles are required per iteration.

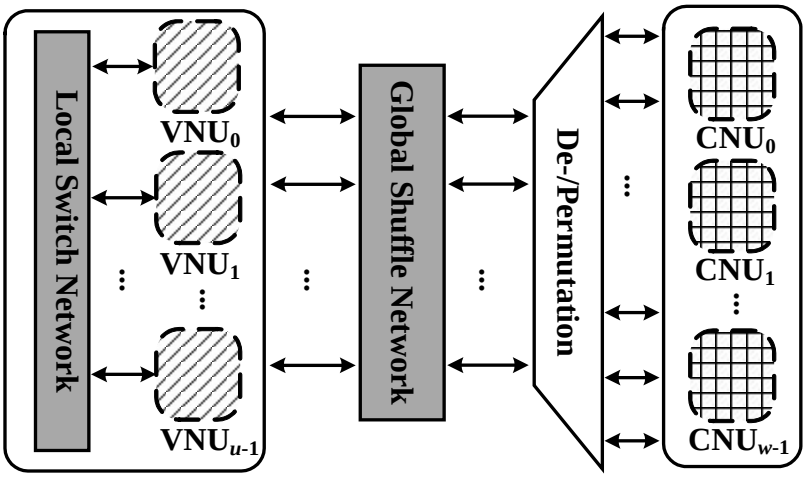


Figure 2.3 Block diagram of proposed layered non-binary QC-LDPC decoder.

The most significant point of the *global shuffle network* is, it stays unchangeable in the whole decoding process rather than being reconfigured for each layer. Once the parity check matrix is determined, no more reconfiguring operation is required. The switch network reconfiguration for the remaining layers can be eliminated by employing the *local switch network*.

2.5.2 Algorithm for Generating Local Switch Network of VNUs

Local Switch Network for Class-I Codes Cases

With the shifting properties of Class-I codes in **Proposition 1.2** and **1.3**, the *local switch network* now can be constructed based on a specific circulant permutation algorithm. It is clear that **Proposition 1.2** and **1.3** only differ in the value of the multiplicand (β or α). Without loss of generality, here we chose the *Layer-I* decoding scheme as an example.

Intuitively, we can split the *local switch network* shuffling operation between two layers into two steps. Mentioned previously, Eq. (2-2) can be employed to implement the partition of *Layer-I*. In *Step One*, the double mod operation $((k-1) \bmod n, (l-1) \bmod n)$ is carried out. The multiplication with β, which introduces another permutation at the level of CPM, is implemented in *Step Two*. The algorithm is given in detail:

Scheduling Algorithm for Local Shuffle Network - I

Step One

1: for all $0 \le i < \rho(q-1)$ **do**
2: Pass the *extrinsic* result of last layer from
3: VNU_i to $\text{VNU}_{[i-(q-1)] \bmod \rho(q-1)}$
4: endfor

Step Two

5: for all $0 \le i < \rho$ **do**
6: **for all** $0 \le j < q-1$ **do**
7: Pass the *extrinsic* result of last layer from
8: $\text{VNU}_{i(q-1)+j}$ to $\text{VNU}_{i(q-1)+(j-c) \bmod (q-1)}$
9: **endfor**
10: endfor

Proof. Assume the code length is $\rho(q\text{-}1)$. Therefore, we need the same number of VNUs. Indicated by Eq. (2-2), the interconnection among CNUs and VNUs of the last layer can be reused by the current layer. That is, the *extrinsic* result of the last layer can simply be

shuffled among existing VNUs before decoding the current layer. More specifically, since $\mathbf{w}^{(1)}_{(i,j)(k,l)}$ is associated with $\mathbf{w}^{(1)}_{(i,j)((k-1) \bmod n,(l-1) \bmod n)}$, and the row index modulation can be eliminated if the decoding process is carried out layer by layer, it is only required to assign the *extrinsic* result of VNU_i to $\mathrm{VNU}_{[i-(q-1)] \bmod \rho(q-1)}$. Also be aware that the jumping stride of *Step One* is q-1, which is exactly the size of CPM.

However, the two sub-matrices in Eq. (2-2) are actually not identical, because of the multiplication operation. Therefore, another permutation step named *Step Two* is required. According to the definition of CPM, the CPM of $\mathbf{w}^{(1)}_{(i,j)(k,l)}$ can be obtained as follows. Firstly, we have to right cyclic shift the CPM of $\mathbf{w}^{(1)}_{(i,j)((k-1) \bmod n,(l-1) \bmod n)}$ by $\log_\alpha \beta$ steps. Then, we multiply the shifted CPM with β. It is Based on the construction method, we know that,

$$\beta = \alpha^c \Rightarrow \log_\alpha \beta = c. \tag{2-14}$$

Therefore, in *Step Two* an inner permutation with jumping stride of c is carried out. In this way, we assign the *extrinsic* result of $\mathrm{VNU}_{i(q-1)+j}$ to $\mathrm{VNU}_{i(q-1)+(j-c) \bmod (q-1)}$. ■

Take the layered decoding of the 4-ary (9, 3) rate-⅓ Class-I code shown in Figure 2.4 as an example. The factorization parameters are given as c = 1, n = 3. Therefore, we have $\beta = \alpha^c = \alpha$ over $\boldsymbol{GF}(2^2)$. For instance, as shown in Figure 2.4, VNU_7 is connected with CNU_0 during the 1st layer decoding, and with CNU_1 during the 2nd layer decoding. On the other hand, according to the decoding scheduling, the *extrinsic* result of VNU_7 is first passed to VNU_4 (*Step One*), then to VNU_3 (*Step Two*) after the 1st layer decoding. It is similar for other VNUs. Also it is observed that rather than establishing a new switch network for the 2nd layer, the *extrinsic* results of the 1st layer can be efficiently shuffled perfectly with the help of the *local switch network*.

Considering both *Step One* and *Step Two* involve the circulant permutations, we can further simplify the scheduling algorithm by removing redundant shifting operations. The resulting new scheduling algorithm merges the former two steps into a single step. The proof is given as follows:

Proof. Each VNU's index can be rewritten in the form of $i(q\text{-}1)+j$, where $0 \le i < \rho$ and $0 \le j < q\text{-}1$. Therefore, every VNU can be represented with a new notation of (i, j). During *Step One*, whose jumping stride is q-1, the *extrinsic* result is transferred to the $((i\text{-}1) \bmod n, j)$ VNU. Thereafter, the message is shuffled to the $((i\text{-}1) \bmod n, (j\text{-}c) \bmod (q\text{-}1))$ VNU in *Step Two*. That is, only one step is required to pass the *extrinsic* result of last layer from $\mathrm{VNU}_{i(q-1)+j}$ to $\mathrm{VNU}_{[(i-1) \bmod n](q-1)+(j-c) \bmod (q-1)}$. ■

New Scheduling Algorithm for Local Shuffle Network - I

1: for all $0 \le i < \rho$ **do**
2: **for all** $0 \le j < q-1$ **do**
3: Pass the *extrinsic* result of last layer from
4: $\mathrm{VNU}_{i(q-1)+j}$ to $\mathrm{VNU}_{[(i-1) \bmod n](q-1)+(j-c) \bmod (q-1)}$
5: **endfor**
6: endfor

For ease of explanation, the schedule shown in Figure 2.4 is employed as an example again. The index of VNU_7 can be changed into the new form (2, 1). Using the new scheduling algorithm, we can easily find out that the destination index is (1, 0). Therefore, the *extrinsic* message is transferred from VNU_7 to VNU_3 ($1 \times 3 + 0 = 3$), which matches our previous analysis perfectly.

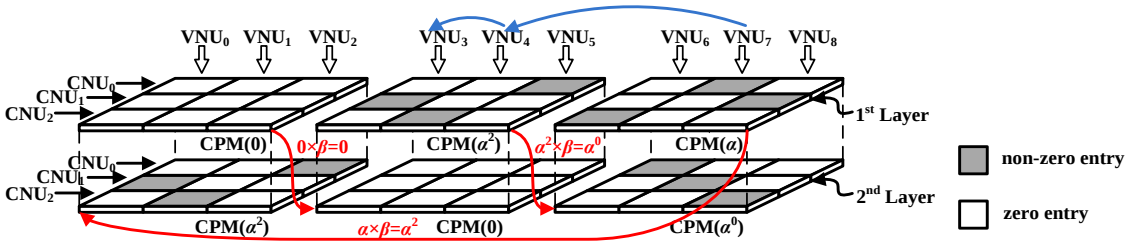


Figure 2.4 Layered decoding example of the 4-ary (9, 3) rate-⅓ Class-I code.

Local Switch Network for Class-II Codes Cases

Similarly, the *local switch network* for Class-II codes can be implemented based on the inherent symmetry properties in **Proposition 2.2** and **2.3**. At the same time, it is worth noting that, indicated by Eq. (2-7)-(2-10), we only need to take care of the symmetry execution rather than both symmetry and multiplication. The scheduling algorithm for Class-II codes can be derived as follows,

Scheduling Algorithm for Local Shuffle Network - II

1: for all $0 < v \le l,$ the beginning of decoding the vth layer **do**
2: **for all** $0 \le i < \rho$ **do**
3: **for all** $0 \le j < q-1$ **do**
4: Pass result of $\mathrm{VNU}_{(q-1)(\mathbf{index}_{v-1,i \bmod n}+n\lfloor i/n \rfloor)+j}$
5: to $\mathrm{VNU}_{(q-1)(\mathbf{index}_{v,i \bmod n}+n\lfloor i/n \rfloor)+j}$
6: **endfor**
7: **endfor**
8: endfor

The $\mathbf{INDEX}^{(n)}$ matrix is an $n \times n$ matrix defined as $\mathbf{INDEX}^{(n)} = [\mathbf{index}_{i,j}]_{0 \le i < n, 0 \le j < n}$. The entries of the first row of **INDEX** are determined by $\mathbf{index}_{0,j} = j$ as default. Other entries can be derived from the index assignment of *surjective function* and symmetry properties of **Proposition 2.2** and **2.3**. For instance, $\mathbf{INDEX}^{(4)}$ is given as follows,

$$\mathbf{INDEX}^{(4)} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}. \tag{2-15}$$

It is observed that the matrix $\mathbf{INDEX}^{(4)}$ is symmetric about both its diagonal and anti-diagonal. The entries on the diagonal are 0's, and the entries on the anti-diagonal are all 3's (= n-1). The last row (column) is the reverse-order version of the first row (column), and *vice versa*. The proof is given as follows,

Proof. According to Eq. (2-8) and Eq. (2-10), the sub-matrix $\mathbf{w}^{(2)}_{(i,j)(k,l)}$ is identical to both $\mathbf{w}^{(2)}_{(i,j)(n-l-1,n-k-1)}$ and $\mathbf{w}^{(2)}_{(i,j)(l,k)}$. Therefore, the interconnection among CNUs and VNUs of the last layer is exactly the same as that of the current layer, and can be reused afterwards. Indicated by **Proposition 2**, the index of destination VNU can be obtained by using symmetry properties. Since there is no permutation for the very beginning row, the first row of **INDEX** is set as an array of n elements from 0 to n-1. Each column is associated with one specific VNU during a iteration. Since the dimension of **INDEX** is n, the same mapping scheme based on **INDEX** is performed by every n VNUs. Therefore, a modulo operation on the VNU index is required.■

A simple example is employed to give a clear explanation. For the 4-ary (12, 6) rate-½ Class-II code illustrated in Figure 2.5, the factorization parameters can be obtained by choosing t = 1. Accordingly, $c = 2^{m-t} = 2$, and $n = 2^t = 2$ over $\boldsymbol{GF}(2^2)$. The subgroups are $F'_t = \{0, \beta_1\} = \{0,1\}$ and $F''_{m-t} = \{0, \delta_1\} = \{0, \alpha\}$. Therefore, the index matrix $\mathbf{INDEX}^{(4)}$ is given as follows,

$$\mathbf{INDEX}^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2-16}$$

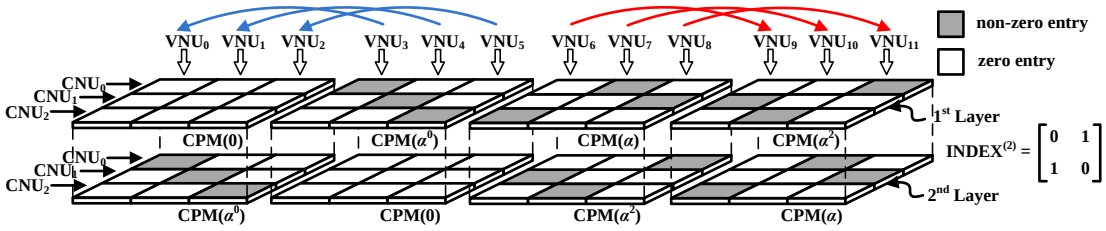


Figure 2.5 Layered decoding example of the 4-ary (12, 6) rate-½ Class-II code.

As illustrated in Figure 2.5, after the 1st layer decoding, the *extrinsic* message of VNU_0 is transferred according to the following direction:

$$\mathrm{VNU}_{3\times(\mathbf{index}_{0,0}+0)+0} \rightarrow \mathrm{VNU}_{3\times(\mathbf{index}_{1,0}+0)+0} = \mathrm{VNU}_3. \tag{2-17}$$

For the other VNUs, similar permutations can be obtained accordingly. The permutations are listed as follows,

$\mathrm{VNU}_0 \longrightarrow \mathrm{VNU}_3$,	$\mathrm{VNU}_6 \longrightarrow \mathrm{VNU}_9$,
$\mathrm{VNU}_1 \longrightarrow \mathrm{VNU}_4$,	$\mathrm{VNU}_7 \longrightarrow \mathrm{VNU}_{10}$,
$\mathrm{VNU}_2 \longrightarrow \mathrm{VNU}_5$,	$\mathrm{VNU}_8 \longrightarrow \mathrm{VNU}_{11}$,
$\mathrm{VNU}_3 \longrightarrow \mathrm{VNU}_0$,	$\mathrm{VNU}_9 \longrightarrow \mathrm{VNU}_6$,
$\mathrm{VNU}_4 \longrightarrow \mathrm{VNU}_1$,	$\mathrm{VNU}_{10} \longrightarrow \mathrm{VNU}_7$,
$\mathrm{VNU}_5 \longrightarrow \mathrm{VNU}_2$,	$\mathrm{VNU}_{11} \longrightarrow \mathrm{VNU}_8$.

2.5.3 Architectures of VNUs' Local Switch Network

Local Switch Network for Class-I Codes

It can be observed that the inter-layer message shuffle scheduling is irrelevant of the current layer index. It means, no matter what number *i* is, the *extrinsic* message transfer between the *i*-th layer and the (*i*+1)-th layer is exactly the same. Therefore, it can be implemented with fixed interconnections. Before further decoding steps are carried out, the intermediate results of VNUs are re-directed via the *local switch network*.

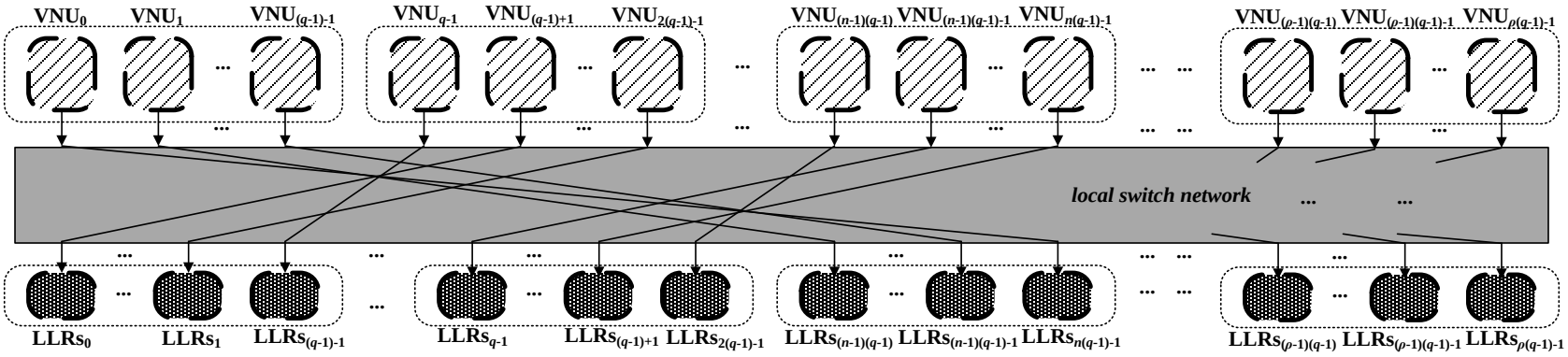


Figure 2.6 Local switch network of Class-I codes case.

The architecture of the *local switch network* for Class-I codes is illustrated in Figure 2.6. A total of $\rho(q\text{-}1)$ VNUs are employed and shuffling details of the first $n(q\text{-}1)$ VNUs are given. Rather than store all *extrinsic* results immediately, we only need to store the shuffled LLRs output by the *local switch network*, which can be directly used by other elements. Similar scheme is also employed by VNUs with indices from $n(q\text{-}1)$ to $\rho(q\text{-}1)$.

Local Switch Network for Class-II Codes

Different from Class-I codes, the inter-layer message shuffle scheduling for Class-II codes varies with the layer index. Nevertheless, the $\mathbf{INDEX}^{(n)}$ matrix can be used to design an efficient switch network. Since the output order is actually a permutation group of the input order, to provide the congestion-free communication from VNUs to LLRs, one-way *Benes* network is employed by Class-II decoders. Unlike previous approaches, the control signals for the re-configuring network become much simpler. For ease of analysis, we assume that the code length is $\rho(q\text{-}1)$, and ρ is power of 2. Therefore, a total of $\rho(q\text{-}1)$ VNUs and a $\rho q \times \rho q$ *Benes* network are required. In conventional designs, such network has $2\log_2(\rho q)\text{-}1$ stages and $\rho q[\log_2(\rho q)\text{-}1/2]$ 2×2 crossbar switches. The control bits and control complexity are $\rho q[\log_2(\rho q)\text{-}1/2]$ and $\mathcal{O}(2\log_2(\rho q)\text{-}1)$, respectively.

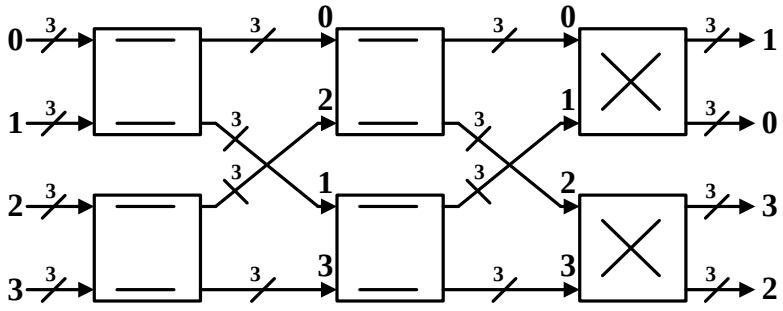


Figure 2.7 Local switch network of the Class-II code defined by Eq. (2-15).

According to the proposed algorithm, the size of *Benes* network has been reduced to $\rho \times \rho$. Only $2\log_2\rho\text{-}1$ stages and $\rho(\log_2\rho\text{-}1/2)$ 2×2 crossbar switches are needed. The

control part has $\rho(\log_2\rho\text{-}1/2)$ bits. Its complexity is $(2\log_2\rho\text{-}1)$. In addition, the control bits can be acquired by pre-computation with the aid of $\mathbf{INDEX}^{(n)}$ matrix easily. For instance, the *local switch network* for the example in Figure 2.7 is given as above.

We group VNUs with indices from i to i+2 and mark them with i, where i = 0, 1, 2, and 3. Compared with conventional approaches, we succeed in taking the symmetry properties into account. In the proposed designs, the main *Benes* network, the control circuits, as well as the routing complexity are much simpler. Here, only 4 2×2 crossbar switches are needed. It is clear that, the greater the parameter q is, the more hardware reduction can be expected. When the value of ρ is not a power of 2, similar conflict-free reconfigurable *Clos* network can be employed also. Please refer to [73, 79, 80] for more details. To sum up, no matter what network is employed, the whole size can be reduced. The low complexity advantage of the proposed method remains attractive.

2.5.4 Hardware Architectures of VNU Block

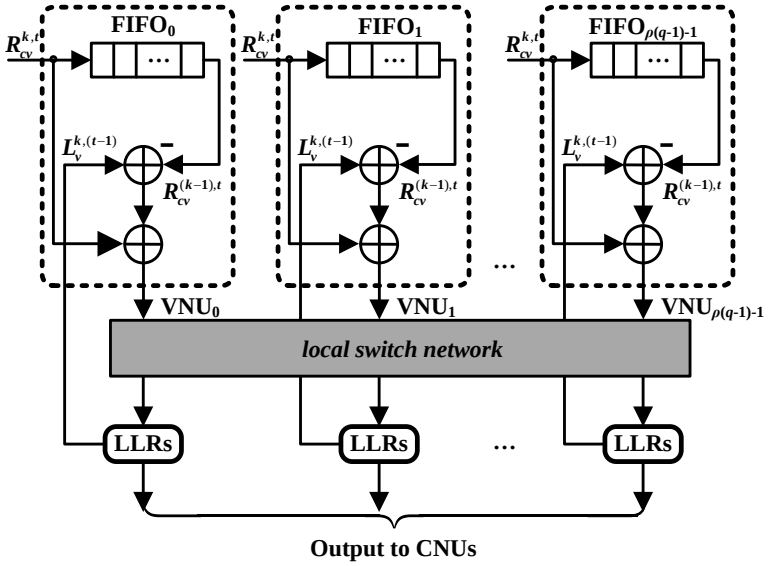


Figure 2.8 Hardware implementation of the variable node unit (VNU).

Figure 2.8 illustrates the straightforward architecture of a VNU, which is based on the layered Min-Max decoding algorithm [75]. For each VNU, there is a FIFO employed. Its depth is exactly the number of layers given by the partition scheme. In this manner, the *intrinsic* messages for layer *t* during the *k*-th iteration can be correctly popped out for layer *t* during the (*k*+1)-th iteration. Furthermore, based on the layered decoding scheme, in each VNU a cascade of two adders are employed to update the *variable to check* messages layer by layer. The output messages are then shuffled by the *local switch network*, which will be slightly different from Class-I codes to Class-II codes, and stored in the correctly arranged registers for the next layer decoding.

2.5.5 Hardware Architectures of CNU Block

The function of CNU block is to calculate the intrinsic message $R_{cv}(a)$, which is the probability of check *c* being satisfied if the value of variable *v* (denoted as a_v) has been determined as *a* and the value of any other variable *v'* (denoted as $a_{v'}$) has an independent distribution given by $L_{cv'}(a)$. Here, variable *v'* could be any variable connected with check *c* rather than variable *v*. In binary cases, the possible choices for *a* are 1 or 0. Therefore, whether the *c*-th check equation is satisfied or not can be easily determined by checking the parity of 1's in the *c*-th row of check matrix **H**. However, for non-binary LDPC codes, provided the a_v is fixed at *a*, possible combinations which can make the check equation true are not unique. In order to compute the value of $R_{cv}(a)$, we need to traverse all the possible solution sequences for the equation below without missing a single one, given a_v equals *a*.

$$h_{cv}a + \sum_{v' \in \mathcal{H}(c) \setminus (v)} h_{cv'}a_{v'} = 0. \tag{2-18}$$

Referred as $\mathcal{L}(c \mid a_v = a)$, the set which contains all the possible sequences can be found with an efficient recursive algorithm called forward-backward algorithm [26]. By making the use of partial sums, this algorithm computes sum of the items prior to $h_{cv}a$

(denoted as σ_{cv}) and sum of items posterior to $h_{cv}a$ (denoted as ρ_{cv}) in forward and backward direction, respectively. If and only if the following equation holds, the check c is satisfied.

$$\sigma_{cv} + \rho_{cv} = h_{cv}a. \tag{2-19}$$

Though claimed to be efficient, the computation complexity of the forward-backward approach is (q^2). Mentioned previously, the CNU operation will become extraordinarily computation intensive when the order of finite field q increases.

Check Equation Solution Sequences

Suppose the set $\mathcal{L}(c \mid a_v = a)$ gives all the solution sequences for check c. That is, for any sequence in $\mathcal{L}$, Eq. (2-18) holds. Two conclusions are stated as follows.

1. According to the commutativity of addition operation, any permutation of addends in Eq. (2-18) will not affect the equality;
2. Also, if h_{cv} and $h_{cv'}$ are changed with $\alpha^i h_{cv}$ and $\alpha^i h_{cv'}$ respectively, Eq. (2-18) will still hold. Here, α is the primitive element of corresponding finite field ***GF***(q) and i can be any integer.

According to **Proposition 1** and **2**, it is known that for *Layer-I* partition scheme, each layer is either the cyclic shift of previous row multiplied by β (Class-I codes) or the symmetric version of previous row (Class-II codes). Also for *Layer-II* partition scheme, no matter which class of non-binary QC-LDPC codes it belongs to, all q-1 entries of each row are exactly 1-step right cyclic-shift of those in the above row multiplied by α. Therefore, no matter what class the code is, no matter what layer partition is chosen, once the set $\mathcal{L}(c \mid a_v = a)$ which gives all the possible finite solution sequences for check c is determined for the first layer, it will stay unchangeable for the remaining layers. As a result, for the decoding process of these layers, the conventional forward-backward step

can be eliminated thereafter. Since the set of solution sequences is actually the same for all CNUs, it is possible to pre-determine set $\mathcal{L}(c \mid a_v = a)$ before the CNU decoding process is carried out.

Generation for Finite Solution Sequences

In order to figure out the set of all finite solution sequences, a specific calculator has been designed to search all the possible choices that satisfy the check equation. Illustrated in Figure 2.9, a total of $q^{\rho-1}$ possible combinations are associated with the ρ-1 inputs of the calculator. All inputs are multiplied with the non-zero entries lying in the first row of check matrix **H** except for the last one (denoted as h_{00}, h_{01}, h_{02}, …, $h_{0(\rho-2)}$), respectively.

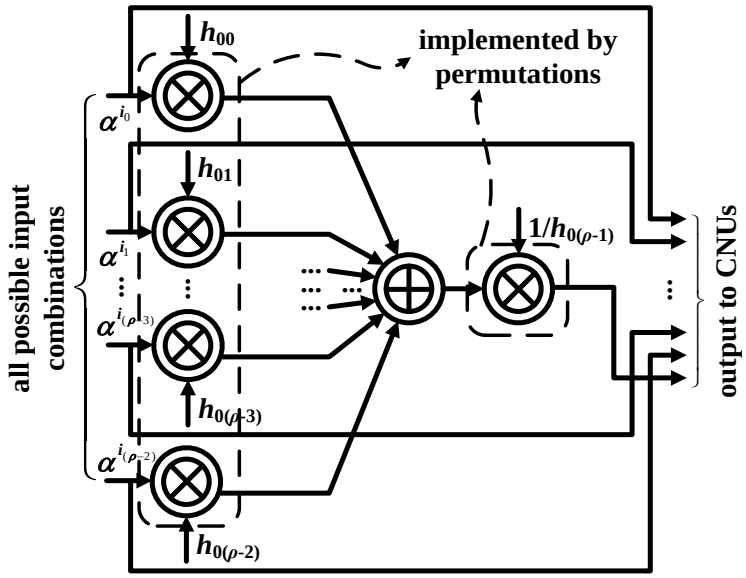


Figure 2.9 Internal structure of generator for possible solution sequences.

The intermediate results are summed up then. For the *selective* Min-Max decoding algorithm employed in this paper, the computation complexity can be further reduced with the decrease of parameter n_m. This is because both CNUs and VNUs only deal with n_m values rather than the values corresponding to all the elements in the finite field. An

additional, division step with parameter $h_{0(\rho\text{-}1)}$ is necessary to obtain the last required element in the sequence. Suppose $h_{0(\rho-1)} = \alpha^i$, in finite field $\boldsymbol{GF}(q)$ it always holds that $\alpha^i \cdot \alpha^{(q-1)-i} = 1$, the operation of division by $h_{0(\rho\text{-}1)}$ can then be simply implemented by multiplication with its reciprocal $h_{0(\rho-1)}^{-1} = \alpha^{(q-1)-i}$. Furthermore, from the permutation representation of finite field, all multiplication operations can be easily implemented with permutations. After all input combinations have been simulated, all results as well as their LLRs are output to CNUs for further use.

CNU Block Architecture Design

Usually, CNU blocks are the most complicated parts of the entire decoder. However, with the nice geometry properties of non-binary QC-LDPC codes, the architecture of the CNU block can be simplified. With the help of the proposed generator, all possible finite solution combinations for a specific check node can be determined in advance. During the check node process, we only need to compare the LLRs. In the Min-Max decoding algorithm, the CNU process is aim to find the minimum value among infinite norms of all possible check equation combinations given the corresponding variable node equals to *a*. Here, *a* can be any element in a given finite field $\boldsymbol{GF}(q)$.

We can split the check node process into two steps. The first step is to find the minimum value of all infinite norms if the variable node is fixed at *a*. In the second step, we sort all possible *a*'s according to their likelihood values and pick up n_m of them, which have the most significant probabilities (LLRs). For ease of clarity, suppose the check node degree is d_c. We use the notation of $\mathcal{S} \Leftarrow \{R_{cv}^{k,t}(a)\}$ to denote the insertion of the LLR value $R_{cv}^{k,t}(a)$ into a pre-sorted LLR sequence. The CNU processing algorithm is presented as follows,

Decoding Processing Algorithm for CNU Block

1: for all $0 \le i < q-1$ **or** $i = -\infty$ **do**

2: $R_{cv}^{k,t}(\alpha^i) = 0, \mathcal{S} = []$

3: **for all** $(a_{v'})_{v' \in \mathcal{H}(c) \setminus (v)} \in \mathcal{L}(c \mid a_v = \alpha^i)$ **do**

4: $R_{cv}^{k,t}(\alpha^i) = \min_{\substack{(a_{v'})_{v' \in \mathcal{H}(c) \setminus (v)} \\ \in \mathcal{L}(c \mid a_v = a)}} \left(\max_{v' \in \mathcal{H}(c) \setminus (v)} L_{cv'}^{k,t}(a_{v'}) \right)$

5: **endfor**

6: $\mathcal{S} \Leftarrow \{R_{cv}^{k,t}(\alpha^i)\}$

7: endfor

The data sorter can be implemented with right-shift cells [81]. A total of n_m processing elements (PEs) are required by the sorter. Each PE is composed of two cells: the right-shift cell and the compare cell. The former one is in charge of the date storage and right-shift operation. The latter one realizes the comparison and generates the control signals for the right-shift cell. Here, r_i is the right-shift enable signal, p_i denotes the pre-sorted data, and c_i represents the result of comparison. Finally, a total of n_m intrinsic messages with the most significant magnitudes will be output.

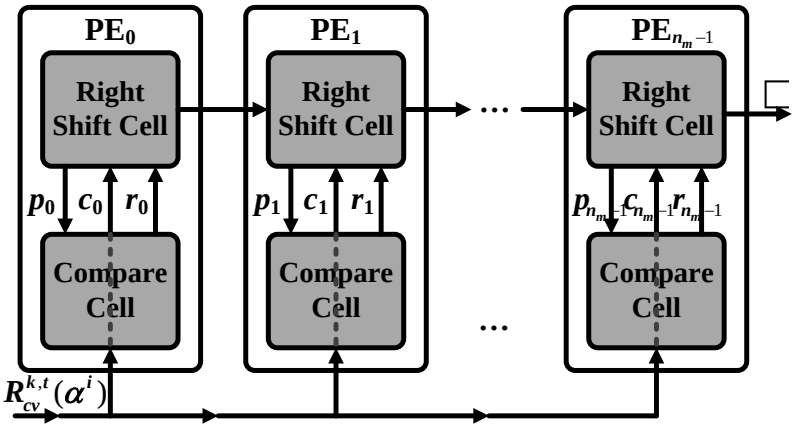
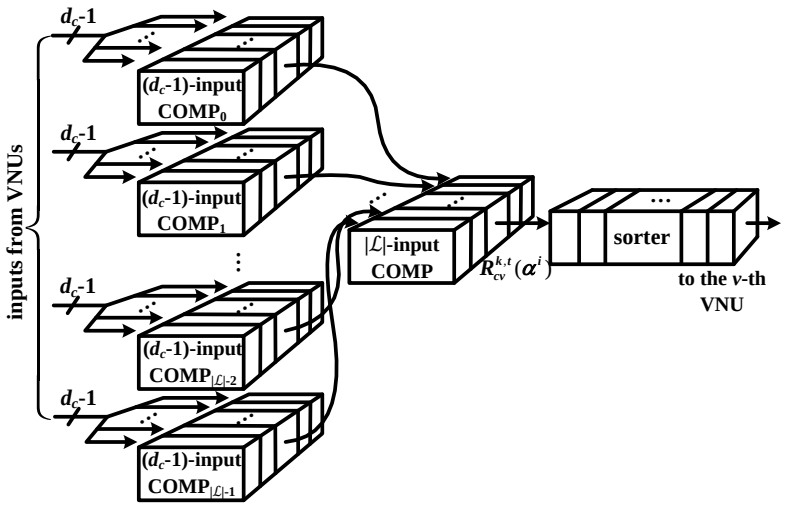


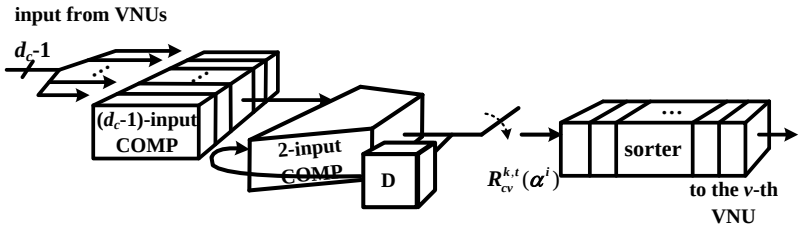
Figure 2.10 Data sorter structure with length of n_m.

Therefore, in order to implement the decoding processing algorithm of CNU, two sub-blocks are needed. The compare sub-block consists of a total of $|\mathcal{L}|$ (d_c-1)-input

comparators and one ($|\mathcal{L}|$-1)-input comparator, where $|\mathcal{L}|$ is the cardinality of the set $\mathcal{L}(c \mid a_v = \alpha^i)$. The inputs of this sorter can be read off from the LUT of the finite solution sequence generator. The input operation is achieved by a simple control unit selecting α^j. The result of $R_{cv}^{k,t}(\alpha^i)$ is output to the sorter, which will pick up the n_m most significant ones, realign them in decreasing order, and output them to the v-th VNU for further operations.



(a) Concurrent version of CNU architecture.



(b) Simplified version of CNU architecture.

Figure 2.11 Proposed CNU block architecture employing data sorter.

The corresponding hardware architecture of the CNU block is illustrated in Figure 2.11 (a) as above. Moreover, the architecture in Figure 2.11 (a) can be further simplified

as shown in Figure 2.11 (b), where only one (d_c-1)-input comparator is employed. For both structures, given the value of α^i, all the $|\mathcal{L}|$ finite solution sequences are generated by the module illustrated in Figure 2.9. In Figure 2.11 (a), all possible solutions are processed in a parallel manner to select the required one. In the outer loop, only n_m intrinsic messages with the most significant magnitudes are chosen by the data sorter, which is shown in Figure 2.10. In Figure 2.11 (b), the switch keeps open during the first step. All the solution sequences are input in serial. Here we employ a 2-input comparator with one delay element to generate LLR value of $R_{cv}^{k,t}(\alpha^i)$. In the outer loop, the switch is closed and the sorter will output the n_m most significant intrinsic messages.

2.6 Comparison with Prior Decoder Designs

In this section, we present the hardware complexity of the proposed two classes of non-binary QC-LDPC decoder architectures and compare them with previous designs. The decoding throughput of the proposed designs is also estimated at the end of this section. Without loss of generality, it is assumed that the proposed (u, v) decoders employ the *Layer-I* partition scheme. Therefore, a total of $\rho(q\text{-}1)$ VNUs and q-1 CNUs are required. It is assumed that a (b_q, b_f) uniform quantization scheme is employed, in which b_q bits are used for the entire message and b_f bits are used for the fractional part. Table 2.2 lists the comparison results of the decoder architectures for Class-I and Class-II codes, respectively. For each decoder, both versions with specific and configurable shuffle networks have been taken into account. The former version is designed for certain specific code, whereas the latter version can be employed for decoding codes from the same category. Design details are depicted in terms of cost of processing units, complexity of shuffle network, and consumption of finite solution sequence generator.

Table 2.2 Comparisons for different non-binary QC-LDPC code decoders.

Different designs		Class-I decoder		Class-II decoder		[42][a]		[44][a]	[43][b]	[38][b]
		Specific	Flexible	Specific	Flexible					
Hardware cost of processing units										
Number of CNUs		q-1		q-1		3(q-1)		q-1	3(q-1)	$<\gamma(q-1)$
Each CNU	Registers	$b_q(n_m+1)$		$b_q(n_m+1)$		REGs	9300	2635	9300	37696
	Comp.s	n_m+d_c-1		n_m+d_c-1		XORs	20693	11284	20693	40036
Number of VNUs		$\rho(q-1)$		$\rho(q-1)$		$\rho(q-1)$		$\rho(q-1)$	$\rho(q-1)$	$<\rho(q-1)$
Each VNU	Registers	$b_q(\gamma+n_m)$		$b_q(\gamma+n_m)$		–		–	–	$b_q(\gamma+n_m)$
	Adders	2		2						2
Total units	Registers	$(q-1)b_q[\gamma\rho+(\rho+1)n_m+1]$		$(q-1)b_q[\gamma\rho+(\rho+1)n_m+1]$		REGs	39990	35810	39990	3639648
	Comp.s	$(q-1)(n_m+d_c-1)$		$(q-1)(n_m+d_c-1)$		XORs	75469	79388	75460	3723348
	Adders	$2\rho(q-1)$		$2\rho(q-1)$		MEMs	1181410	451760	935110	206×10^6
Complexity of shuffle network										
Global network	Wires	$b_q n_m(q-1)d_c$		$b_q n_m(q-1)d_c$		$3b_q n_m(q-1)d_c$		$3b_q n_m(q-1)d_c$	$3b_q n_m(q-1)d_c$	$\gamma b_q(q-1)d_c$
	De-MUX's	0	$(q-1)\rho$	0	$(q-1)\rho$	$3(q-1)\rho$		$3(q-1)\rho$	$3(q-1)\rho$	$\gamma(q-1)\rho$
	LUT bits	0	$p(q-1)\rho$	0	$p(q-1)\rho$	$p(q-1)[\rho+\gamma(\gamma-1)/2]$		$p(q-1)(3\rho+\gamma-2)$	$p(q-1)(3\rho+\gamma-2)$	$\rho p(q-1)\gamma$
Local network	Wires	$b_q(q-1)\gamma$		$b_q(q-1)\gamma$		–		–	–	–
	Crossbars	0		$\rho(\log_2\rho-1/2)$		–		–	–	–
	LUT bits	0		$(\gamma\rho\log_2\rho)/2$		–		–	–	–
Consumption of finite solution sequence generator										
LUT bits		pd_c		pd_c		–		–	–	–
Galois adders		d_c-2		d_c-2		–		–	–	–

[a]Both designs are for a 32-ary (837, 726) rate-0.85 Class-I code.
[b]Both designs are for a 32-ary (744, 653) rate-0.875 non-binary LDPC code.

For different versions of decoders, the architecture of CNU stays the same. Each CNU requires $b_q n_m$ registers as well as n_m+d_c-1 2-input comparators, of which $b_q n_m$ registers and n_m 2-input comparators are required by the shift-register sorter. For the VNU, besides $b_q(\gamma+n_m)$ registers for the FIFO, another 2 adders are also required. The *global shuffle network* connects VNUs with CNUs. Once established, this network would not change throughout the entire decoding process. Therefore, $b_q(q\text{-}1)d_c$ wires are sufficient. Otherwise, for additional flexibility consideration, $b_q(q\text{-}1)\rho$ bits LUT and $(q\text{-}1)\rho$ de-MUX's are needed. Because of the different algebraic construction schemes, the *local switch network* varies from Class-I decoder to Class-II decoder. For Class-I decoder, it is only required to connect all VNUs end to end with $b_q(q\text{-}1)\rho$ wires, no matter whether the design is specific or configurable. However, for Class-II decoder, another LUT containing $\gamma\times\rho$ elements and related crossbar switches are required. Each element can be represented with $\log_2\rho$ quantization bits. Since the LUT is symmetric with respected to both its diagonal and anti-diagonal as indicated by Eq. (2-15), the memory size can be further reduced by half. Therefore, only $(\gamma\rho\log_2\rho)/2$ bits LUT is required. For the finite solution sequence generator, all multiplication operations are implemented with finite-step permutation, which requires pd_c bits of LUT. In addition, a total of d_c-2 *Galois* field adders are required as well.

In order to demonstrate the advantages of the proposed designs, comparisons between the proposed works and state-of-the-art designs [38, 42-44] are carried out. According to the comparison results in Table 2.2, it is observed that compared with the state-of-the-art decoder designs, the proposed designs can greatly reduce the hardware complexity by making use of the geometry properties of non-binary QC-LDPC codes. Based on the design scheme of [38, 42-44], the programming information of shuffle network should be pre-stored into ROM to guarantee the proper execution of the decoding. Since ROM is non-volatile, it is not convenient to incorporate flexibility into corresponding decoder

designs. Also, the decoder proposed in [42-44] are suitable for Class-I codes only. For code choices such as Class-II codes, a fully constructed global network, such as the one proposed in [38], is required by those decoders. On the other hand, the design approach proposed in this dissertation can be employed for both Class-I and Class-II codes.

For the proposed Class-I decoder architecture, only fixed wires are necessary. It means that, by introducing the *local switch network*, all memories can be eliminated. If we can use more de-MUX's and LUT, the decoder will be capable of dealing with a class of codes rather than a specific one. Even in this case, the proposed decoder architecture still shows advantages in hardware complexity. Now, we employ the 64-ary (1260, 630) rate-0.5 Class-I code as an example for a better demonstration. Here, we know $q = 64$, $\rho = 20$, and $\gamma = 10$. Therefore, the proposed configurable Class-I decoder needs a total of $p(q\text{-}1)\rho = 5\times(32\text{-}1)\times20 = 3{,}100$ bits LUT and $(q\text{-}1)\rho = (32\text{-}1)\times20 = 620$ de-MUX's. However, its counterparts in [42-44] require 3 times the amount of wires and de-MUX's, and $p(q\text{-}1)[\rho+\gamma(\gamma\text{-}1)/2] = 5\times(32\text{-}1)\times[20+10(10\text{-}1)/2] = 10{,}075$ bits of LUT. The shuffle network in [44] needs even more, that is, a total of $p(q\text{-}1)(3\rho+\gamma\text{-}2) = 5\times31\times(3\times20+10\text{-}2) = 10{,}540$ bits of LUT are required. Since a conventional shuffle network is employed by the decoder proposed in [38], it not surprising that the highest complexity is required. Although having more flexibility, the proposed shuffle network still achieves hardware saving of 69.2%, 70.6%, and 66.7% (because the number of CNUs is only 3 in [38]) compared with prior works, respectively. For the proposed Class-II decoder design, we fail to give one comparison with other literatures, because no hardware implementation has been addressed by previous literatures. However, we can still derive the hardware reduction compared with the conventional approach which stores all the permutation information for each layer. For configurable version of the 32-ary (992, 496) rate-0.5 Class-II code decoder, the resulting hardware saving is

$$\frac{p(q-1)\rho k - p(q-1)\rho}{p(q-1)\rho k} = \frac{k-1}{k} = \frac{16-1}{16} \approx 93.8\%. \quad (2\text{-}20)$$

Although another $\rho(\log_2\rho\text{-}1/2) = 32\times(\log_2 32\text{-}1/2) = 144$ crossbar switches and $(\gamma\rho\log_2\rho)/2 = (32\times16\times\log_2 32)/2 = 1{,}280$ bits of LUT are required, the expense is relatively low compared with its benefits introduced. All the above analysis is for the proposed flexible decoders, for the specific ones, more hardware reduction can be expected with a sacrifice of configurability.

For the design of CNU block, the forward and backward steps in [41] can be eliminated as a result. Therefore, simple implementation of the proposed CNU architecture is achieved. Take a 32-ary (837, 726) rate-0.85 Class-I code as an example, it can be computed that only 85 registers and 572 equivalent XOR gates are required. This design requires less hardware consumption compared with designs in [42] and [44] while achieving the same critical path of 12 XOR gates. Here, all messages are quantized in 5 bits. In order to further meet the requirements of high speed applications, pipelines can be inserted into the comparator network. The pipelined comparator network is able to achieve critical path of 2 XOR gates with 125 more registers. If shorter latency is required, the concurrent version of comparator network can be employed, which will reduce the latency at the expense of larger area. In general, the throughput of an LDPC decoder can be calculated as follows:

$$\textbf{Throughput} = \frac{p \times u \times f}{T \times I_{\textbf{avg}}}, \quad (2\text{-}21)$$

where f is the clock frequency determined by the critical path, T is the number of clock cycles required by one decoding iteration, and $I_{\textbf{avg}}$ is the average number of decoding iterations to process one code-word. As an example, the throughput of the proposed 32-ary (992, 496) rate-0.5 Class-II flexible decoder can be estimated as follows. With the state-of-the-art technology, it is reasonable to assume that the executing frequency of the

decoder to be 150 MHz [44]. Also the maximum number of decoding iterations is set to 10, n_m is chosen to be 16, and the timing margin is given as 100 ns. The number of clock cycles required by each iteration is $(32\times32+16)\times16 = 16{,}640$. Hence, the resulting throughput is 4.47 Mb/s. If codes with higher rate and CNUs with larger concurrent number are employed, higher throughput can be expected.

2.7 Conclusion

In this chapter, new non-binary QC-LDPC decoder architectures have been presented by exploiting the unique geometry properties of the check matrices of both Class-I and Class-II non-binary LDPC codes. Examples of Class-I and Class-II decoders have demonstrated that the proposed design approaches can lead to up to 70.6% and 93.8% hardware reduction for the switch networks. Moreover, by introducing the generator for possible solution sequences, the CNU process is further simplified as well. Systematic analysis has shown that the proposed schemes can result in efficient decoder designs with decoding throughput of 6.46 Mb/s.

Chapter 3

Low-Latency Sequential SC Polar Decoder

This chapter is organized as follows. An introduction to the SC decoding algorithm and its logarithm domain variants is provided in Section 3.1. Section 3.2 reviews the prior works on this topic briefly. In Section 3.3, the DFG of the SC polar decoder is constructed and analyzed [58]. To the best knowledge of the authors, the architecture of the *feedback part* for the conventional SC tree decoder is presented [36] in Section 3.4. Using the DFG, the pre-computation look-ahead SC decoder architecture [36] is derived in Section 3.5. Performance estimation and comparison with the state-of-the-art designs are presented in Section 3.6. Section 3.7 concludes the whole chapter.

3.1 Introduction

Introduced by Arıkan recently [28], polar codes are capable of achieving the symmetric capacity $I(W)$ of any given binary-input discrete memoryless channel (B-DMC) W, when the code length N is considerable large. By recursively combining and splitting N copies of the B-DMC, we obtain a set of N binary-input coordinate channels $\{W_N^{(i)}\}$ $(1 \le i \le N)$. Among the N newly constructed channels, only those with the highest capacity are used

for data transmission. We refer inputs of these channels as *information bits* ($u_{\mathcal{A}}$), and the set of corresponding indices as $\mathcal{A}$. The inputs of the other channels are denoted as *frozen bits* ($u_{\mathcal{A}^c}$), and those channels' indices make up the set $\mathcal{A}^c$. A common used example is illustrated in Figure 3.1 [28], where the channel polarization phenomenon is quite obvious.

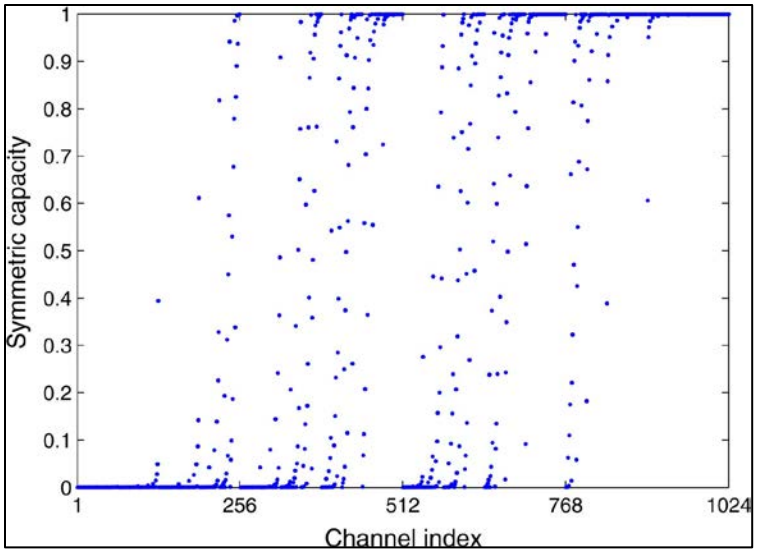


Figure 3.1 Channel polarization for binary erasure channel (BEC) of rate 0.5.

Compared to the well-known Turbo codes [82-84] and LDPC codes [20, 85, 86], polar codes are considered as the first codes that provably achieve the capacity for a fairly wide array of channels. Also it is claimed that the encoding and decoding of polar codes are of low complexity. These advantages make polar codes very attractive for real-life communication applications.

Suppose x_1^N and u_1^N are the input vector and encoded vector, respectively. G_N is the generator matrix. The encoding process of polar codes can be demonstrated with the following Eq. (3-1):

$$x_1^N = u_1^N G_N = B_N F^{\otimes N} = F^{\otimes N} B_N, \tag{3-1}$$

where B_N is the bit-reversal permutation, $F^{\otimes N}$ is the Kronecker power of N with

$$F \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{3-2}$$

As illustrated in Figure 3.2, this encoding operation can be implemented with the real-valued fast Fourier transform (RFFT) architectures proposed in [87]. Therefore, several newly proposed pipelined techniques can be employed.

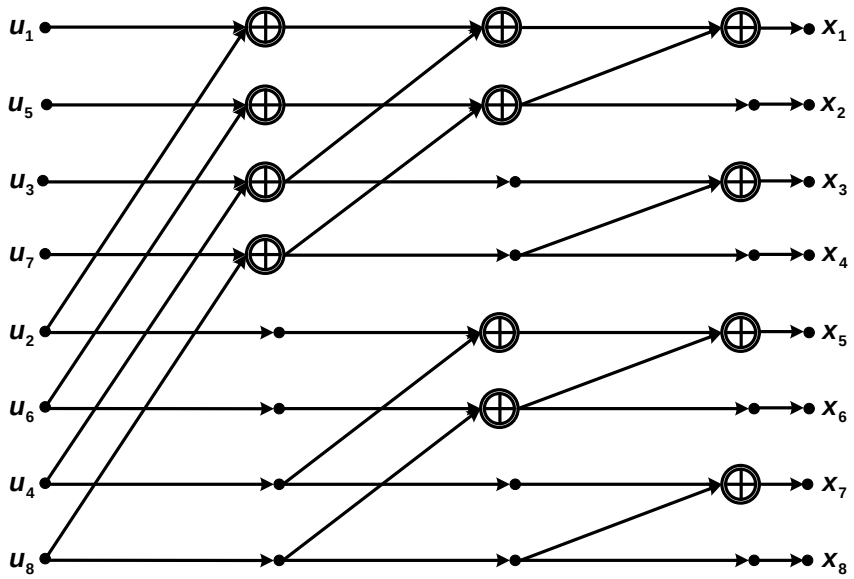


Figure 3.2 Encoding operation of the 8-bit polar code [28].

Similar to other codes such as Turbo codes and LDPC codes, for polar codes, the decoding process is more complicated compared with the encoding. In the remainder of this section, we provide the preliminaries of the SC decoding algorithm. Moreover, its logarithm variant and Min-Sum simplification version are explained as well.

3.1.1 SC Decoding Algorithm

Consider an arbitrary polar code with parameters (N, K, $\mathcal{A}$, $u_{\mathcal{A}^c}$) [28], where N and K represent the lengths of code bits and information bits, respectively. We denote the input vector as u_1^N,

which consists of a *random part* $u_{\mathcal{A}}$ and a *frozen part* $u_{\mathcal{A}^c}$. The corresponding output vector through channel W_N is y_1^N with conditional probability $W_N(y_1^N \mid u_1^N)$. The likelihood ratio (LR) is defined as,

$$L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} \mid 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} \mid 1)}. \tag{3-3}$$

In order to finish the decoding procedure, we are required to calculate the values of $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1})$ for all $1 \le i \le N$. The detailed calculations can be done by applying the following two equations recursively.

$$\begin{aligned} & L_N^{(2i)}(y_1^N, \hat{u}_1^{2i-1}) \\ &= [L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})]^{1-2\hat{u}_{2i-1}} \cdot L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}), \end{aligned} \tag{3-4}$$

$$\begin{aligned} & L_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-2}) \\ &= \frac{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}) + 1}{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) + L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})}. \end{aligned} \tag{3-5}$$

It can be observed that the calculation of $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1})$ depends on the estimate of the previous bit, from which the SC decoding algorithm is named. After we have got all the values of $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1})$ for $1 \le i \le N$, we need to finish the last step of the entire procedure. It is worth noting that if $i \in \mathcal{A}^c$, that is the corresponding bit is a *frozen bit*, the value of $\hat{u}_i$ is simply assigned as zero. The decision scheme is given as follows:

A Posteriori Decision Scheme with Frozen Bits

1: if $i \in \mathcal{A}^c$ **then** $\hat{u}_i = u_i$;
2: else
3: **if** $L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \ge 1$ **then** $\hat{u}_i = 0$;
4: **else** $\hat{u}_i = 1$;
5: **endif**
6: endif

The decoding procedure of polar codes with $N = 8$ is illustrated in Figure 3.3, where Type I and Type II processing elements (PEs) compute Eq. (3-4) and (3-5), respectively. The red label attached to each PE indicates the index of clock cycle when the corresponding PE is activated during the decoding process. It can be seen that the maximum value of the labels is 14, which means a total of 14 clock cycles are required to finish the SC decoding process for the 8-bit polar codes. This issue will be discussed further in following sections.

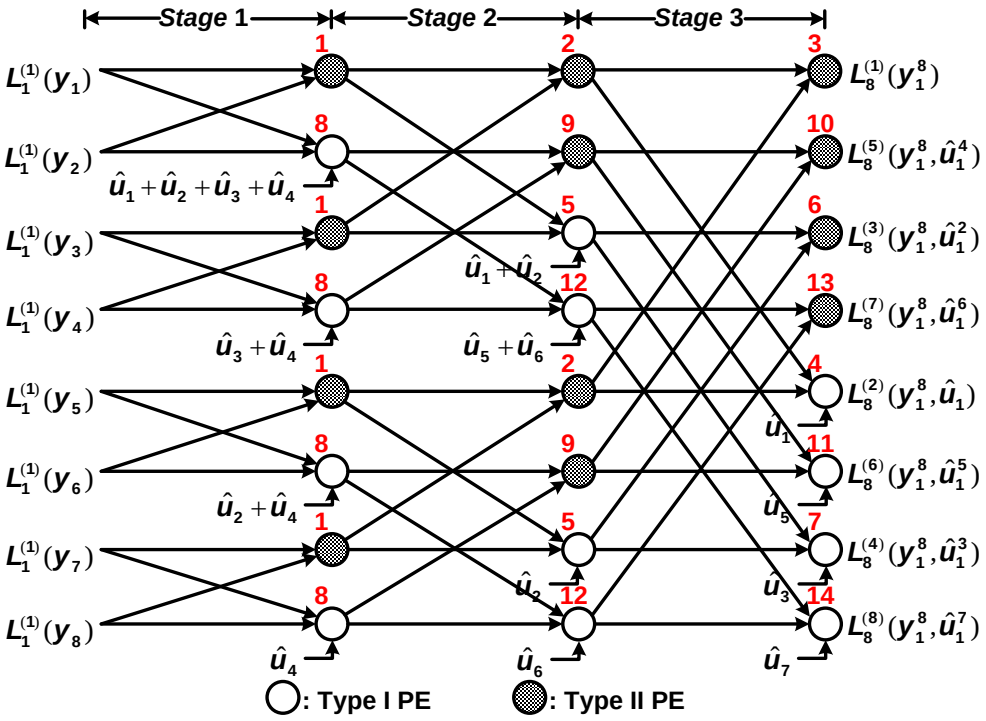


Figure 3.3 SC decoding process of polar codes with length $N = 8$.

3.1.2 SC Decoding Algorithm in Logarithm Domain

Generally, for every decoding algorithm defined in real domain, its logarithm variant always has advantages in terms of hardware utilization, computational complexity, and numerical stability [88]. Therefore, compared with their real-domain counterparts, the decoding algorithms defined upon logarithm-domain are more attractive to hardware

designers [89-91]. Similar to the approach addressed in [89], the SC algorithm dealing with logarithm-likelihood ratio (LLR) was also mentioned in [92]. First, we define the LLRs as follows:

$$\mathbb{L}_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \ln L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}). \tag{3-6}$$

Then, the previous real-domain Eq. (3-4) and (3-5) can then be rewritten as follows:

$$\begin{aligned} &\mathbb{L}_N^{(2i)}(y_1^N, \hat{u}_1^{2i-1}) \\ &= (-1)^{\hat{u}_{2i-1}} \mathbb{L}_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) + \mathbb{L}_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}), \end{aligned} \tag{3-7}$$

$$\begin{aligned} &\mathbb{L}_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-1}) \\ &= 2ar\tanh\{\tanh[\mathbb{L}_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})/2] \cdot \tanh[\mathbb{L}_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})/2]\}. \end{aligned} \tag{3-8}$$

3.1.3 Min-Sum SC Decoding Algorithm

Although the SC decoding algorithm in logarithm domain seems more hardware-friendly compared with the real-domain one, there is still one problem. In order to implement the hyperbolic tangent function and its inverse function in Eq. (3-8), a look-up table (LUT) of large size is required. In the cases when code length N is considerable large, the hardware consumption for the LUT would be very high. To this end, some sub-optimal algorithms which can achieve better trade-off between decoding performance and hardware cost are required as a result.

Note that in logarithm domain, for any variable $x \gg 1$, then we have the following approximation holds:

$$\ln[\cosh(x)] \simeq |x| - \ln 2. \tag{3-9}$$

Consequently, by applying the approximation, we reduce Eq. (3-8) to the Min-Sum update rule, which is LUT free:

$$\begin{aligned}\mathbb{L}_N^{(2i\text{-}1)}(y_1^N,\hat{u}_1^{2i-2}) \simeq & \frac{1}{2}\cdot\left|\mathbb{L}_{N/2}^{(i)}(y_1^{N/2},\hat{u}_{1,o}^{2i-2}\oplus\hat{u}_{1,e}^{2i-2})+\mathbb{L}_{N/2}^{(i)}(y_{N/2+1}^{N},\hat{u}_{1,e}^{2i-2})\right| - \\ & \frac{1}{2}\cdot\left|\mathbb{L}_{N/2}^{(i)}(y_1^{N/2},\hat{u}_{1,o}^{2i-2}\oplus\hat{u}_{1,e}^{2i-2})-\mathbb{L}_{N/2}^{(i)}(y_{N/2+1}^{N},\hat{u}_{1,e}^{2i-2})\right| \\ & =\operatorname{sgn}[\mathbb{L}_{N/2}^{(i)}(y_1^{N/2},\hat{u}_{1,o}^{2i-2}\oplus\hat{u}_{1,e}^{2i-2})]\operatorname{sgn}[\mathbb{L}_{N/2}^{(i)}(y_{N/2+1}^{N},\hat{u}_{1,e}^{2i-2})]\cdot \\ & \min[\left|\mathbb{L}_{N/2}^{(i)}(y_1^{N/2},\hat{u}_{1,o}^{2i-2}\oplus\hat{u}_{1,e}^{2i-2})\right|,\left|\mathbb{L}_{N/2}^{(i)}(y_{N/2+1}^{N},\hat{u}_{1,e}^{2i-2})\right|].\end{aligned} \tag{3-10}$$

Simulation results have demonstrated that the Min-Sum SC decoding algorithm only suffers from little performance degradation than the optimal one while achieving high hardware utilization [92]. This makes Min-Sum SC decoding algorithm very attractive for VLSI implementation. Therefore, in this chapter we will focus our discussion of the polar decoder design on this sub-optimal algorithm only.

3.2 Prior Works on SC Polar Decoder Designs

Most of the research on polar codes has been focused on code performance rather than the design of high efficiency decoder architectures. Shown in [28], the straightforward polar decoder implementation using successive cancellation (SC) algorithm results in the complexity of $\mathcal{O}(N\log_2 N)$. Several design examples based on belief propagation (BP) algorithm were also proposed in [28] and [34, 93]. However, due to its lower complexity compared with the BP algorithm, the SC approach appears more attractive for hardware implementation. Reduced SC decoders with complexity of $\mathcal{O}(N)$ were presented in [92] and [94]. However, to decode a polar code with length of N, 2(N-1) clock cycles are required by the decoder. For real-time applications, in order to achieve the required decoding performance, code length N higher than 2^{10} is usually a necessity. Therefore, the approach proposed in [92] and [94] requires long decoding latency for large N. Another problem is that in each active stage of the decoder the highest hardware utilization can be only 50%, which means half of the processing elements (PEs) are idle at the same time. This is simply because the estimation of the current bit depends on the

value of the previous coded bit, which forces all coded bits to be output sequentially. A latency-reduction approach called *simplified SC* decoding has been proposed in [60] recently. Since this method is code-specific and highly influenced by the channel model, it would not be discussed in this chapter. An empirical method to construct pre-computation look-ahead decoding schedule for any polar codes was presented in [36] to shorten the decoding latency by half. However, without investigating the inherent properties of the decoder's data-flow graph (DFG), this method cannot be generalized to derive other decoder designs to meet different real-time application restrictions.

This chapter makes three contributions. First, a DFG is derived and analyzed for the SC decoding process. According to the analysis result, validation of the empirical pre-computation look-ahead approach in [36] is guaranteed. It is shown that the SC decoding process can in fact be described by a multi-rate DFG [95, 96], whose iteration bound is determined by the execution time of a single processing element (PE) only. Second, based on the DFG analysis, we present a complete pipelined hardware architecture for the conventional tree SC polar decoder for the first time. A new sub-block called the *feedback part* is proposed to compute the 1-bit input signals for Type I processing elements (PEs). We show that this sub-block can be constructed recursively. We also give a systematic method to conveniently determine the select signals for de-/multiplexers. The pre-computation look-ahead decoder can be easily constructed within the new DFG context. It is demonstrated that compared with the architecture in [92], the proposed low-latency architecture can halve the decoding latency with similar hardware consumption. Comparison results show that the proposed pre-computation look-ahead sequential decoder has the advantage in latency or throughput compared with the conventional tree SC decoders.

3.3 DFG Analysis of the SC Polar Decoder

The DFG captures the data-driven property of DSP algorithms where any node can fire whenever all the input data are available [37]. In a DFG, the nodes represent functions, while the directed edges represent the data communications between nodes. Each edge has a nonnegative number of delays associated with it [97]. In order to derive the pre-computation look-ahead SC decoder architectures, we first consider construction of the DFG of the SC polar decoder in this section.

3.3.1 DFG Construction for the SC Polar Decoder

In order to construct the DFG of the SC polar decoder, certain pre-processing is required. Illustrated in Figure 3.4, each PE in the SC decoding process is marked with a new label (red) for better identification.

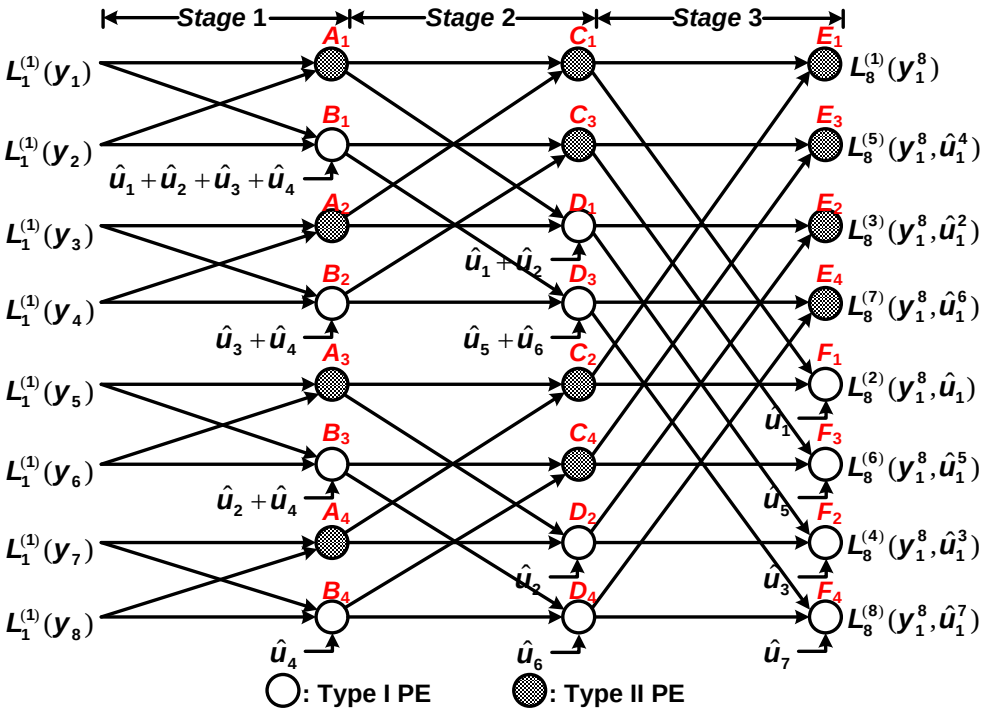


Figure 3.4 SC decoding process with new labels.

According to the dependence relationship of each PE, the straightforward DFG can be constructed as follows in Figure 3.5. The “D” represents a delay element. All output bits are marked at the same clock cycle in which they are generated. By the way, it is worth to point out that the proposed DFG can be partitioned into two identical parts as shown by the dotted circles.

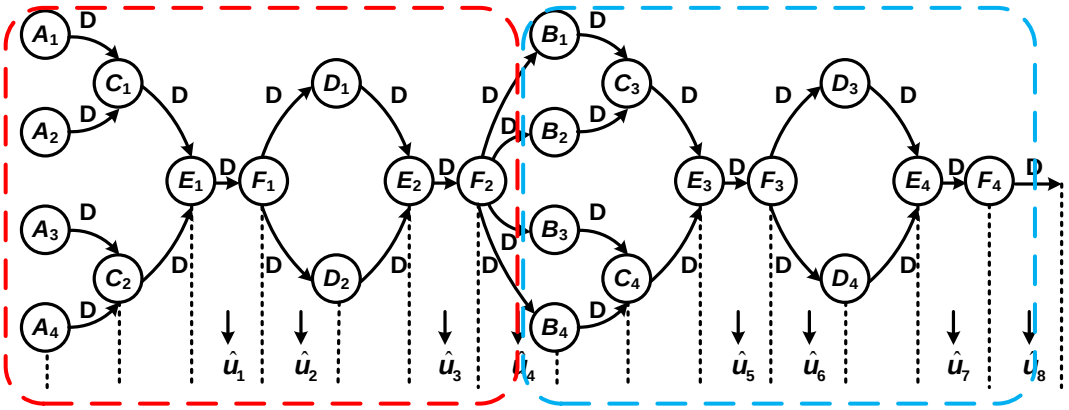


Figure 3.5 DFG for the 8-bit SC decoding process.

Since the PEs $A_{1,2,3,4}$ are functionally identical, we can merge them together and represent the *merged* one with *A*. Similar approaches can be applied to other PEs. After merging all similar PEs together, we can derive a more compact version of DFG shown in Figure 3.6. The red dash line indicates the flow of the entire decoding procedure, which also corresponds to the decoding latency. The {*i*} affiliated to each switch means the corresponding closure occurs during *Clock Cycle* $14n+i$, where $n \geq 0$. The figures at the input or output of each PE represent the number of samples consumed or produced by an invocation of that PE [37].

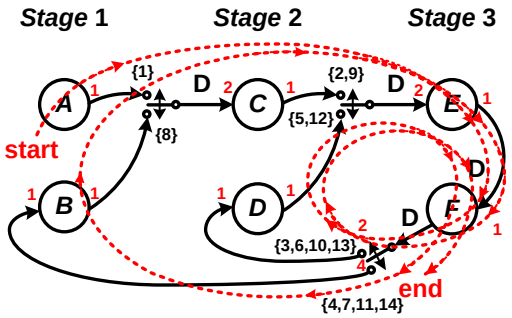


Figure 3.6 Simplified DFG for 8-bit SC decoding process.

According to the proposed DFG, it can be seen that during the entire decoding process, PEs in *Stage i* will be activated 2^{i-1} times. Therefore, the decoding latency of 8-bit SC polar decoder is $2\times(2^2+2^1+2^0) = 14$ clock cycles. More generally, with similar DFGs we can further verify that the decoding latency should be

$$2\times\sum_{i=1}^{\log_2 N} 2^{i-1} = 2\times\frac{2^{\log_2 N}-1}{2-1} = 2(N-1), \tag{3-11}$$

which matches with the empirical conclusion derived in [36]. Since the numbers at both ends of an edge are different, the conventional SC polar decoder is actually a multi-rate system rather than a single-rate system [98].

More importantly, the iteration bound of the DFG is always determined by the processing time of one PE. The *iteration bound* is defined as the maximum loop bound. The *loop bound* represents the lower bound on the loop computation time. In summary, two properties of the DFG are stated as follows:

1. The decoding latency for an N-bit SC polar decoder equals $2(N-1)$ clock cycles;
2. The iteration bound of the DFG equals the processing time of a single PE.

Proof The proof of the first property is given by Eq. (3-11) already. For the second property, the DFG in Figure 3.6 contains four loops (Figure 3.7), namely, the loops

$$l_1 = E \to F \to D \to E, \qquad l_2 = F \to D \to E \to F,$$
$$l_3 = E \to F \to B \to C \to E, \quad l_4 = F \to B \to C \to E \to F. \tag{3-12}$$

Suppose the processing time for each PE is T. The loop bounds for l_1, l_2, l_3, and l_4 are $3T/3 = T$, $3T/3 = T$, $4T/4 = T$, and $4T/4 = T$, respectively. Thus, the iteration bound is

$$T_\infty = \max\{\tfrac{3T}{3}, \tfrac{3T}{3}, \tfrac{4T}{4}, \tfrac{4T}{4}\} = T. \tag{3-13}$$

Generally, in the DFG for the N-bit SC polar decoder, each loop has the same number of PEs and delay elements. Hence, the iteration bound of the DFG equals the processing time of a single PE.■

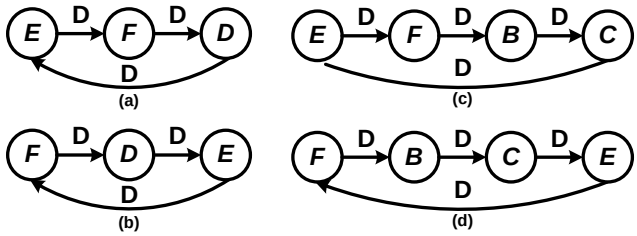


Figure 3.7 Four loops of the simplified DFG.

For the ease of understanding, the corresponding decoding schedule for the 8-bit SC decoder is shown in Table 3.1. *C* refers to the code-word which is being processed. Again, similar to the proposed DFG, it is observed that the second half of the decoding schedule is identical to the first half.

Table 3.1 Decoding schedule for 8-bit SC decoder.

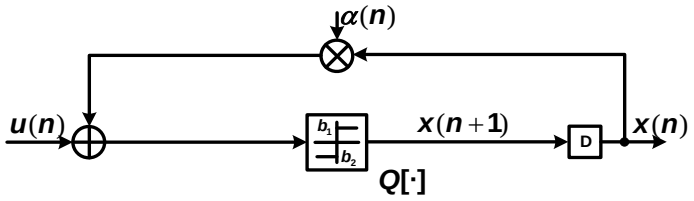
Stage	**Clock cycle**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	*C*	—	—	—	—	—	—
2	—	*C*	—	—	*C*	—	—
3	—	—	*C*	*C*	—	*C*	*C*
Stage	**Clock cycle**						
	8	**9**	**10**	**11**	**12**	**13**	**14**
1	*C*	—	—	—	—	—	—
2	—	*C*	—	—	*C*	—	—
3	—	—	*C*	*C*	—	*C*	*C*

We now can transform the proposed DFG in Figure 3.6 to achieve iteration bound with lower decoding latency. This is addressed in the following sub-section.

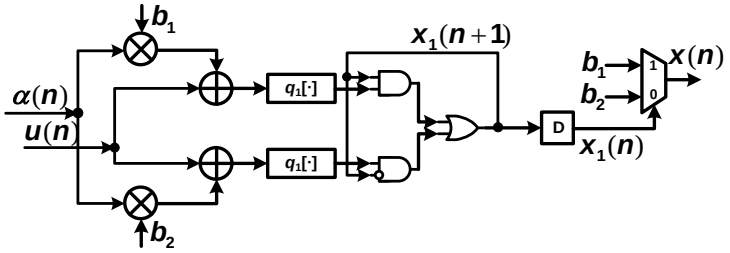
3.3.2 DFG with Pre-Computation Look-Ahead Techniques

Pre-computation look-ahead approach is also referred to as *parallel branch and delayed decision* approach [99]. This approach is often used to achieve faster processing rate when the number of possible outputs is finite.

Therefore, we could always perform all possible calculations *a priori*, get all the candidate results, and select the suitable one from the candidate pool thereafter. This approach can save us more processing time. Figure 3.8 illustrates a simple example of a first-order two-level quantizer loop [99]. Its pre-computation look-ahead reformulated version is illustrated in (b) part of the same figure. According to the figure, we can see that the latter one can potentially achieve faster speed.



(a) A first-order two-level quantizer loop.



(b) Its pre-computation look-ahead reformulated version.

Figure 3.8 A quantizer example for pre-computation look-ahead approach [99].

As mentioned in [36], for two possible LLR inputs, there are only two output candidates available for a Type I PE, depending on whether the control bit $\hat{u}_{2i-1}$ in Eq. (3-7) is 1 or 0. According to the pre-computation look-ahead approach, both output candidates can be pre-computed with the look-ahead manner [100, 101] and Type I and Type II PEs in the same stage are activated within the same clock cycle. The DFG illustrated in Figure 3.6 can then be modified as follows:

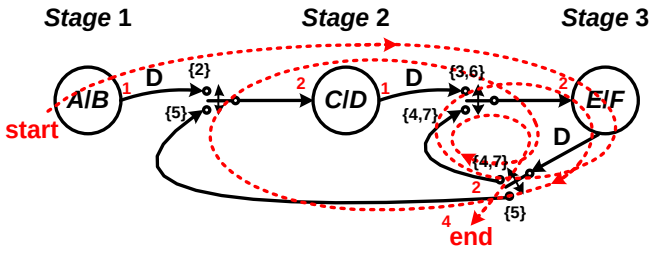


Figure 3.9 DFG for 8-bit pre-computation look-ahead SC decoding process.

Similar to Figure 3.6, each PE in *Stage i* will be activated for 2^{i-1} times. Compared with the conventional one, the total number of PEs in the transformed DFG has been reduced by 50%. For the 8-bit SC polar decoder example, its decoding latency has been halved and is only $2^2+2^1+2^0 = 7$ clock cycles, which matches with the results derived in [36].

Table 3.2 8-bit pre-computation look-ahead decoding schedule.

Stage	**Clock cycle**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Pre-computation look-ahead decoding schedule							
1	*C*	—	—	—	—	—	—
2	—	*C*	—	—	*C*	—	—
3	—	—	*C*	*C*	—	*C*	*C*

According to the proposed design details described later in the same chapter, the critical path of the transformed DFG stays the same as before. Therefore, the decoding latency has been halved also. Not only it coincides with the prior results, the DFG

approach indeed provides a more fundamental explanation of the pre-computation look-ahead decoder construction. The length of the decoding schedule is reduced as well, which is shown in Table 3.2:

3.4 Pipelined Tree Decoder Architecture

Rather than dealing with the complicated conventional time-chart construction methods, the DFG approach proposed here enables us to design different SC polar decoder architectures formally. In this section, we proposed a systematic construction approach for the conventional tree SC polar decoder architecture for the first time.

3.4.1 Complete SC Tree Decoder Architecture Design

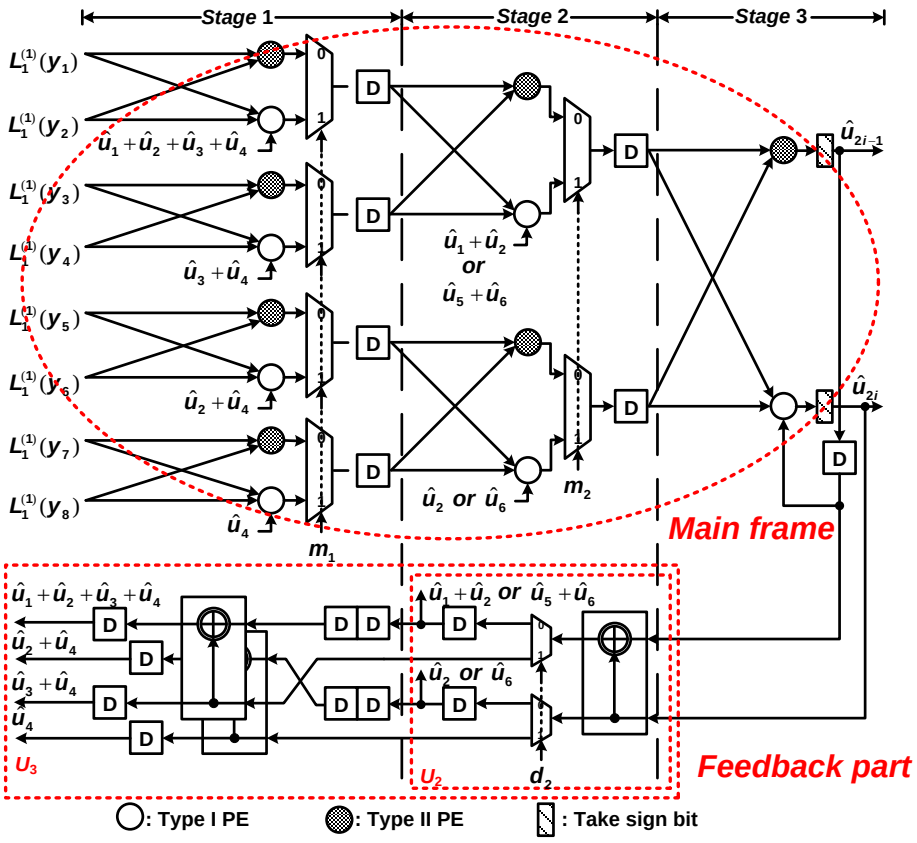


Figure 3.10 The 8-bit conventional tree decoder architecture.

The conventional tree SC polar decoder architecture was first proposed in [92]. However, the architectures for the *feedback part* and the control logic were not presented in [92]. In this section, the complete design of the conventional tree decoder architecture is described in detail. Without loss of generality, an 8-bit SC polar decoder example is used in the following sections. All approaches employed by this simple example can be extended to general *N*-bit SC polar decoders in a similar fashion.

Figure 3.10 illustrates the architecture for the conventional pipelined 8-bit SC decoder. The top part is referred to as the *main frame*, and the bottom part is referred to as the *feedback part*. In this decoder architecture, two kinds of PEs are employed. With those PEs, the *main frame* (dotted circle in Figure 3.10) of the decoder can be constructed in a *full binary tree* manner. A total of three stages are required, and *Stage i* is composed of $N/2^i$ copies of Type I PEs and Type II PEs.

3.4.2 Architecture of Type I PE

According to Eq. (3-7), the Type I PE is nothing but a *W*-bit adder-subtractor, where *W* is the fixed word-length of LLRs. It can be simply implemented using the architecture [102] in Figure 3.11 as follows.

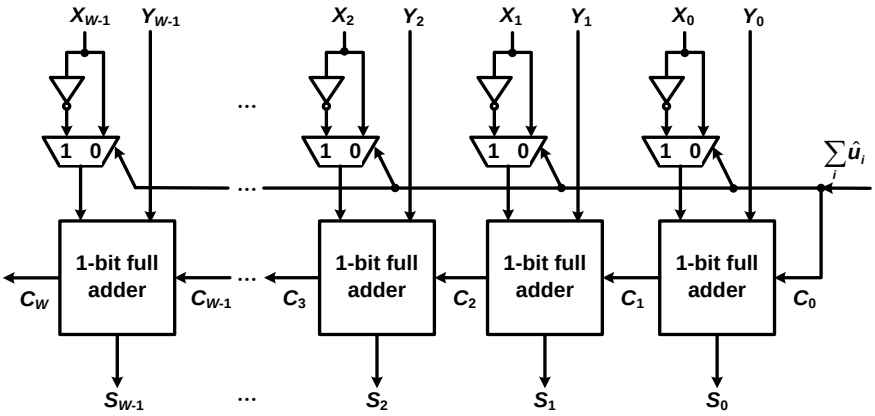


Figure 3.11 Proposed Type I PE architecture.

3.4.3 Architecture of Type II PE

Mentioned previously, the Type II PE employs the Min-Sum algorithm instead of employing the *tanh* and *artanh* functions. The architecture of the Type II PE is given in Figure 3.12 as follows.

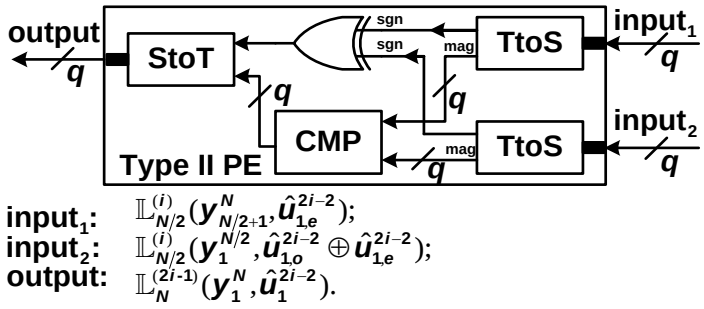


Figure 3.12 Proposed architecture of Type II PE.

The *TtoS* block performs the conversion from two's complement representation to sign-magnitude representation. The *StoT* block performs the reverse conversion. The architecture of the *TtoS* block is illustrated in Figure 3.13. In order to avoid the overflow situation, a sign extension operation is required as well.

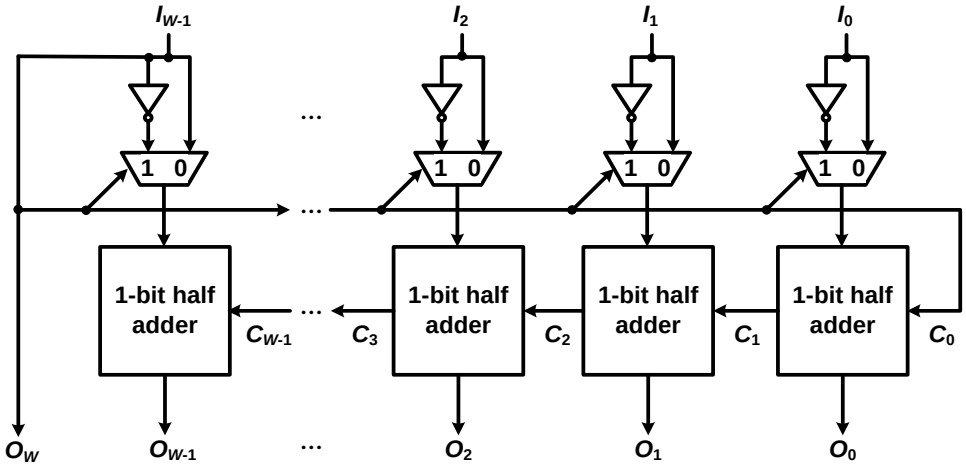


Figure 3.13 Proposed structure of the *TtoS* block.

The *StoT* block is similar to the *TtoS* block. The only difference is that a sign compression operation is needed by the *StoT* block to make the output data in the form of the *W*-bit quantization.

3.4.4 Architecture of the Feedback Part

Now, we have derived the architectures of the PEs employed by the *main frame*. Compared with the straightforward implementation of the *main frame*, the construction of feedback loop is not trivial. The *feedback part* computes the 1-bit input signals for Type I PEs. Indicated by Eq. (3-7), except for the two inputs $\mathbb{L}^{(i)}_{N/2}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})$ and $\mathbb{L}^{(i)}_{N/2}(y_{N/2+1}^{N}, \hat{u}_{1,e}^{2i-2})$, a third input $\hat{u}_{2i-1}$ is also required by Type I PE to process the computation.

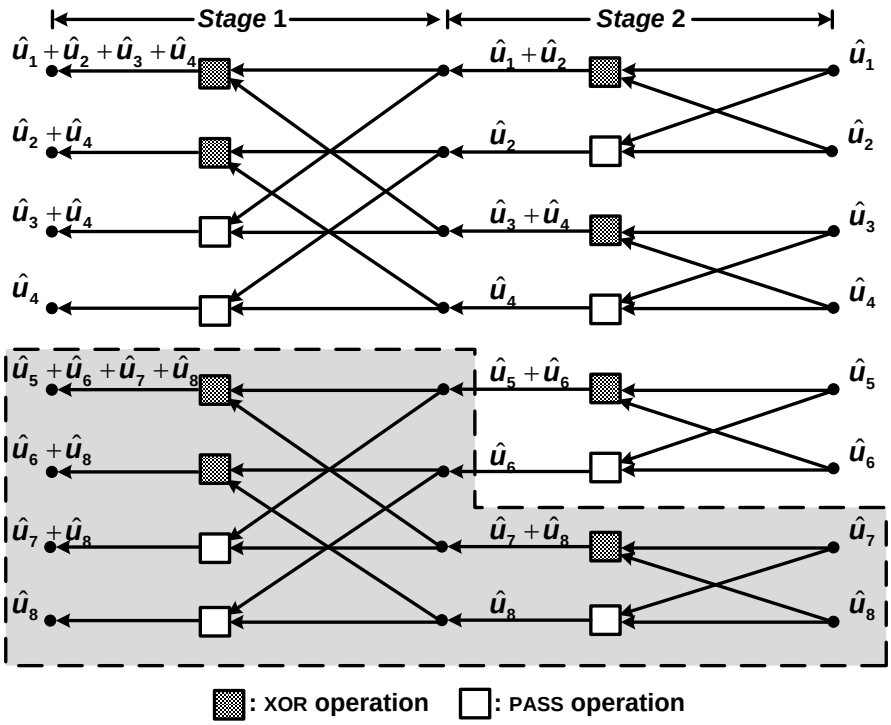


Figure 3.14 Flow graph of *feedback part* for 8-point polar decoder.

For efficient execution of each Type I PE, the value of $\hat{u}_{2i-1}$ needs to be provided on the fly. However, even for the 8-bit decoder illustrated in Figure 3.10, the complicated interleaving of odd and even indices makes the straightforward calculation of $\hat{u}_{2i-1}$ inconvenient. In order to solve this inherent problem, a method to construct the *feedback part* is presented here.

Careful investigation has shown that it is possible to generate the required $\hat{u}_{2i-1}$ using the real FFT-like signal flow [87]. All the extra input values $\hat{u}_{2i-1}$ for 8-bit polar code decoder can be easily generated using the flow graph in Figure 3.14. Here, the PASS operation PE only lets the lower input get through.

Furthermore, the flow graph in Figure 3.14 can be simplified with the two properties explained below:

1. The first simplification is to consider that all outputs associated with inputs $\hat{u}_{N-1}^{N}$ are not necessary. Consequently, the shaded region in Figure 3.14 can be removed. Similar concept can be applied to the general case of N inputs. For any *Stage i*, its lower region which contains $N/2^i$ processing elements (PEs), can be removed. For example, the lower 4 PEs of *Stage* 1 and the lower 2 PEs of *Stage* 2 are removed from the flow graph in Figure 3.14. Therefore, $(N/2)(\log_2 N\text{-}1)$ outputs need to be computed.
2. The second simplification refers to the fact that the PASS operation element can be replaced by the wire connection while the flow graph stays functionally the same. According to Figure 3.14, the PASS operation element only allows the lower input to the next stage. Thus, if the upper input is not treated as an input to the PASS operation element any more, one simple wire which connects the lower input and the output can be employed instead. Thus, complexity of the feedback flow graph can be halved with respect to the former one.

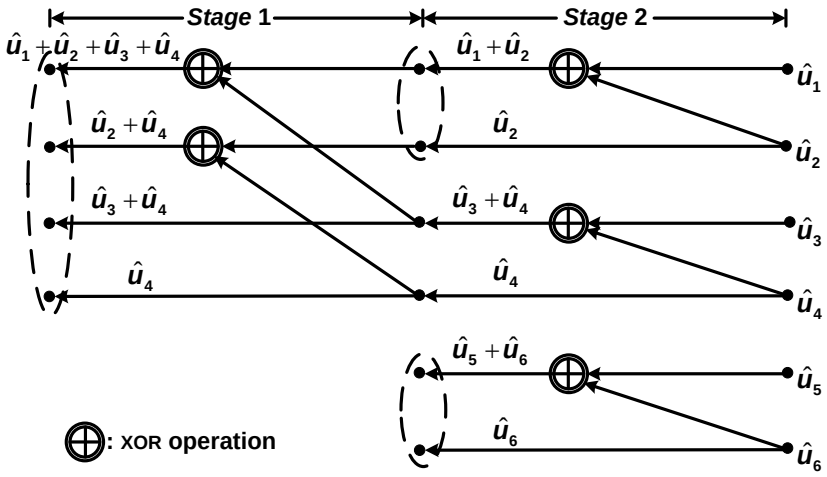


Figure 3.15 Simplified flow graph of the proposed *feedback part*.

The resulting simplified data-flow graph is shown in Figure 3.15, where a new symbol is employed to denote the XOR operation. Both properties have been fully utilized. It can be seen that a total of ($\log_2 N$-1) stages are required and the number of XOR operations is given by:

$$\begin{aligned}&[N(\log_2 N-1)-2(1-N/2)/(1-2)]/2\\&=N(\log_2 N-2)/2+1.\end{aligned} \tag{3-14}$$

In addition, it is worthwhile to note that with the help of *generator matrix* G_N [1], the same simplified flow graph can be obtained as well. As mentioned previously, We define $\otimes$ to be the matrix Kronecker product [103] and n_0 equals $\log_2 N$. Then the g*enerator matrix* G_N is given by the following equation:

$$G_N = B_N F^{\otimes n_0} = F^{\otimes n_0} B_N, \tag{3-15}$$

where B_N is the bit-reversal permutation matrix and F matrix is defined as:

$$F \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{3-16}$$

The pipelined architecture of the simplified flow graph in Figure 3.15 can be implemented with the following feed-forward architecture, where two XOR-PASS elements are required.

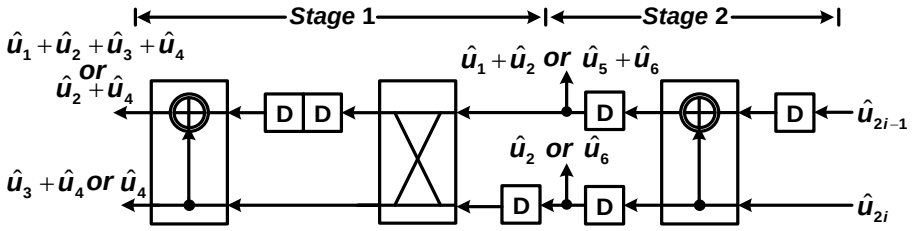


Figure 3.16 Pipelined feed-forward architecture for 8-bit *feedback part*.

It can be observed that the proposed pipelined architecture can only generate two outputs during the same clock cycle. However, as indicated by Figure 3.10, in order to work compatibly with the *main frame* of the decoder, *Stage* 1 of the *feedback part* should be able to generate 4 outputs at the same time. Therefore, *Stage* 1 needs to be modified to a 2-level parallel processing structure, which is shown in Figure 3.17, where U_i denotes an architecture which consists of i stage(s). *Stage* i computes $N/2^i$ outputs at the same time. In addition, the details of how to determine the select signal d_2 will be described in the following sub-section.

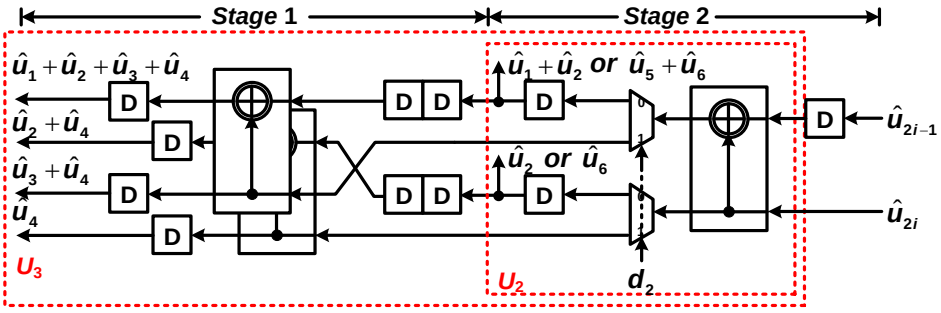


Figure 3.17 The 2-parallel version of the architecture in Figure 3.16.

In general, for an N-bit length decoder, since the data structures of the *feedback part* are defined recursively for powers of 2, the general parallel pipelined architecture can be constructed with the recurrence relationship. The recursion for the general case is shown explicitly in Figure 3.18.

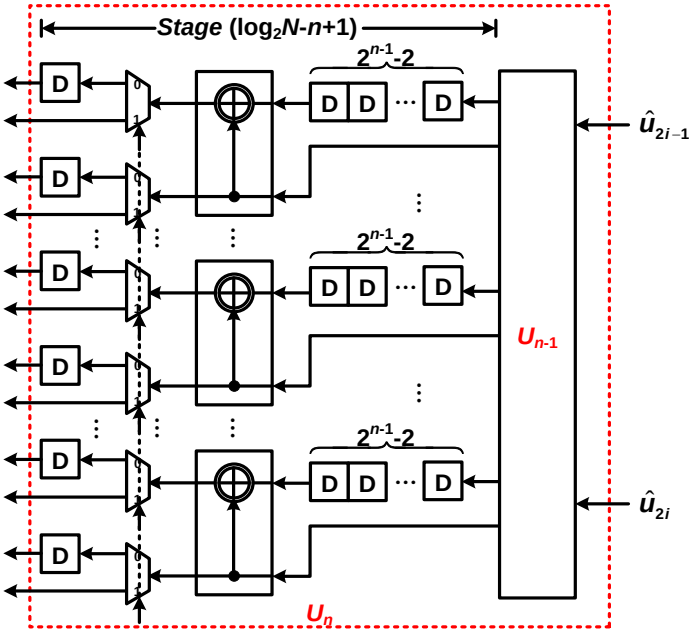


Figure 3.18 Recursive construction of U_n based on U_{n-1}.

Here module U_n can be constructed based on module U_{n-1} and (n-1) extra XOR-PASS elements. It can be observed that in order to cooperate with the *main frame* perfectly, the delay elements have been rearranged to equip each stage with output registers. For U_n with $n_0 = \log_2 N$, the de-multiplexers are not required.

3.4.5 Selecting Signals for De-/Multiplexers

Another important issue we need to address clearly is at which time instance the de-/multiplexers need to be switched. As shown in Figure 3.10, the select signals of the multiplexers and de-multiplexers in *Stage* i are denoted as m_i and d_i ($1 \leq i < \log_2 N$),

respectively. The switching schedule for m_i can be obtained according to Table 3.3. The corresponding clock cycle index (c) is determined with the following equation:

$$c = \sum_j b_j w_j + i, \tag{3-17}$$

where b_j, w_j, and i represent the binary digit, weight, and stage index, respectively. The values of m_i are assigned to be alternately "0" and "1". For example, for *Stage* 1, m_1 is set to 0 in *Clock cycle* 1, where $1 = 0 \times (2^3\text{-}1) + 1$.

Although it can be observed in Figure 3.10 that m_3 is actually not required in the design, the given switch time for m_3 is still helpful to determine the clock cycle when each decoded bit is output. As indicted by Table 3.1, the decoded bits with odd indices are output in *Clock cycle*s 3, 6, 10, and 13. Those with even indices are output in *Clock cycle*s 4, 7, 11, and 14.

Table 3.3 Calculation of switching time for m_n.

Weight			**Stage index**	**Clock cycle**	m_n
2^3-1	2^2-1	2^1-1			
***Stage* 1 (*n* = 1)**					
0	—	—	1	1	0
1	—	—	1	8	1
***Stage* 2 (*n* = 2)**					
0	0	—	2	2	0
0	1	—	2	5	1
1	0	—	2	9	0
1	1	—	2	12	1
***Stage* 3 (*n* = 3)**					
0	0	0	3	3	0
0	0	1	3	4	1
0	1	0	3	6	0
0	1	1	3	7	1
1	0	0	3	10	0
1	0	1	3	11	1
1	1	0	3	13	0
1	1	1	3	14	1

In order to determine the switching time for de-multiplexers' control signal d_i, we re-visit Figure 3.10. Two properties are observed as follows:

1. Signal d_i needs to be set as 0 one clock cycle before signal m_i is set as 1;
2. Signal d_i needs to be set as 1 in the same clock cycle when any d_j ($i<j$) is set as 0.

Using these properties along with Table 3.3, the switch details for m_i and d_i signals shown in Figure 3.10 are listed in Table 3.4 below.

Table 3.4 Select signals for 8-bit decoder example.

Clock cycle	m_1	m_2	d_2
1	0	—	—
2	—	0	—
3	—	—	—
4	—	—	0
5	—	1	—
6	—	—	—
7	—	—	1
8	1	—	—
9	—	0	—
10	—	—	—
11	—	—	0
12	—	1	—
13	—	—	—
14	—	—	

3.5 Pre-Computation Look-Ahead Sequential Decoder

Since the transformed DFG of the conventional SC decoder has been derived in Section 3.3, it provides us a more fundamental way to explain the pre-computation look-ahead approach. In this section, a novel pre-computation look-ahead sequential SC polar decoder architecture is presented based on the DFG analysis. Comparison has shown that this architecture can achieve 50% decoding latency with similar hardware consumption.

3.5.1 Architecture of the Revised Type I PE

According to the pre-computation look-ahead scheme, the Type I PE in the pre-computation look-ahead decoder is in charge of pre-computing two possible outputs **in parallel**. Therefore, the new Type I PE is called the revised Type I PE here.

In the revised Type I PE, we need to incorporate both capabilities of adding or subtracting operands together and finish them simultaneously. As shown in Figure 3.19, the straightforward parallel implementation of the Type I PE can be employed here. It can be observed that in the straightforward way, two identical copies of the architecture in Figure 3.11 are employed.

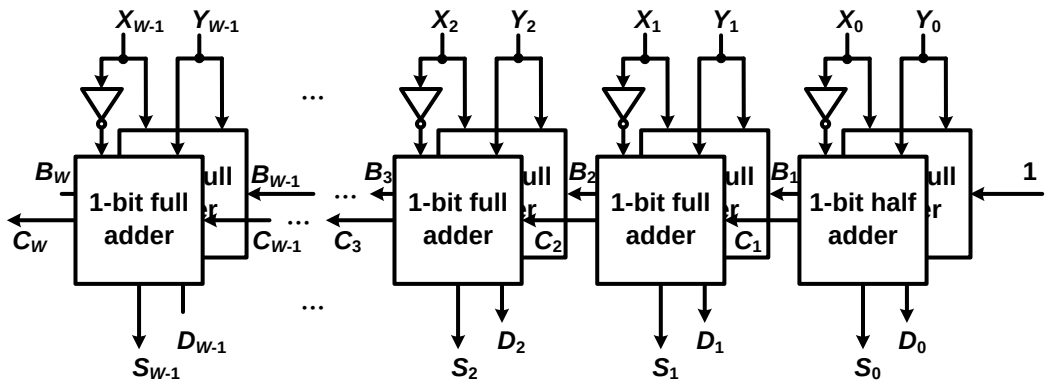


Figure 3.19 The *W*-bit parallel adder-subtractor architecture.

However, simple duplication results in penalty of doubling both area and power consumption. For a *W*-bit parallel adder-subtractor, totally 2*W*-1 1-bit full adder and one 1-bit half adder are required. In order to implement the revised Type I PE more effectively, the complexity-reduced architecture of the parallel adder-subtractor is proposed here. Rather than implementing the revised Type I PE with two's complement approach, the original carry-borrow idea is employed here. Suppose *X* and *Y* are the two operands, and Z_{in} is the carried-in or borrowed-from bit. For the 1-bit full adder, the two output bits of summation and carry-out are represented by *S* and C_{out}, respectively. In

similar way, the difference and borrow-out bits produced by the 1-bit full subtractor are denoted with D and B_{out}. Therefore, the truth table for the 1-bit full adder and subtractor is given as follows.

Table 3.5 Truth table of both full adder and subtractor.

Inputs			**Outputs**			
			Adder		**Subtractor**	
X	Y	Z_{in}	S	C_{out}	D	B_{out}
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

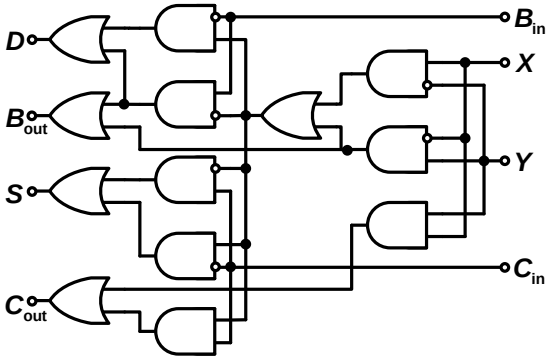
From Table 3.5, we can draw the corresponding *Karnaugh* map for all outputs. Then the logic equations are derived as follows:

$$\begin{cases} S = X \oplus Y \oplus Z_{\mathbf{in}}; \\ C_{\mathbf{out}} = X \cdot Y + (X \oplus Y) \cdot Z_{\mathbf{in}}. \end{cases} \tag{3-18}$$

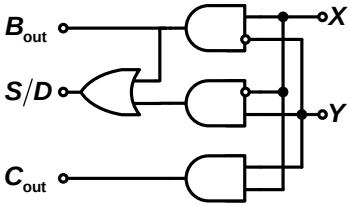
$$\begin{cases} D = X \oplus Y \oplus Z_{\mathbf{in}}; \\ B_{\mathbf{out}} = \overline{X} \cdot Y + \overline{X \oplus Y} \cdot Z_{\mathbf{in}}. \end{cases} \tag{3-19}$$

According to the equations, it can be noticed that S and D are actually the same. With the help of Boolean logic, we know that $\overline{X} \cdot Y$ is an intermediate term of $X \oplus Y$ operation. Similarly, $\overline{(X \oplus Y)} \cdot Z_{\mathbf{in}}$ can be treated as a byproduct of the term $X \oplus Y \oplus Z_{\mathbf{in}}$ as well. These observations enable us to employ the gate-level sub-block sharing scheme to implement the required parallel adder-subtractor. This scheme not only implements the parallel processing but also helps us to reduce the hardware consumption. The gate-level

structures of the proposed 1-bit parallel full and half adder-subtractor are depicted in Figure 3.20 (a) and (b), respectively.



(a) 1-bit parallel full adder-subtractor.



(b) 1-bit parallel half adder-subtractor.

Figure 3.20 Proposed 1-bit parallel adder-subtractor architectures.

For the sake of easy estimation and comparison, the hardware consumption of the proposed architectures is converted in the form of equivalent XOR gate number. According to Figure 3.20, the complexities of 1-bit parallel full and half adder-subtractor equal to 4 XOR gates and 1 XOR gate, respectively. Compared with the straightforward implementation methods, the total hardware savings of the proposed approaches are 43% and 50%, respectively.

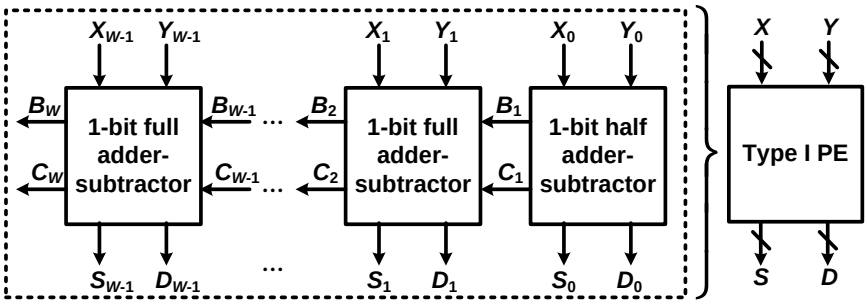


Figure 3.21 The revised Type I PE architecture.

Illustrated in Figure 3.21, the revised Type I PE is composed of *W*-1 1-bit full adder-subtractor and a 1-bit half adder-subtractor. With the proposed design method, it requires 57% hardware of the conventional one in Figure 3.19 while achieves exactly the same function.

3.5.2 Architecture of the Merged PE

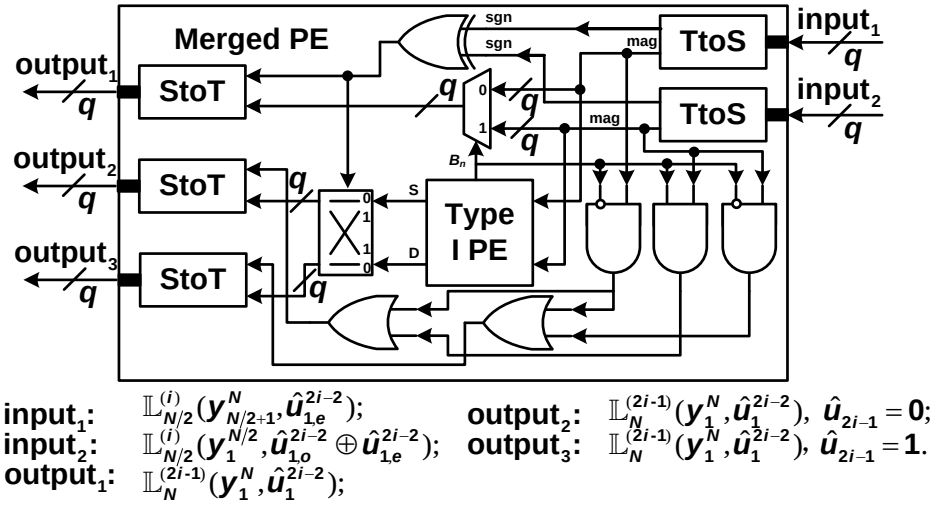


Figure 3.22 Proposed structure of the *merged* PE.

Since in the pre-computation look-ahead approach, the function of the Type II PE stays unchanged, the corresponding hardware architecture is the same as the one illustrated in Figure 3.12. According to Figure 3.12, a comparator is required by the Type

II PE. Actually, the comparator is a *W*-bit subtractor, which is also employed by the revised Type I PE. Hence, it is possible to incorporate both the revised Type I PE and Type II PE together. By employing the sub-structure sharing scheme again, we can generate the architecture of *merged* PE. The detailed structure is illustrated in Figure 3.22.

In the *merged* PE architecture, the comparison operation is carried out by the revised Type I PE illustrated in Figure 3.21. Two more *StoT* blocks as well as additional control logic are required here. According to Figure 3.22, a total of 2*W*-3 XOR gates can be saved with the proposed sub-structure sharing approach. As mentioned previously, in order to achieve good decoding performance in real-time communication applications, polar codes with length *N* over 2^{10} is required. For the conventional tree SC polar decoder architecture, a total of *N*-1 Type I PEs and *N*-1Type II PEs are used, respectively. If *N*-1 *merged* PEs can be employed instead, the resulting hardware saving could be around $2^{10}\cdot(2W\text{-}3)$ XOR gates.

3.5.3 Decoder Architecture Construction

Since now we have the architecture of the proposed *merged* PE ready, the pre-computation look-ahead sequential decoder architecture is presented in this sub-section. The 8-bit pre-computation look-ahead SC decoder is illustrated in Figure 3.23. The proposed *merged* PEs are used instead of Type I and Type II PEs.

It can be noticed that the positions of pipelines have changed a little bit. However, since the decoding algorithm stays the same, the decoding performance will not be affected. Also the *feedback part* for the pre-computation look-ahead sequential decoder has changed a little. More details of the new *feedback part* will be discussed in the following sub-section.

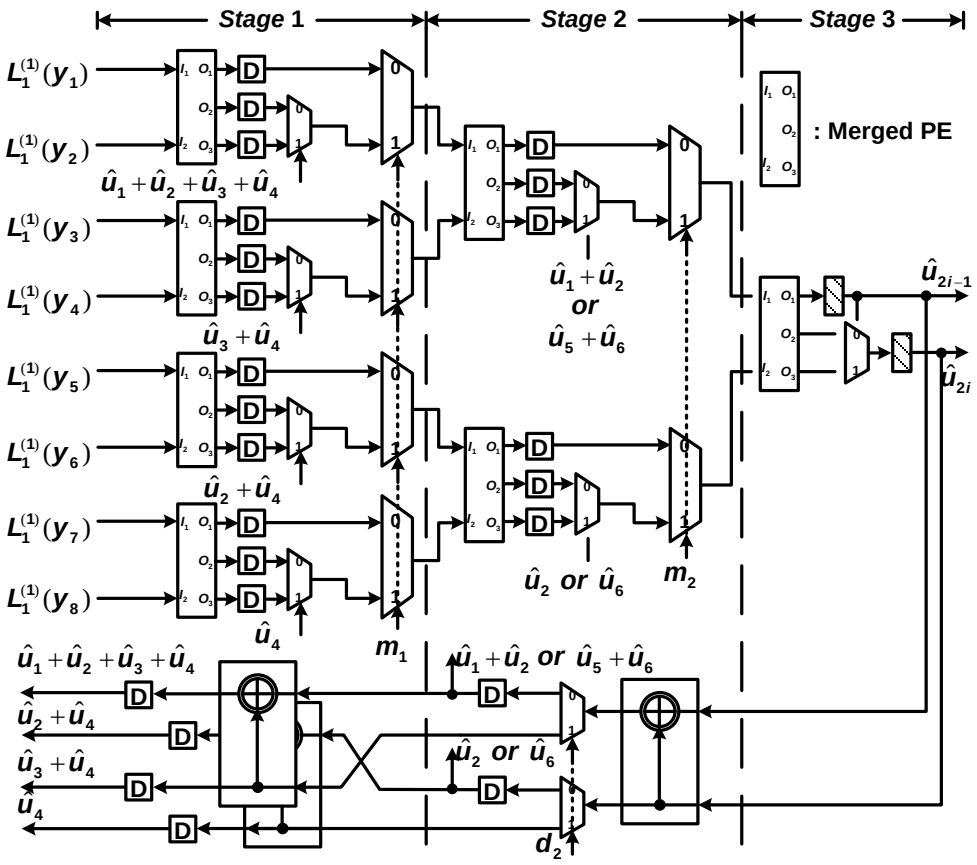


Figure 3.23 8-bit pre-computation look-ahead polar decoder architecture.

3.5.4 Architecture of the Revised Feedback Part

Compared with the one shown in Figure 3.10, only half of the delay elements are employed by the feedback loop in accordance with the halved decoding latency. This is simply because the entire decoding latency of the proposed pre-computation look-ahead sequential decoder has been halved.

The revised *feedback part* for the *N*-bit length pre-computation look-ahead decoder is shown as follows (Figure 3.24). Compared to the one in Figure 3.18, the only difference is the number of delay elements prior to each XOR-PASS element is (2^{n-2}-2) rather than (2^{n-1}-2).

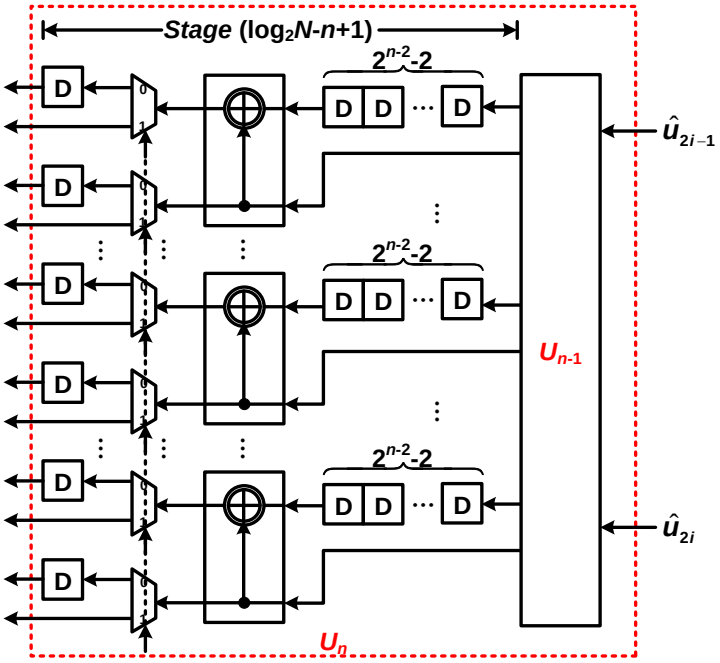


Figure 3.24 Revised recursive construction of U_n based on U_{n-1}.

Note that slight modification of control signal m_i is required. The revised calculation of the switching time is shown in Table 3.6.

Table 3.6 Revised calculation of switching time for m_n.

Weight		Stage index	Clock cycle	m_n
2^2-1	2^1-1			
***Stage* 1 (*n* = 1)**				
0	—	1	2	0
1	—	1	5	1
***Stage* 2 (*n* = 2)**				
0	0	2	3	0
0	1	2	4	1
1	0	2	6	0
1	1	2	7	1

It can be observed that since the total decoding latency has been halved, the switching time for the signal m_i should be revised according to Eq. (3-20). This is because the position of delay elements in the *main frame* is now before the multiplexers rather than after them.

$$c = \sum_j b_j w_j + (i+1). \tag{3-20}$$

For the control signal d_i of de-multiplexers, the same properties addressed in Section 3.4 can be employed to determine the switching time as well. These details of the derivation are omitted here.

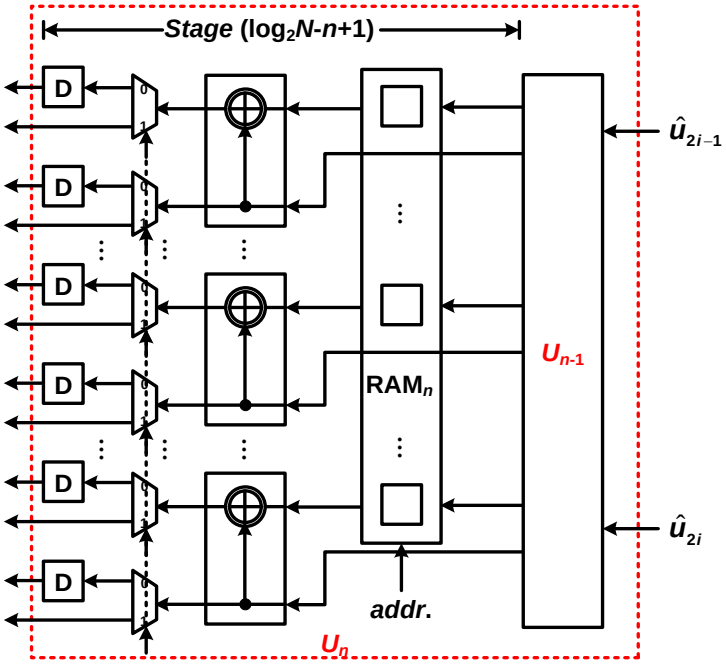


Figure 3.25 Revised recursive construction of U_n using the memory bank.

Here we focus on the optimized architecture for the revised *feedback part*. The previous pipelined *feedback part* structure in Figure 3.24 works best with the proposed pre-computation look-ahead scheme, which enables all intermediate results ($\hat{u}_{2i-1}$) to be generated in place without any extra clock cycles. However, it can be noted that for U_n, the number of corresponding registers increases with complexity of 2^n, which is impractical for polar codes of length over 2^{10}. One possible approach is to employ memory banks instead of flip-flops, which is shown in Figure 3.25. For RAM$_n$, a total of 2^{n-1} memory elements are required. And the data lifetime is (2^{n-2}-2) clock cycles. The memory enable signals (*addr.*) can be determined accordingly.

3.6 Comparison of Latency and Hardware

In this section, we compare the decoding latency and the hardware consumption for the proposed pre-computation look-ahead sequential SC polar decoder along with the state-of-the-art references [28, 92]. Table 3.7 lists the comparison results of those designs in terms of hardware consumption, decoding latency, and data throughput. The design in the 1^{st} column is the implementation of the proposed pre-computation look-ahead sequential decoding schedule in Section 3.5. The design in the 2^{nd} column is the straightforward SC polar decoder design proposed in [28]. The one in the 3^{rd} column is the tree design presented in [92].

The most significant point is that, due to the pre-computation look-ahead decoding scheme, all the proposed architectures require only half the latency of their counterparts. This advantage makes the proposed decoders more attractive for practical implementations, which always demand code length N higher than 2^{10}. This means that we can always achieve twice higher data throughput with the proposed pre-computation approach.

Table 3.7 Comparison for different polar decoder architectures.

Different designs		Proposed pre-computation look-ahead sequential design	Straightforward design [28]	Tree design [92]
Hardware consumption				
# of *merged* PEs		N-1	($N\log_2 N$)/2	N-1
1 PE	XOR	9W	11W-3	
	REG	0	1	
	MUX	6W	5W	
# of *fbk parts*		N-1	—	
1 *fbk part*	XOR	N/2-1	—	
	RAM	N/2-2	—	
	MUX	N/2-2	—	
# of other REGs		W(3N-4)	W($N\log_2 N$)/2	W(N-1)
# of other MUXs		W(2N-3)	0	0
Total	**XOR**	**~17WN**	**~(16W-3)($N\log_2 N$)/2**	**~(16W-3)N**
	REG	**~3WN**	**~(W+1)N($N\log_2 N$)/2**	**~(W+1)N**
Decoding schedule				
Latency		**N-1**	**2(N-1)**	**2(N-1)**
Throughput		**2**	**1**	**1**

The second design point that should be underlined is the proposed *feedback part*. The design of the *feedback part* is inspired by the real FFT processor proposed in [87]. It outputs all control bits (for the pre-computation look-ahead decoders) required by the multiplexers on the fly. Its architecture can be generated in a nice and easy recursive manner. Therefore, with the help of the *feedback part*, no additional clock cycles are needed for computation of $\hat{u}_{2i-1}$, which preserves the advantage of short latency. To the best of our knowledge, this is the first detailed design of the feedback module with such features.

Usually, in order to fairly compare the decoder designs, both throughput and hardware consumption need to be taken into account. In Table 3.7, a detailed hardware architecture comparison for the proposed decoder design has been presented. Meanwhile, since no information could be found in reference [92] for the "$\hat{u}_s$ computation block", which is the counterpart of the proposed *feedback part*, only the hardware consumption for the *main frame* is compared. Despite larger number of registers, which inherently result from the pre-computation look-ahead scheme, the proposed design requires much less hardware compared with the straightforward design proposed in [28]. Even compared with the state-of-the-art design listed in the 3rd column, the proposed design consumes similar hardware area (in terms of number of XOR gates). More specifically, the proposed pre-computation look-ahead sequential design needs 6.25% more hardware than the one presented in [28]. However, considering the latency reduction it can achieve, this small overhead hardware is acceptable. Here, without loss of generality, all data throughputs are represented in a normalized format.

3.7 Conclusion

In this chapter, based on thorough investigation of the DFG for the conventional tree SC polar decoder, a novel pre-computation look-ahead sequential decoder architecture for polar codes, which can halve the decoding latency required by conventional approaches, is proposed.

For efficient hardware implementation, a *merged* PE is presented by using the sub-structure sharing technique. Its control signal $\hat{u}_{2i-1}$ can be generated with a real FFT-like diagram. This unique feature is directly applied to develop an efficient architecture for the *feedback part*, which works best with the low-latency polar decoder architectures. Compared with its conventional counterparts, aside from the *feedback part*, the proposed design shows comparable hardware utilization with 50% decoding latency. The next work will be directed towards designing architectures for even higher throughput communication applications.

Chapter 4

High-Throughput Interleaved SC Polar Decoders

This chapter presents SC polar decoder architectures which are suitable for high-throughput applications. A brief introduction is given in Section 4.1. Section 4.2 presents a TC 3-interleaved SC polar decoder example. A general discussion on the properties of TC interleaved SC decoder architectures [58] is conducted in Section 4.3. Section 4.4 introduces a new design approach using folding technique. Based on the folding technique, the RC interleaved SC decoder architectures [59] are proposed in Section 4.5. Comparison with other works is conducted in Section 4.6. Section 4.7 concludes this chapter.

4.1 Introduction

In Chapter 3, the pre-computation look-ahead approach helps us to reduce the decoding latency of polar decoders by 50%. However, the proposed pre-computation look-ahead serial decoder still needs to be improved for real-time communication applications. It can be noticed that for the proposed pre-computation look-ahead sequential SC decoder

architecture, although the hardware utilization of each active stage is 100%, other stages still remain idle at the same time. For the proposed 8-bit pre-computation look-ahead SC polar decoder example, a total of 7 clock cycles are required before the next code-word can be processed.

Generally, for an N-bit pre-computation look-ahead SC polar decoder, each code-word needs (N-1) clock cycles to be properly decoded with the given pre-computation look-ahead approach. During the entire process no new code-word could be input to the decoder. Therefore, even as high as half of the decoding latency has been reduced by the pre-computation look-ahead SC polar decoder, the data throughput and hardware utilization remain low for large values of N. To this end, the pre-computation TC interleaved polar decoder architectures are proposed in this chapter. For the TC interleaved approach, the decoding latency stays the same, and more PEs can be added.

Another way to improve the decoding throughput is to introduce the RC interleaved processing. The RC interleaved approach always employs the same number of PEs while optimizing the decoding schedule. Compared with the pre-computation polar decoder proposed in [36], the RC 2-interleaved decoder proposed here can achieve 200% throughput with only 50% hardware consumption. Even if the number of processing elements (PEs) required is the same as [94], our design achieves up to four times throughput. Detailed architecture including the control logic is given as well. The general RC L-interleaved polar decoder is also proposed and discussed with respects to both hardware and latency.

4.2 TC 3-Interleaved SC Polar Decoder Example

4.2.1 TC 3-Interleaved Decoding Schedule

As mentioned in Section 4.1, we first try to interleave the decoding processes of multiple code-words. According to the DFG depicted in Figure 3.6, a tentative decoding schedule which triples the decoding throughput for 8-bit polar decoder is given in Table 4.1.

Table 4.1 The TC 3-interleaved decoding schedule for 8-bit decoder.

Stage	**Clock cycle**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
TC 1-interleaved pre-computation decoding schedule								
1	C_1	—	—	—	—	—	—	C_2
2	—	C_1	—	—	C_1	—	—	—
3	—	—	C_1	C_1	—	C_1	C_1	—
TC 3-interleaved pre-computation decoding schedule								
1	C_1	C_2	C_3	—	—	—	—	C_4
2	—	C_1	C_2	C_3	C_1	C_2	C_3	—
3	—	—	C_1	C_2	C_3	C_1	C_2	C_3
3'	—	—	—	C_1	C_2	C_3	C_1	C_2

According to Table 4.1, it can be observed that in order to process three code-words simultaneously, *Stage* 3 has been duplicated. Another copy of *Stage* 3, that is *Stage* 3', has been included to avoid data conflict. As a result, the hardware utilization has been improved. For any *Stage* i (i>1), it is active 6 out of 7 clock cycles. Therefore, the hardware utilization is 85.7%. For *Stage* 1, it is active 3 out of 7 clock cycles. The hardware utilization is 42.9%. On the other hand, for the pre-computation look-ahead sequential design proposed in Chapter 3, the highest hardware utilization is only 57.1% (*Stage* 3). Compared with the sequential design, the TC 3-interleaved one achieves 3 times higher decoding throughput. Moreover, the nice decoding schedule shown in Table 4.1 is very easy to follow.

4.2.2 TC 3-Interleaved SC Decoder Architecture

Based on the pre-computation look-ahead sequential SC decoder we have got previously, the TC 3-interleaved SC polar decoder architecture can be obtained as illustrated in Figure 4.1. It is worth noting that *Stage* 3 and 3', which are activated in serial, are generated by using the unfolding transformation technique [104] with factor of 2. Since the consecutive 3 input code-words are independent, a total of 3 identical copies of the *feedback part* are required as a result. $\{i\}$ means the closure clock cycle index associated with the corresponding point is $7n+i$, where $n \geq 0$.

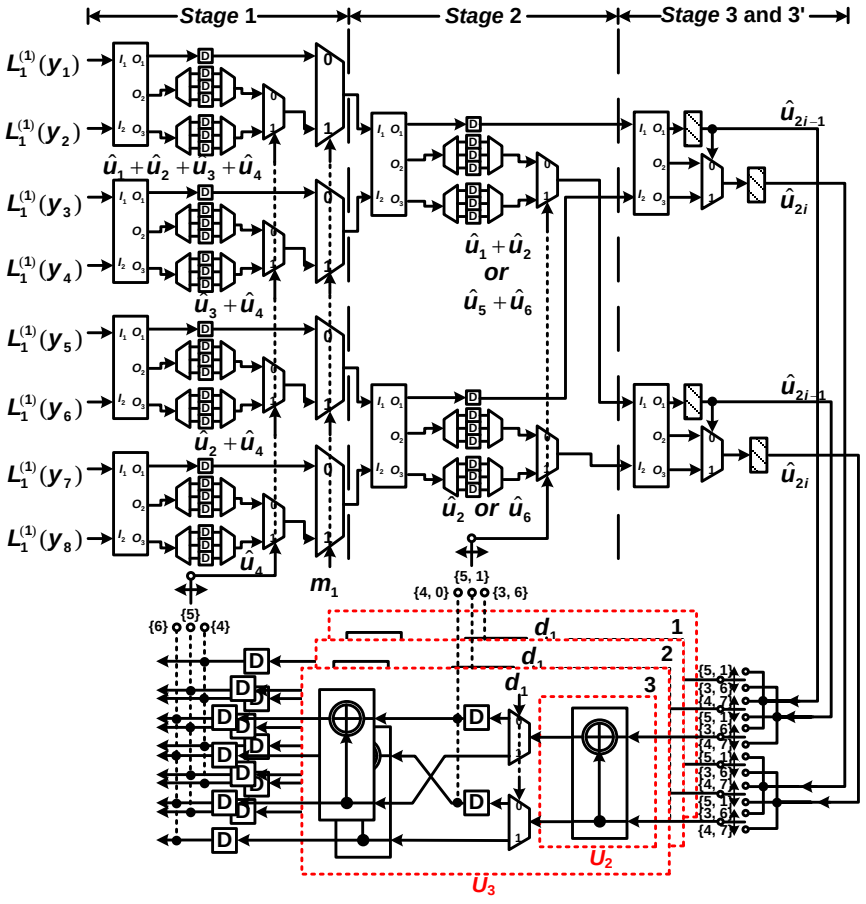


Figure 4.1 The 8-bit TC 3-interleaved SC polar decoder.

4.3 Properties of TC Interleaved SC Decoder Architectures

Although higher decoding throughput has been achieved by the proposed TC 3-interleaved decoder example, the iteration bound of the SC decoder has not been achieved yet. According to the DFG analysis of the pre-computation look-ahead time chart presented in Chapter 3, for an N-bit SC polar decoder, the maximum value of concurrent inputs is (N-1). Therefore, how to systematically construct the TC (N-1)-interleaved SC polar decoder while guaranteeing the sequential decoding process still needs to be addressed.

Moreover, if only TC M-interleaved processing is required, where $1 \leq M \leq N$-1, what is the relationship between the concurrent number M and the hardware consumption? Here the *concurrent number* represents the maximum number of code-words the decoder can process at the same time. The design approach proposed in Section 4.2 can be treated as an empirical design method, which is somewhat case-specific. Can we derive a formal design methodology which only costs the minimal design efforts while meets all the design requirements. All these issues are well addressed with the following four properties.

Property 1 *For an N-bit pre-computation look-ahead polar decoder, the highest concurrent number M is (N-1).*

Proof According to the recursive construction of the pre-computation look-ahead DFG, the decoding latency equals

$$\sum_{i=0}^{\log_2 N-1} 2^i = \frac{2^{\log_2 N}-1}{2-1} = N-1 \tag{4-1}$$

clock cycles. During the entire decoding process for a single code-word, *Stage* 1 is only activated during one clock cycle. Therefore, in the remaining (N-2) clock cycles, *Stage* 1

is available for other possible input code-words. The maximum number of other input code-words is (*N*-2). Therefore, the highest possible concurrent number is (*N*-1).■

Property 2 *For a given N-bit pre-computation look-ahead polar decoder, its TC (2^i-1)-interleaved version can be constructed based on its TC (2^{i-1}-1)-interleaved version. The method is: A. find out the (2^{i-1}-1) stages with the most significant stage indices of the TC (2^{i-1}-1)-interleaved decoder; B. duplicate each of those (2^{i-1}-1) stages*.

Proof It can be noticed that

$$2^i - 1 = 2 \times (2^{i-1} - 1) + 1, \tag{4-2}$$

which is in the same form as the pre-computation look-ahead decoder. In order to achieve maximum hardware utilization of the entire decoder (or certain specific stages), the number of PEs in each relative decoding stage should stay **the same**. Since the DFG is constructed in the time domain, we only need to apply the same approach in the "stage domain", which leads to Property 2.■

A simple example is employed here for a better explanation. For instance, now we would like to construct the TC 3-interleaved version of 8-bit pre-computation look-ahead polar decoder based on the TC 1-interleaved version (or the sequential one proposed in Chapter 3). According to **Property 2**, we have

$$\begin{cases} 3 = 2^2 - 1 \\ 1 = 2^{2-1} - 1. \end{cases} \tag{4-3}$$

Therefore, we can determine that $i = 2$. We only need to include another copy of *Stage* 3 (that is *Stage* 3') to get the decoder architecture illustrated in Figure 4.1.

Table 4.2 The TC 1-, 3-, and 7-interleaved decoding schedule for 8-bit decoder.

Stage	Clock cycle													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
TC 1-interleaved pre-computation decoding schedule														
1	C_1	—	—	—	—	—	—	C_2	—	—	—	—	—	—
2	—	C_1	—	—	C_1	—	—	—	C_2	—	—	C_2	—	—
3	—	—	C_1	C_1	—	C_1	C_1	—	—	C_2	C_2	—	C_2	C_2
TC 3-interleaved pre-computation decoding schedule														
1	C_1	C_2	C_3	—	—	—	—	C_4	C_5	C_6	—	—	—	—
2	—	C_1	C_2	C_3	C_1	C_2	C_3	—	C_4	C_5	C_6	C_4	C_5	C_6
3	—	—	C_1	C_2	C_3	C_1	C_2	C_3	—	C_4	C_5	C_6	C_4	C_5
3’	—	—	—	C_1	C_2	C_3	C_1	C_2	C_3	—	C_4	C_5	C_6	C_4
TC 7-interleaved pre-computation decoding schedule														
1	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…						
2	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…					
3	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…				
3’	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…			
2’	—	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…		
3’’	—	—	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…	
3’’’	—	—	—	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…

Moreover, its TC 7-interleaved version, which can achieve 100% hardware utilization in each decoding stage, can be constructed based on the TC 3-interleaved one accordingly as shown in Table 4.2. The construction steps are shown as follows. According to **Property 2**, since

$$\begin{cases} 7=2^3-1 \\ 3=2^{3-1}-1, \end{cases} \tag{4-4}$$

We have $i = 3$. For the TC 3-interleaved version decoder, the 3 stages with the most significant indices are *Stage* 2, 3, and 3'. Therefore, the architecture of the TC 7-interleaved pre-computation look-ahead SC polar decoder can be derived based on the TC 3-interleaved one by duplicating *Stage* 2, 3, and 3'. According to Table 4.2, it can be seen that the proposed TC 7-interleaved decoder can handle all 7 inputs perfectly and achieve 100% utilization rate during any *Decoding iteration i* (i>1). Also the iteration bound is achieved as a result.

Since all the interleaving factors mentioned now can be represented in the form of (2^i-1), what if when we are required to implement a TC M-interleaved decoder with M which could not be written in the form of (2^i-1)? This problem can be addressed by **Property 3** as follows.

Property 3 *For any M which satisfies* $2^{i-1}-1<M\leq 2^i-1$*, the TC M-interleaved polar decoder requires the same amount of hardware as the TC* (2^i-1)*-interleaved version. A 100% hardware utilization can be achieved if and only if M = N-1.*

Proof The proof of **Property 2** can be used to prove this. According to **Property 2**, it can be noted that (2^i-1) is the maximum number of code-words the TC (2^i-1)-interleaved version decoder can handle. This gives the proof.■

Table 4.3 The TC 3-, 5-, and 7-interleaved decoding schedule for 8-bit decoder.

Stage	Clock cycle													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
TC 3-interleaved pre-computation decoding schedule														
1	C_1	C_2	C_3	—	—	—	—	C_4	C_5	C_6	—	—	—	—
2	—	C_1	C_2	C_3	C_1	C_2	C_3	—	C_4	C_5	C_6	C_4	C_5	C_6
3	—	—	C_1	C_2	C_3	C_1	C_2	C_3	—	C_4	C_5	C_6	C_4	C_5
3'	—	—	—	C_1	C_2	C_3	C_1	C_2	C_3	—	C_4	C_5	C_6	C_4
TC 5-interleaved pre-computation decoding schedule														
1	C_1	C_2	C_3	C_4	C_5	—	—	C_6	C_7	C_8	C_9	C_{10}	—	—
2	—	C_1	C_2	C_3	C_4	C_5	—	—	C_6	C_7	C_8	C_9	C_{10}	—
3	—	—	C_1	C_2	C_3	C_4	C_5	—	—	C_6	C_7	C_8	C_9	C_{10}
3'	—	—	—	C_1	C_2	C_3	C_4	C_5	—	—	C_6	C_7	C_8	C_9
2'	—	—	—	—	C_1	C_2	C_3	C_4	C_5	—	—	C_6	C_7	C_8
3''	—	—	—	—	—	C_1	C_2	C_3	C_4	C_5	—	—	C_6	C_7
3'''	—	—	—	—	—	—	C_1	C_2	C_3	C_4	C_5	—	—	C_6
TC 7-interleaved pre-computation decoding schedule														
1	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…						
2	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…					
3	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…				
3'	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…			
2'	—	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…		
3''	—	—	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…	
3'''	—	—	—	—	—	—	C_1	C_2	C_3	C_4	C_5	C_6	C_7	…

A simple example in Table 4.3 clearly shows that the TC 5-interleaved version consumes the same number of PEs as the TC 7-interleaved decoder does.

Now, we are ready to derive the relationship between the concurrent number *M* and the hardware consumption.

Property 4 *For any M which satisfies $2^{i-1}-1<M\leq 2^{i}-1$, the hardware consumption stays the same. The total number of merged PEs employed by the M-concurrent polar decoder is $N+2^{i-1}\cdot(i-2)$, where N is the code length of the polar codes.*

Proof According to **Property 1**, the number of *merged* PEs can be calculated as follows:

$$\begin{aligned}&\sum_{j=i-1}^{\log_2 N-1} 2^{j}+2^{i-1}\cdot(i-1)\\&=(N-2^{i-1})+2^{i-1}\cdot(i-1)\\&=N+2^{i-1}\cdot(i-2).\blacksquare\end{aligned} \tag{4-5}$$

For example, according to the calculations given by **Property 4**, the TC 1-, 3-, 5-, and 7-interleaved versions of 8-bit pre-computation look-ahead polar decoder are supposed to have 7, 8, 12, and 12 *merged* PEs, respectively. By checking the details shown in both

Table 4.2 and Table 4.3, those results can be easily verified.

With the four properties proposed in this chapter, we now have a systematic methodology to design TC interleaved versions of pre-computation look-ahead SC polar decoders for various application requirements.

4.4 The Construction Approach with Folding Technique

The folding transformation provides a systematic technique for designing control circuits where multiple algorithm operations are time-multiplexed to a single function unit [37].

The folding transformation technique can help us derive a formal procedure for designing a family of architectures for a specific DSP algorithm [105, 106]. In this section, we will see, with the help of the folding transformation technique, existing hardware architectures can be implemented systematically.

4.4.1 Preliminaries of Folding Transformation Technique

To implement the folding transformation upon the DFG, a folding set is required first. Each folding set is an ordered set of *n* operations executed by the same PE [37]. The operation in the *i*-th position in the folding set is executed by the PE during the *i*-th time instance. For example, according to the folding set $A=\{A_1, A_2, \phi, A_3\}$, we have *n* = 4. Operations A_1, A_2, and A_3 will be carried out by PE A at clock cycle 4*l*+1, 4*l*+2, and 4*l*, respectively. Since the 3rd operation within the folding set is null, no operation will be executed by PE A during clock cycle 4*l*+3.

Aside from the folding set, another key factor we should be aware of is the folding equation. The folding equation is employed to calculate the number of delay elements associated with a specific directed edge in the DFG. The general form of the *n*-folding equation can be written as follows:

$$D_F(U \xrightarrow{e} V) = nw(e) - P_U + v - u \tag{4-6}$$

Here, *e* is the *w*(*e*) weighted directed edge connecting PE *U* and PE *V*. $D_F(U \xrightarrow{e} V)$ is the number of folding delay elements associated with edge *e*. PE *U* is P_U-stage pipelined. *u* and *v* are the folding set indices of PE *U* and PE *V*, respectively. About the derivation of the folding equation, please refer to Chapter 6 of [105] for more details.

4.4.2 Previous Decoders with Folding Transformation Technique

In this sub-section, we will derive the previous SC polar decoder architectures based on the folding transformation technique.

Tree Architecture of the SC Decoder

First, let us re-visit the DFG illustrated in Figure 3.5. The SC decoder architecture proposed in [28] can be treated as the original version which has not been folded yet. Then, the tree architecture of the SC polar decoder in [92] can be constructed with the following folding sets:

$$\begin{cases} A_1^f = \{A_1,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi\}, \\ A_2^f = \{A_2,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi\}, \\ A_3^f = \{A_3,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi\}, \\ A_4^f = \{A_4,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi\}; \end{cases} \quad \begin{cases} B_1^f = \{\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,B_1,\phi,\phi,\phi,\phi,\phi,\phi\}, \\ B_2^f = \{\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,B_2,\phi,\phi,\phi,\phi,\phi,\phi\}, \\ B_3^f = \{\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,B_3,\phi,\phi,\phi,\phi,\phi,\phi\}, \\ B_4^f = \{\phi,\phi,\phi,\phi,\phi,\phi,\phi,\phi,B_4,\phi,\phi,\phi,\phi,\phi,\phi\}. \end{cases} \quad (4\text{-}7)$$

$$\begin{cases} C_1^f = \{\phi,C_1,\phi,\phi,\phi,\phi,\phi,\phi,C_3,\phi,\phi,\phi,\phi,\phi,\phi\}; \\ C_2^f = \{\phi,C_2,\phi,\phi,\phi,\phi,\phi,\phi,C_4,\phi,\phi,\phi,\phi,\phi,\phi\}. \end{cases} \quad \begin{cases} D_1^f = \{\phi,\phi,\phi,\phi,D_1,\phi,\phi,\phi,\phi,\phi,\phi,\phi,D_3,\phi,\phi\}; \\ D_2^f = \{\phi,\phi,\phi,\phi,D_2,\phi,\phi,\phi,\phi,\phi,\phi,\phi,D_4,\phi,\phi\}. \end{cases}$$

$$E_1^f = \{\phi,\phi,E_1,\phi,\phi,E_2,\phi,\phi,\phi,E_3,\phi,\phi,E_4,\phi\}. \quad F_1^f = \{\phi,\phi,\phi,F_1,\phi,\phi,F_2,\phi,\phi,\phi,F_3,\phi,\phi,F_4\}.$$

Here, the superscript "f" means "folding". Assume neither Type I PE nor Type II PE has any pipeline stages and $n = 14$. We can write down the folding equation in Eq. (4-6) for each of the 28 edges in the DFG (Figure 3.5). The equations are:

$$\begin{array}{ll} D_F(A_{1,2} \to C_1) = 1, & D_F(F_1 \to D_{1,2}) = 1, \\ D_F(A_{3,4} \to C_2) = 1, & D_F(D_{1,2} \to E_2) = 1, \\ D_F(C_{1,2} \to E_1) = 1, & D_F(E_2 \to F_2) = 1, \\ D_F(E_1 \to F_1) = 1, & D_F(F_2 \to B_{1,2,3,4}) = 1; \\ \hline D_F(B_{1,2} \to C_3) = 1, & D_F(F_3 \to D_{3,4}) = 1, \\ D_F(B_{3,4} \to C_4) = 1, & D_F(D_{3,4} \to E_4) = 1, \\ D_F(C_{3,4} \to E_3) = 1, & D_F(E_4 \to F_4) = 1. \\ D_F(E_3 \to F_3) = 1, & \end{array} \quad (4\text{-}8)$$

For example, $D_F(A_{1,2} \to C_1) = 1$ means that the outputs of both A_1^f and A_2^f will be hold for 1 clock cycle before they are output to node C_1^f .Therefore, the tree architecture proposed by Figure 4 of [92] can be derived accordingly.

Pre-Computation Look-Ahead Sequential Decoder Architecture

Now we derive the pre-computation look-ahead sequential decoder architecture with the folding transformation technique. Consider the following folding sets:

$$\begin{cases} (AB)_1^f = \{(AB)_1, \phi, \phi, \phi, \phi, \phi, \phi\}, \\ (AB)_2^f = \{(AB)_2, \phi, \phi, \phi, \phi, \phi, \phi\}, \\ (AB)_3^f = \{(AB)_3, \phi, \phi, \phi, \phi, \phi, \phi\}, \\ (AB)_4^f = \{(AB)_4, \phi, \phi, \phi, \phi, \phi, \phi\}. \end{cases} \tag{4-9}$$

$$\begin{cases} (CD)_1^f = \{\phi, (CD)_1, \phi, \phi, (CD)_3, \phi, \phi\}; \\ (CD)_2^f = \{\phi, (CD)_2, \phi, \phi, (CD)_4, \phi, \phi\}. \end{cases}$$

$$(EF)_1^f = \{\phi, \phi, (EF)_1, (EF)_2, \phi, (EF)_3, (EF)_4\}.$$

Again, suppose neither Type I nor Type II PEs has any pipeline stages and $n = 7$. The folding equations are as follows:

$$\begin{array}{ll} D_F(A_{1,2} \to C_1) = 1, & D_F(F_1 \to D_{1,2}) = -1, \\ D_F(A_{3,4} \to C_2) = 1, & D_F(D_{1,2} \to E_2) = 2, \\ D_F(C_{1,2} \to E_1) = 1, & D_F(E_2 \to F_2) = 0, \\ D_F(E_1 \to F_1) = 0, & D_F(F_2 \to B_{1,2,3,4}) = -3; \\ \hline D_F(B_{1,2} \to C_3) = 4, & D_F(F_3 \to D_{3,4}) = -1, \\ D_F(B_{3,4} \to C_4) = 4, & D_F(D_{3,4} \to E_4) = 2, \\ D_F(C_{3,4} \to E_3) = 1, & D_F(E_4 \to F_4) = 0. \\ D_F(E_3 \to F_3) = 0, & \end{array} \tag{4-10}$$

For example, $D_F(B_{1,2} \to C_3) = 4$ means that the outputs of both B_1^f and B_2^f will be hold for 4 clock cycles before they are output to node C_1^f . $D_F(F_2 \to B_{1,2,3,4}) = -3$ indicates that the outputs of nodes $B_{1,2,3,4}^f$ will be hold for 3 clock cycles until the output of F_1^f arrives. The pre-computation decoder can therefore be realized in Figure 3.10.

Pre-Computation TC Interleaved Decoder Architecture

For the efficient TC interleaved decoder architectures in Section 4.2 and Section 4.3, they can also be derived by changing the folding sets. We know that the TC interleaved pre-computation look-ahead decoder architectures are categorized into a big family. Without loss of generality, here we only show the construction details for the TC 3-interleaved decoder architecture. Other TC interleaved versions can be derived in a similar fashion.

Consider the new folding sets listed as follows, where the superscripts "1", "2", and "3" mean the PE is dealing with Codeword 1, Codeword 2, and, Codeword 3, respectively.

$$\begin{cases} (AB)_1^f = \{(AB)_1^{(1)}, (AB)_1^{(2)}, (AB)_1^{(3)}, \phi, \phi, \phi, \phi\}, \\ (AB)_2^f = \{(AB)_2^{(1)}, (AB)_2^{(2)}, (AB)_2^{(3)}, \phi, \phi, \phi, \phi\}, \\ (AB)_3^f = \{(AB)_3^{(1)}, (AB)_3^{(2)}, (AB)_3^{(3)}, \phi, \phi, \phi, \phi\}, \\ (AB)_4^f = \{(AB)_4^{(1)}, (AB)_4^{(2)}, (AB)_4^{(3)}, \phi, \phi, \phi, \phi\}. \end{cases}$$

$$\begin{cases} (CD)_1^f = \{\phi, (CD)_1^{(1)}, (CD)_1^{(2)}, (CD)_1^{(3)}, (CD)_3^{(1)}, (CD)_3^{(2)}, (CD)_3^{(3)}\}; \\ (CD)_2^f = \{\phi, (CD)_2^{(1)}, (CD)_2^{(2)}, (CD)_2^{(3)}, (CD)_4^{(1)}, (CD)_4^{(2)}, (CD)_4^{(3)}\}. \end{cases} \quad (4\text{-}11)$$

$$\begin{cases} (EF)_1^f = \{(EF)_3^{(3)}, \phi, (EF)_1^{(1)}, (EF)_1^{(2)}, (EF)_1^{(3)}, (EF)_3^{(1)}, (EF)_3^{(2)}\}, \\ (EF)_1^f = \{(EF)_4^{(2)}, (EF)_4^{(3)}, \phi, (EF)_2^{(1)}, (EF)_2^{(2)}, (EF)_2^{(3)}, (EF)_4^{(1)}\}. \end{cases}$$

Here we can also derive the hardware utilization from the folding sets. Take the following folding set as an example:

$$(EF)_1^f = \{(EF)_3^{(3)}, \phi, (EF)_1^{(1)}, (EF)_1^{(2)}, (EF)_1^{(3)}, (EF)_3^{(1)}, (EF)_3^{(2)}\}. \quad (4\text{-}12)$$

It can be observed from the folding set that only 1 out of 7 clock cycle is occupied by the null operation. Therefore, the hardware utilization of *Stage* 3' is 85.7%, which matches the previous analysis in Section 4.2. The utilization of other stages can be derived in the same manner.

The corresponding folding equations are given as follows. With these folding equations, the TC 3-interleaved pre-computation look-ahead decoder can be constructed:

$$\begin{array}{ll}
D_F(A_{1,2}^{(1,2,3)} \to C_1^{(1,2,3)}) = 1, & D_F(F_1^{(1,2,3)} \to D_{1,2}^{(1,2,3)}) = 1, \\
D_F(A_{3,4}^{(1,2,3)} \to C_2^{(1,2,3)}) = 1, & D_F(D_{1,2}^{(1,2,3)} \to E_2^{(1,2,3)}) = 1, \\
D_F(C_{1,2}^{(1,2,3)} \to E_1^{(1,2,3)}) = 1, & D_F(E_2^{(1,2,3)} \to F_2^{(1,2,3)}) = 1, \\
D_F(E_1^{(1,2,3)} \to F_1) = 1, & D_F(F_2^{(1,2,3)} \to B_{1,2,3,4}^{(1,2,3)}) = 1; \\
\hline
D_F(B_{1,2}^{(1,2,3)} \to C_3^{(1,2,3)}) = 1, & D_F(F_3^{(1,2,3)} \to D_{3,4}^{(1,2,3)}) = 1, \\
D_F(B_{3,4}^{(1,2,3)} \to C_4^{(1,2,3)}) = 1, & D_F(D_{3,4}^{(1,2,3)} \to E_4^{(1,2,3)}) = 1, \\
D_F(C_{3,4}^{(1,2,3)} \to E_3^{(1,2,3)}) = 1, & D_F(E_4^{(1,2,3)} \to F_4^{(1,2,3)}) = 1. \\
D_F(E_3^{(1,2,3)} \to F_3^{(1,2,3)}) = 1, &
\end{array} \tag{4-13}$$

4.5 The RC Interleaved Pre-Computation Decoders

In Section 4.2 and Section 4.3, the TC interleaved decoder architectures are proposed. However, in order to achieve higher decoder throughput, the TC interleaved decoders required higher hardware consumption. Therefore, the decoder architecture with higher data rate and lower hardware cost is appreciated. To this end, the RC interleaved pre-computation look-ahead decoder architecture is present based on the folding transformation technique. In order to save the hardware consumption, for *N*-bit RC interleaved decoders, the number of *merged* PEs employed is always *N*/2. Also, in order to increase the data rate, we try to process multiple code-words in an interleaved manner.

4.5.1 The Original Design Employing *N*/2 *Merged* PEs

Now we construct the folding sets for the original 8-bit SC polar decoder which only employs 4 *merged* PEs:

$$\begin{cases} (AB)_1^f = \{(AB)_1, (CD)_1, (EF)_1, (EF)_2, (CD)_3, (EF)_3, (EF)_4\}, \\ (AB)_2^f = \{(AB)_2, (CD)_2, \phi, \phi, (CD)_4, \phi, \phi\}, \\ (AB)_3^f = \{(AB)_3, \phi, \phi, \phi, \phi, \phi, \phi\}, \\ (AB)_4^f = \{(AB)_4, \phi, \phi, \phi, \phi, \phi, \phi\}. \end{cases} \tag{4-14}$$

The decoding schedule of the original decoder design employing 4 *merged* PEs is listed in Table 4.4 as follows:

Table 4.4 Number of active *merged* PEs for the original decoder.

PE	Clock cycle						
	1	2	3	4	5	6	7
1							
2			—	—		—	—
3		—	—	—	—	—	—
4		—	—	—	—	—	—

4.5.2 The RC 2-Interleaved Decoders

According to the decoding schedule of the original decoder design, it is observed that we can use the same number of *merged* PEs to carry out the decoding of two code-words at the same time. Now we construct the folding sets for the 8-bit RC 2-interleaved SC polar decoder as follows:

$$\begin{cases} (AB)_1^f = \{(AB)_1^{(1)}, (AB)_1^{(2)}, (CD)_1^{(1)}, (EF)_1^{(1)}, (EF)_2^{(1)}, (CD)_3^{(1)}, (EF)_3^{(1)}, (EF)_4^{(1)}\}, \\ (AB)_2^f = \{(AB)_2^{(1)}, (AB)_2^{(2)}, (CD)_2^{(1)}, \phi, \phi, (CD)_4^{(1)}, \phi, \phi\}, \\ (AB)_3^f = \{(AB)_3^{(1)}, (AB)_3^{(2)}, (CD)_1^{(2)}, \phi, \phi, (CD)_3^{(2)}, \phi, \phi\}, \\ (AB)_4^f = \{(AB)_4^{(1)}, (AB)_4^{(2)}, (CD)_2^{(2)}, (EF)_1^{(2)}, (EF)_2^{(2)}, (CD)_4^{(2)}, (EF)_3^{(2)}, (EF)_4^{(2)}\}. \end{cases} \tag{4-15}$$

Again, it is assumed that there are no pipelines within any PE. The folding factor n is set to 8. By applying Eq. (4-6), we have the following folding equations for the new folding sets:

$$
\begin{array}{ll}
D_F(A_{1,2}^{(1)} \to C_1^{(1)}) = 2, & D_F(F_1^{(1)} \to D_{1,2}^{(1)}) = -1, \\
D_F(A_{3,4}^{(1)} \to C_2^{(1)}) = 2, & D_F(D_{1,2}^{(1)} \to E_2^{(1)}) = 2, \\
D_F(C_{1,2}^{(1)} \to E_1^{(1)}) = 1, & D_F(E_2^{(1)} \to F_2^{(1)}) = 0, \\
D_F(E_1^{(1)} \to F_1^{(1)}) = 0, & D_F(F_2^{(1)} \to B_{1,2,3,4}^{(1)}) = -4; \\
\hline
D_F(B_{1,2}^{(1)} \to C_3^{(1)}) = 5, & D_F(F_3^{(1)} \to D_{3,4}^{(1)}) = -1, \\
D_F(B_{3,4}^{(1)} \to C_4^{(1)}) = 5, & D_F(D_{3,4}^{(1)} \to E_4^{(1)}) = 2, \\
D_F(C_{3,4}^{(1)} \to E_3^{(1)}) = 1, & D_F(E_4^{(1)} \to F_4^{(1)}) = 0. \\
D_F(E_3^{(1)} \to F_3^{(1)}) = 0, & \\
\hline
D_F(A_{1,2}^{(2)} \to C_1^{(2)}) = 1, & D_F(F_1^{(2)} \to D_{1,2}^{(2)}) = -1, \\
D_F(A_{3,4}^{(2)} \to C_2^{(2)}) = 1, & D_F(D_{1,2}^{(2)} \to E_2^{(2)}) = 2, \\
D_F(C_{1,2}^{(2)} \to E_1^{(2)}) = 1, & D_F(E_2^{(2)} \to F_2^{(2)}) = 0, \\
D_F(E_1^{(2)} \to F_1^{(2)}) = 0, & D_F(F_2^{(2)} \to B_{1,2,3,4}^{(2)}) = -3; \\
\hline
D_F(B_{1,2}^{(2)} \to C_3^{(2)}) = 4, & D_F(F_3^{(2)} \to D_{3,4}^{(2)}) = -1, \\
D_F(B_{3,4}^{(2)} \to C_4^{(2)}) = 4, & D_F(D_{3,4}^{(2)} \to E_4^{(2)}) = 2, \\
D_F(C_{3,4}^{(2)} \to E_3^{(2)}) = 1, & D_F(E_4^{(2)} \to F_4^{(2)}) = 0. \\
D_F(E_3^{(2)} \to F_3^{(2)}) = 0, &
\end{array}
\tag{4-16}
$$

According to the new folding sets in Eq. (4-15), the RC 2-interleaved SC polar decoder architecture employing four *merged* PEs is illustrated in Figure 4.2. Since the routing for the RC 2-interleaved decoder architecture is a little bit complicated, different colors are employed for better identification. The control signals of the decoder architecture can be determined according to the folding sets. The switching schedule of all the control signals is listed in Table 4.5 as follows:

Table 4.5 Control signals for the RC 2-interleaved decoder.

Signal	**Clock cycle**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{m_1}$	—	—	0	0	1	1	0	1
$\boldsymbol{m_2}$	—	—	0	—	—	1	—	—
$\boldsymbol{d_1}$	—	—	—	0	1	—	—	—

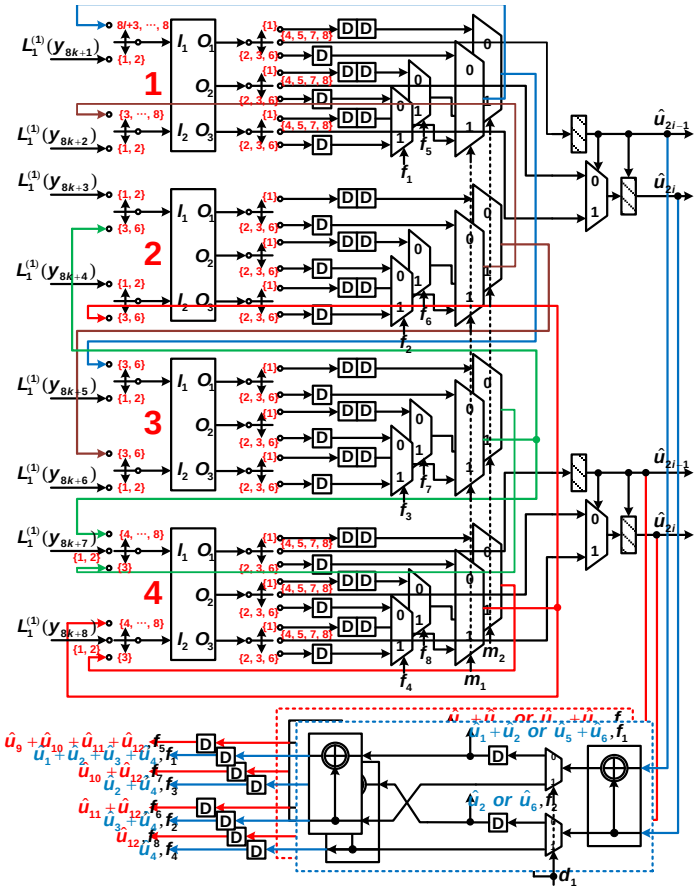


Figure 4.2 The RC 2-interleaved architecture for 8-bit polar decoder.

For easier understanding of the decoding schedule of the proposed RC 2-interleaved polar decoder, the four *merged* PEs are marked with red labels. The number of active PE(s) in each clock cycle is shown in Table 4.6.

Table 4.6 Number of active *merged* PEs for RC 2-interleaved decoder.

PE	Clock cycle							
	1	2	3	4	5	6	7	8
1	■	■	■	■	■	■	■	■
2	■	■	■	—	—	■	—	—
3	■	■	■	—	—	■	—	—
4	■	■	■	■	■	■	■	■

■: dealing with Codeword 1;
■: dealing with Codeword 2.

Indicated by the cardinality of the folding sets, in order to complete the decoding process of both code-words, a total of 8 clock cycles are required. Compared with the conventional pre-computation look-ahead decoding process, an additional clock cycle is required. In general, the *N*-bit RC 2-interleaved SC decoder's decoding latency equals *N* clock cycles. For real-time communication applications, the code length *N* is considerable large, the one clock cycle penalty is negligible. At the same time, its advantages are obvious: the decoding throughput has been doubled while the hardware cost is only 50% as before.

In general, to construct an *L*-interleaved *N*-bit polar decoder architecture, the brute-force way is to simply duplicate the proposed RC 2-interleaved decoder $\lceil L/2 \rceil$ times. Therefore, a total of $\frac{N}{2} \cdot \lceil L/2 \rceil$ *merged* PEs and *N* clock cycles are required. In this case, the decoding latency will remain as *N* clock cycles.

However, in our design situation, since the resources are constrained, we have to stick with $\frac{N}{2}$ *merged* PEs. To avoid too large decoding latency, further optimization is required. According to the DFG analysis in Section 3.3, for the pre-computation look-ahead sequential decoder in Figure 3.10, *Stage i* is associated with 2^{i-1} clock cycles' latency.

Compare the proposed RC 2-interleaved decoder architecture in Figure 4.2 with the one illustrated in Figure 3.10. If we are trying to process L code-words in an interleaved manner using the same architecture in Figure 4.2, the resulting latency associated with *Stage i* in Figure 3.10 is $2^{i-1} \cdot \lceil L/2^{i-1} \rceil$ clock cycles. Therefore, the entire decoding latency for this RC L-interleaved decoder is:

$$T = \sum_{i=1}^{\log_2 N} 2^{i-1} \cdot \lceil L/2^{i-1} \rceil. \tag{4-17}$$

Let us see some examples. First, let $L = 2$ and $N = 8,$ we have the decoding latency for the RC 2-interleaved 8-bit polar decoder architecture is

$$\begin{aligned} T_{(8,2)} &= \sum_{i=1}^{\log_2 8} 2^{i-1} \cdot \lceil 2/2^{i-1} \rceil \\ &= \sum_{i=1}^{3} 2^{i-1} \cdot \lceil 2/2^{i-1} \rceil \\ &= 2^0 \cdot \lceil 2/2^0 \rceil + 2^1 \cdot \lceil 2/2^1 \rceil + 2^2 \cdot \lceil 2/2^2 \rceil \\ &= 2 + 2 + 4 \\ &= 8, \end{aligned} \tag{4-18}$$

which matches the result shown in Table 4.6.

Then, we try to calculate the decoding latency for the RC 3-interleaved example. Set $L = 3$ and $N = 8,$ we have the decoding latency is

$$\begin{aligned} T_{(8,3)} &= \sum_{i=1}^{\log_2 8} 2^{i-1} \cdot \lceil 3/2^{i-1} \rceil \\ &= \sum_{i=1}^{3} 2^{i-1} \cdot \lceil 3/2^{i-1} \rceil \\ &= 2^0 \cdot \lceil 3/2^0 \rceil + 2^1 \cdot \lceil 3/2^1 \rceil + 2^2 \cdot \lceil 3/2^2 \rceil \\ &= 3 + 4 + 4 \\ &= 11. \end{aligned} \quad \text{s} \tag{4-19}$$

On the other hand, the decoding schedule of the RC 3-interleaved decoder can be described by Table 4.7 as follows:

Table 4.7 Number of active *merged* PEs for RC 3-interleaved decoder.

PE	Clock cycle										
	1	2	3	4	5	6	7	8	9	10	11
1	■	■	■	■	■	■	■	■	■	■	■
2	■	■	■	■	■	■	■	■	■	■	■
3	■	■	■	■	—	—	—	■	—	—	—
4	■	■	■	■	—	■	■	■	—	■	■

■: dealing with Codeword 1;
■: dealing with Codeword 2;
■: dealing with Codeword 3.

Again, we have two results match with each other. However, in real applications the RC 2-interleaved pre-computation look-ahead decoder is the most favorable one. Because it can achieve good balance of both decoding throughput and hardware cost.

4.6 Comparison with Other Works

In this section, we conduct a comparison with the state-of-the-art designs. Both the pre-computation look-ahead TC interleaved decoder design and the RC interleaved decoder design proposed in this chapter are included in the comparison. For the TC M-interleaved polar decoder, it is assumed that $2^{i-1}-1<M\leq 2^{i}-1$. For the RC interleaved polar decoder architecture, the 2-interleaved version is compared here. The previous decoders which have already been mentioned in Table 3.7 are not compared here. The other designs included in Table 4.8 are the conventional overlapped decoder in [92], the line decoder in [94], and the pre-computation look-ahead sequential decoder proposed in Section 3.5. All the design details are listed in Table 4.8. The same assumptions on the hardware conversion and throughput estimation, which are used in Table 3.7, are also adopted here. Also, since the *merged* PE architecture we proposed in Chapter 3 consumes similar hardware compared with the conventional one, no more gate count details are provided here.

Table 4.8 Comparison between the two proposed works and others.

Different designs	**Number of *merged* PEs**	**Number of *feedback parts***	**Latency**	**Throughput**
Pre-computation TC interleaved design	$N+2^{i-1}\cdot(i-2)$	M	N-1	$2M$
Pre-computation RC 2-interleaved design	$N/2$	2	N	4
Pre-computation look-ahead sequential design	N-1	1	N-1	2
Overlapped design [92]	$\sim N+M(\log_2 M/2)/2$	—	$2(N-1)$	1
Line design [94]	$N/2$	—	$2(N-1)$	1

First, let us focus on the pre-computation look-ahead TC *M*-interleaved decoder. Compared with the line decoder and the pre-computation look-ahead sequential decoder, this design shows significant improvement in data throughput. Also, the proposed TC interleaved one can achieve the highest throughput and hardware utilization (100%) when $M = N\text{-}1$. If compare the pre-computation look-ahead TC interleaved decoder with the overlapped one given in [92], our design can achieve twice data throughput with similar hardware cost.

Table 4.9 The TC 2-interleaved decoding schedule for 8-bit decoder.

Stage	**Clock cycle**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
TC 3-interleaved pre-computation decoding schedule								
1	C_1	C_2	—	—	—	—	—	C_3
2	—	C_1	C_2	—	C_1	C_2	—	—
3	—	—	C_1	C_2	—	C_1	C_2	—
3'	—	—	—	C_1	C_2	—	C_1	C_2

A recent publication has report the low hardware-complexity SC polar decoder called line decoder. However, if we compare the proposed RC 2-interleaved SC polar decoder with it, the advantages of our design is obvious. The RC 2-interleaved decoder can achieve 4 times decoding throughput while only consume the same hardware cost. If we compare the RC 2-interleaved decoder with the pre-computation look-ahead sequential decoder, it can achieve 200% throughput improvement and 50% hardware reduction. From the previous analysis, we know that the TC interleaved design proposed in Section 4.2 can achieve high hardware utilization. For the TC 2-interleaved polar decoder (shown in Table 4.9), the hardware utilization of the entire decoder can be calculated as follows:

$$\frac{\frac{4}{7}\times 2+\frac{4}{7}\times 2+\frac{2}{7}\times 4}{8} \doteq 42.9\%. \tag{4-20}$$

However, according to Table 4.6, the hardware utilization for the RC 2-interleaved is 75%. That means the proposed RC 2-interleaved polar decoder architecture can attain

even higher utilization than its TC interleaved counterpart. Low latency, high throughput, and high utilization make the RC 2-interleaved SC polar decoder very attractive for real-time applications.

4.7 Conclusion

In this chapter, two kinds of interleaved SC polar decoder architectures are proposed. Compared with the pre-computation look-ahead sequential decoder, they can achieve even higher throughput. With the properties introduced, the pre-computation look-ahead TC interleaved decoder family can be constructed in a systematic manner. With the overlapping scheme, the hardware utilization can be as high as 100%. Also, the RC interleaved SC decoder family is presented. A formal design methodology is introduced as well. Comparison results have shown that the RC 2-interleaved pre-computation look-ahead decoder can achieve 4 times decoding throughput compared with the state-of-the-art design in [92].

Chapter 5

Design of Simplified SC Polar Decoders

This chapter presents the latency analysis and the architecture design of *simplified* SC (SSC) polar decoders [61]. The remainder of this chapter is organized as follows. A brief introduction is given in Section 5.1. A review of the SSC decoding algorithm is provided in Section 5.2. In Section 5.3, a systematic way to calculate the latency for the SSC decoder is proposed. An SSC polar decoder architecture is presented in Section 5.4. Using the pre-computation technique, the latency-reduced SSC decoder architecture is presented in Section 5.5. Comparison results with state-of-the-art works are presented in Section 5.6. Section 5.7 concludes this chapter.

5.1 Introduction

According to Chapter 3 and Chapter 4, we can see that the decoder architecture design is an active area of research. In real-time applications, to guarantee better performance over turbo codes or LDPC codes, polar codes' code length N needs to be greater 2^{10}. For the SC decoding algorithm [28], the long code length leads to a significant increase in decoding latency. To widely employ polar codes in real-time applications, the design of

low-latency decoders is greatly needed. In previous chapters, we have provided several design methodologies to address this problem. However, the few existing SC decoders fail to completely satisfy this requirement. According previous analysis, the decoding latency for conventional SC decoders in [28, 92, 94] is 2(N-1). In Chapter 3 and Chapter 4, though the pre-computation technique is used, their decoding latency remains (N). Significant research has been directed towards code constructions and decoding algorithms for polar codes. However, how to construct efficient SC polar decoders with even shorter latency is still a problem.

To this end, [60] introduced a modified decoding scheme called the *simplified* SC (SSC) decoding algorithm is introduced. First, the SSC algorithm removes all the redundant computations required by the *rate-zero* constituent codes. Second, by simplifying local decoders for *rate-one* constituent codes, the SSC algorithm can further reduce the decoding latency. However, how to efficiently determine the decoding latency for one specific polar code has not been addressed yet. Hardware implementation for the SSC decoding algorithm has not been proposed either. Another important issue, which we should be aware of is, in some cases the decoding latencies for the SSC decoder are still longer than those in [36, 58, 59].

In this paper, first we propose a systematic way to calculate the decoding latency for the SSC algorithm. With this method, we can calculate the decoding latency for any given polar code easily. Then a formal design flow for the SSC decoder architecture is developed. Finally, in order to always achieve a lower decoding latency than those in [36, 58, 59], a novel pre-computation SSC decoder architecture is also proposed.

5.2 Review of the SSC Algorithm

5.2.1 Tree Representation of the SC Decoding Algorithm

Previously, the FFT-like butterfly trellis shown in Figure 3.3 is employed to denote the SC decoding process. [60] has shown that the binary tree can also be employed to provide another representation of the SC decoding algorithm. Again, the 8-bit polar decoder is used as the running example to give a clear explanation.

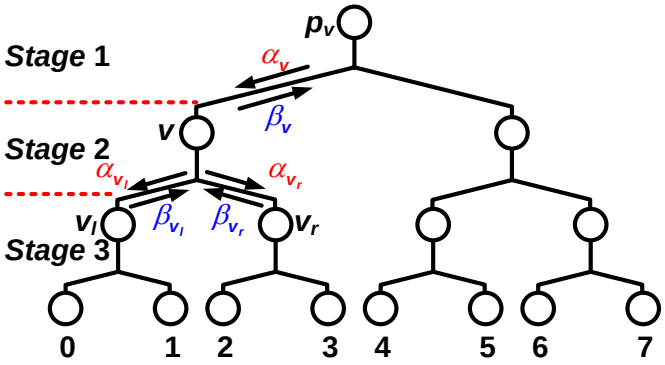
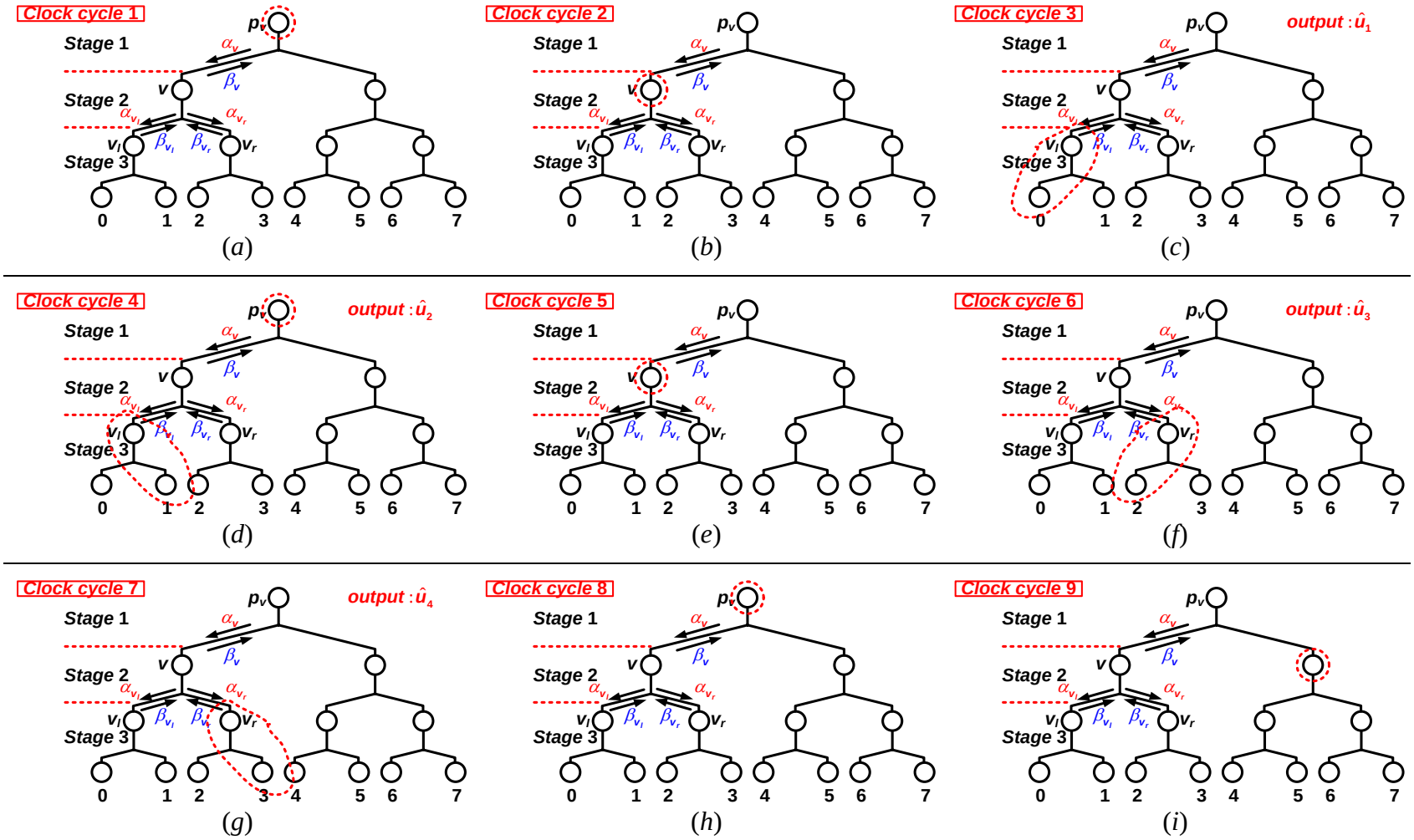


Figure 5.1 Tree representation of the 8-bit SC polar decoder.

As illustrated in Figure 5.1, like the previous trellis representation, this binary tree can be partitioned into three stages. In general, except of the last stage, each stage is composed of one level of nodes of the binary tree. The last stage is composed of the two bottom levels of the binary tree.

Every node within the binary tree representation is associated with some message vectors. For any *Node v*, $\alpha_v(\beta_v)$, $\alpha_{v_l}(\beta_{v_l})$, and $\alpha_{v_r}(\beta_{v_r})$ are the message vectors passing between node v and its parent p_v, left child v_l, and right child v_r, respectively. If, *Node v* is located in the last stage, we have $\beta_v = \text{sgn}(\alpha_v)$, and this β_v is the decoding result of the corresponding code bit. With those new notations, the SC decoding algorithm can be re-descripted by the following equations:


Clock cycle 1
Clock cycle 2
Clock cycle 3
Clock cycle 4
Clock cycle 5
Clock cycle 6
Clock cycle 7
Clock cycle 8
Clock cycle 9
output : $\hat{u}_1$
output : $\hat{u}_2$
output : $\hat{u}_3$
output : $\hat{u}_4$
Stage 1
Stage 2
Stage 3
p_v
α_v
β_v
v
v_l
v_r
α_{v_l}
α_{v_r}
β_{v_l}
β_{v_r}
0 1 2 3 4 5 6 7
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)

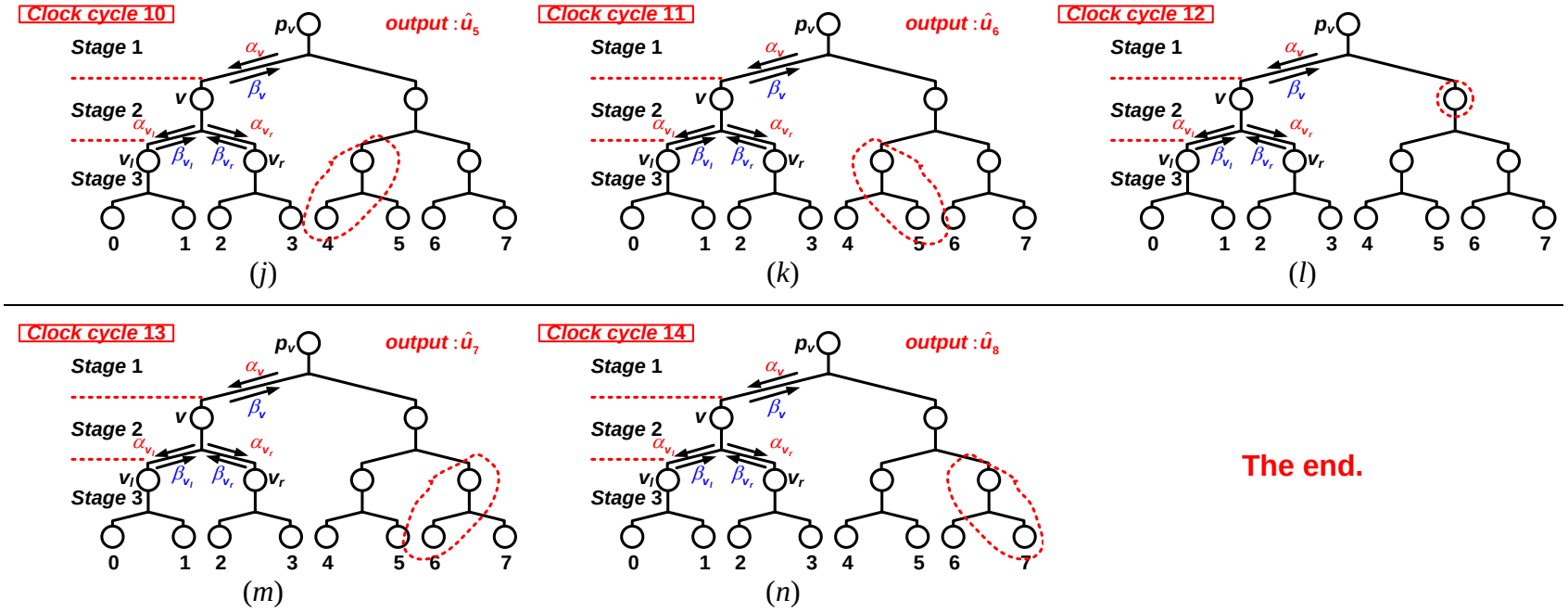


Figure 5.2 Decoding process of the 8-bit SC decoder with tree representation.

The Decoding Scheme for the SC Algorithm

1: begin *from the root node*

2: $\begin{cases} \alpha_{v_l}[i] = \alpha_v[2i] \boxplus \alpha_v[2i+1], \\ \alpha_{v_r}[i] = \alpha_v[2i](1-2\beta_{v_l}[i]) + \alpha_v[2i+1]; \end{cases}$

3: $\begin{cases} \beta_v[2i+1] = \beta_{v_r}[i], \\ \beta_v[2i] = \beta_{v_l}[i] \oplus \beta_{v_r}[i]; \end{cases}$

4: **if** *v in the last row*

5: $\beta_v = \mathrm{sgn}(\alpha_v)$;

6: **output** β_v;

7: **endif**

8: endbegin

For easy understanding of the tree representation, a detailed example is illustrated in Figure 5.2. The nodes within the red dotted circles stay active during the corresponding clock cycles.

5.2.2 The SSC Decoding Algorithm

According to the definition of polar codes, the input vector as u_1^N consists of a *random part* $u_{\mathcal{A}}$ and a *frozen part* $u_{\mathcal{A}^c}$. In order to reduce the error rate, when decoding the *frozen part*, we simply assign $\hat{u}_{\mathcal{A}^c} = u_{\mathcal{A}^c}$. Here the values of $u_{\mathcal{A}^c}$ are known before the decoding process. Therefore, the tedious computations associated with the *frozen part* can be removed. The real decoding task is to generate an estimate $\hat{u}_{\mathcal{A}}$ of $u_{\mathcal{A}}$ [28].

This fact means the node with in the tree representation can be categorized into three parts: the *one nodes* which are associated with the *random part*, the *zero nodes* which are associated with the *frozen part*, and the rest as the *mixed nodes*. It is assumed that the polar code illustrated in Figure 5.1 is actually an $(8,3,\{5,6,7\},(0,0,0,0,0))$ code, whose *random par*t consists of the 5th, 6th, and 7th bits. And all its *frozen bit*s are assigned as "0". Its tree representation is given in Figure 5.3 as follows:

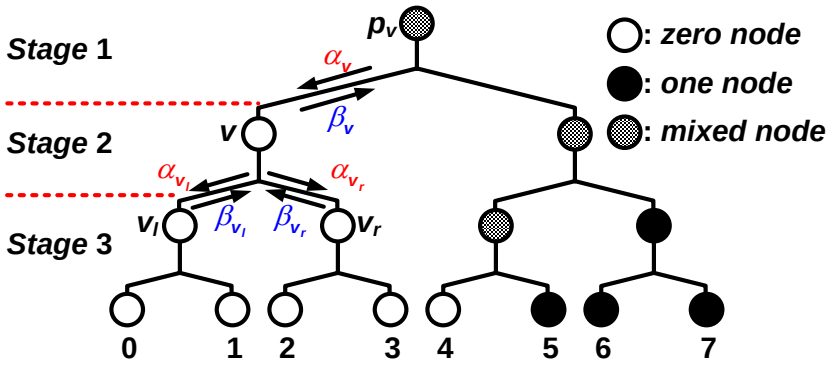


Figure 5.3 The revised tree representation of the 8-bit SC polar decoder.

Since the values of the *frozen part* are pre-determined, we do not need actually go through the data flow of the *zero nodes*. Similarly, the calculations corresponding to the *one nodes* can be replaced by the matrix multiplication operations as well. For more details and the proof, please refer to [60]. In this chapter, we only give the SSC decoding algorithm as follows:

The Decoding Scheme for the SSC Algorithm

1: if $v \in \{mixed\ \ node\}$ **then**

2: $\begin{cases} \alpha_{v_l}[i] = \alpha_v[2i] \boxplus \alpha_v[2i+1], \\ \alpha_{v_r}[i] = \alpha_v[2i](1-2\beta_{v_l}[i]) + \alpha_v[2i+1]; \end{cases}$

3: $\begin{cases} \beta_v[2i+1] = \beta_{v_r}[i], \\ \beta_v[2i] = \beta_{v_l}[i] \oplus \beta_{v_r}[i]; \end{cases}$

4: elseif $v \in \{zero\ node\}$ **then**

5: $(\hat{u}[\min \mathcal{I}_v], \cdots, \hat{u}[\max \mathcal{I}_v]) = (u[\min \mathcal{I}_v], \cdots, u[\max \mathcal{I}_v]);$

6: else $(\hat{u}[\min \mathcal{I}_v], \cdots, \hat{u}[\max \mathcal{I}_v]) = \beta_v G_{2^{n-d_v}}$;

8: endif

Here $\mathcal{I}_v$ is the set which contains indices of all leaf nodes that are descendants of v. d_v is the depth of v. $G_{2^{n-d_v}}$ denotes the generation matrix of dimension 2^{n-d_v} [60]. Index i can vary from 0 to $(2^{n-d_v-1}-1)$. We can see, the computations of both the *zero nodes* and the

one nodes can be saved as a result. Therefore, the decoding latency and hardware complexity are reduced.

5.3 The Latency Analysis of the SSC Decoder

Since decoding latency is a key criterion to weigh the polar decoder design, it is important to be able to estimate the latency accurately for any polar code. However, in most applications the code length is large and the locations of *frozen bits* are random. The calculation for the SSC decoding latency is not trivial. To this end, an efficient approach to calculate the SSC decoding latency for any polar code is proposed in this section.

5.3.1 A Simple Example

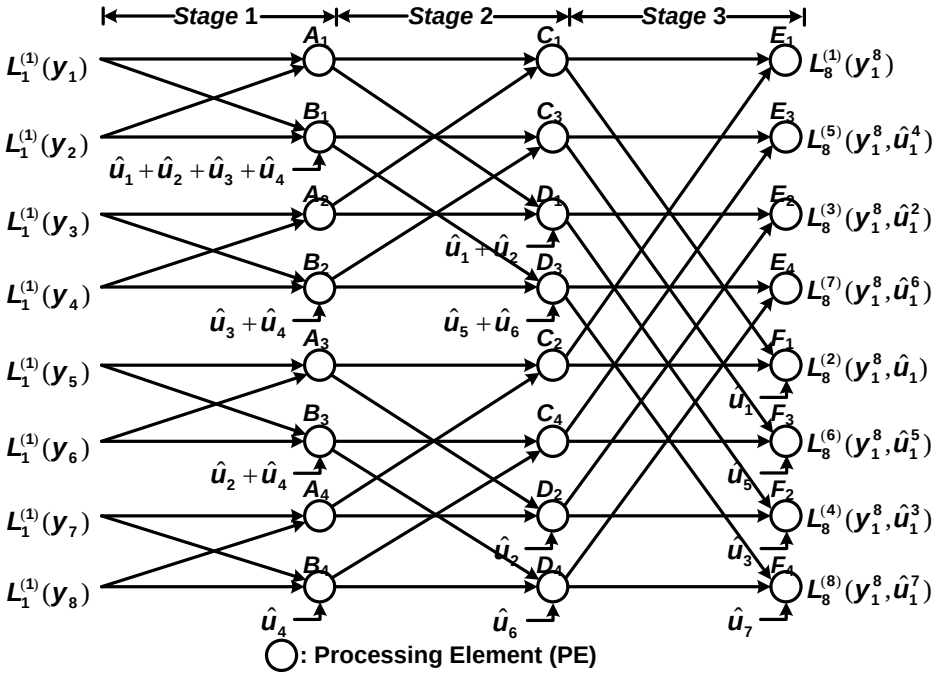


Figure 5.4 Conventional SC decoding process.

Again, consider the $(8,3,\{5,6,7\},(0,0,0,0,0))$ code in Figure 5.3. If we do not try to distinguish *frozen bits* from *information bits*, the dependence relationship of each PE can always be derived as an FFT-like structure shown in Figure 5.4. Here, we do not try to distinguish the Type I PE and Type II PE any more.

Now, with the SSC decoding algorithm, the PEs in Figure 5.4 are not homogeneous any more. As shown in Figure 5.5, all PEs have been categorized into three families. Since the values of *frozen bits* can be determined as zero immediately, computations of all *zero nodes* are eliminated. Also the tedious calculations required by *one nodes* in previous approaches are replaced with simple matrix multiplications ($\times G_{2^{n-d_v}}$), which can be easily implemented with XOR operations. Therefore, only the nodes along the red arrow lines need to be activated and the others can be ignored. It can be observed that the decoding procedure has been simplified.

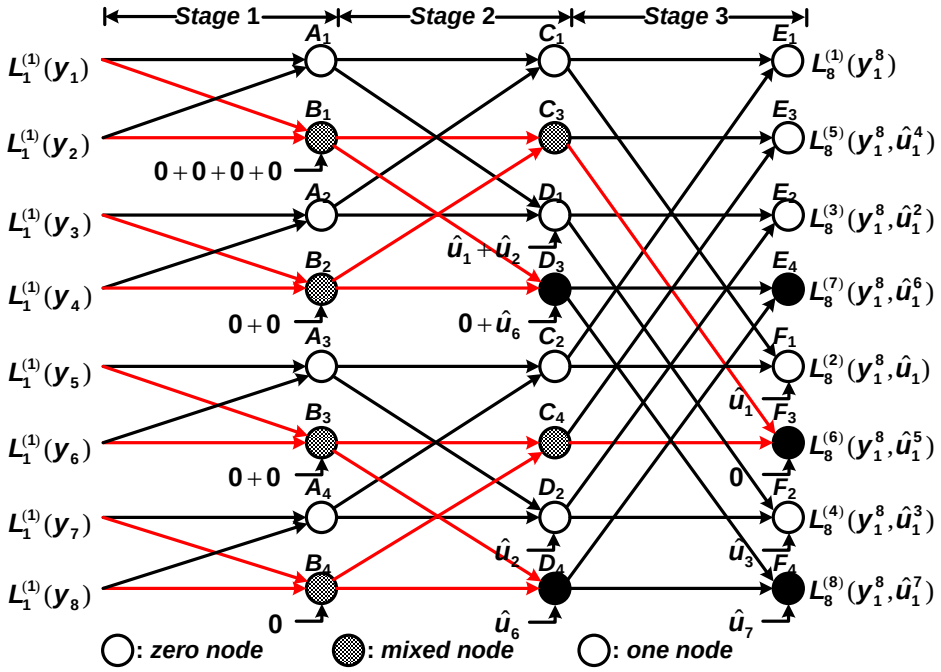


Figure 5.5 SSC decoding process of the (8, 3) polar code.

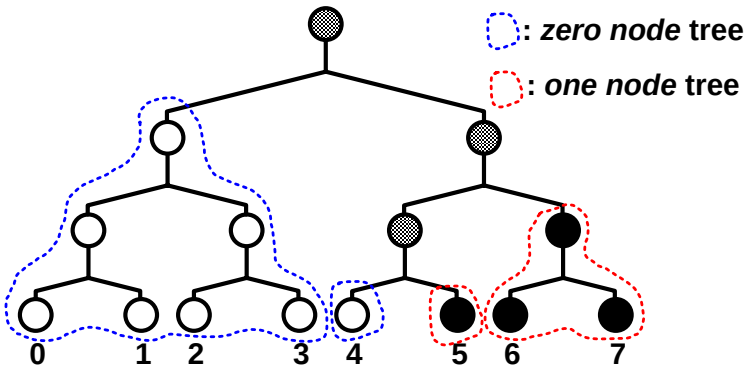


Figure 5.6 Nodes categorization with circled binary trees.

Figure 5.6 illustrates the corresponding tree representation. The calculations of those circled binary trees are removed according to the SSC decoding algorithm. The decoding latency now can be calculated by subtracting the computation latency savings from the previous SC decoding latency.

In this example, the latency for the 8-bit conventional SC decoder is 14 clock cycles. The latency savings for each constituent sub-trees (from left to right) are 7, 1, 1, and 3 clock cycle(s), respectively. One must note that in the SSC decoding process, each *one node* tree will be replaced by a simple matrix multiplication ($\times G_{2^{n-d_v}}$). Hence, we need to take the processing time for the matrix multiplications into account. Here, the two *one node* trees (from left to right) will be replaced by $\times G_1$ and $\times G_2$, respectively. Their latency costs are 1 and 2 clock cycle(s) (see more details in Section 5.4). Therefore, the decoding latency for this (8, 3) decoder can be calculated below:

$$T_{(8,3)} = 14 - (7+1+1+3) + (1+2) = 5. \tag{5-1}$$

Alternatively, the decoding latency of the SSC algorithm can be obtained in the straightforward way. The schedule of events for this 8-bit SSC decoding process is shown

in Table 5.1. The SSC decoding latency of five clock cycles is also verified with this method.

Table 5.1 The SSC decoding schedule for the (8, 3) polar decoder.

Clock cycle	**1**	**2**	**3**	**4**	**5**
Active nodes	$B_{1,\ldots,4}$	$C_{3,4}$	F_3	$D_{3,4}$	$\times G_2$
Output bits	—	—	$\hat{u}_6$	—	$\hat{u}_7, \hat{u}_8$

5.3.2 Latency Analysis for the General Code Length

In this sub-section, we would like to find a method to derive the SSC decoding latency for the general code length. From the remainder of this sub-section, we will see similar calculation approach employed in Figure 5.6 could be used for the general case. However, before deriving this method, two lemmas are presented first.

The first lemma is dealing with the decoding latency of a depth-given a constituent *zero (one) node* tree. As the name implies, a constituent tree is defined as *zero (one) node* tree if and only all its nodes are *zero (one) nodes* (see Figure 5.6). This lemma is stated as follows:

Lemma 1 *The decoding latency for a constituent zero (one) node tree of depth d is* 2^{d+1}*-1.*

Proof Each node of a constituent *zero* (*one*) *node* tree will be activated once. Therefore, the latency equals the number of nodes in a *perfect binary tree* of depth *d*, i.e., (2^{d+1}-1).■

With **Lemma 1**, we can also derive the decoding latency for the SC polar decoder:

Lemma 2 *The decoding latency for a conventional N-bit SC decoder is 2(N-1).*

Proof A conventional *N*-bit SC decoder can be considered as two constituent trees of depth $\log_2 N$-1. The lemma follows immediately.■

Now, we have both **Lemma 1** and **Lemma 2** available. By subtracting the clock savings from the total latency required by the conventional *N*-bit SC decoders, the SSC decoding latency for the general code length is obtained as follows.

Theorem 1 *The decoding latency T for an N-bit SSC polar decoder can be calculated by*:

$$T = 2(N-1) - \sum_{\{i\}} (2^{d_i+1} - 1) - \sum_{\{j\}} [(2^{d_j+1} - 1) - (d_j + 1)], \tag{5-2}$$

where $\{i\}$ *and* $\{j\}$ *are the sets of zero node trees and one node trees, respectively.*

Proof According to Lemma 2, the decoding latency for a conventional *N*-bit SC decoder is 2(*N*-1). Removing a *zero node* tree of depth *d* will reduce latency by $2^{d+1}-1$ clock cycles (Lemma 1). Similarly, removing a *one node* tree will result in the same reduction. But we need to spend additional clock cycles for the matrix multiplication operation. With the fully pipelined matrix multiplication block, which is discussed in next section, *d*+1 additional clock cycles are required by the "$\times G_{2^d}$" operation. This completes the proof.■

Now, we would like to use **Theorem 1** for the previous code example. In Figure 5.6, there are two *zero node* trees in the binary tree plot, whose depths are 2 and 0, respectively. There are also two *one node* trees with depths 1 and 0. According to Eq. (5-2), the latency for the (8, 3) SSC decoder can be calculated as follows:

$$\begin{aligned} T_{(8,3)} &= 2\times(8-1) - [(2^3-1) + (2^1-1)] - \{[(2^1-1) - (0+1)] + [(2^2-1) - (1+1)]\} \\ &= 5, \end{aligned} \tag{5-3}$$

which matches our previous analysis.

Next we use this method for another code, which is more complicated than the first code. Consider the code $(8,5,\{2,3,5,6,7\},(0,0,0))$, its tree representation is illustrated in Figure 5.7 as follows:

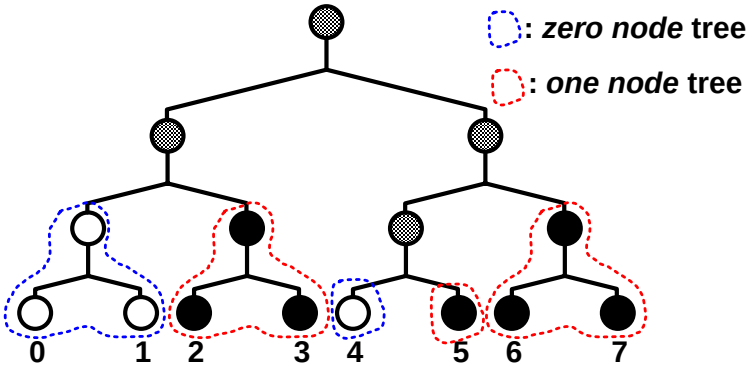


Figure 5.7 Sub-trees identification for an (8, 5) code example.

With the conventional approach, we go through the entire decoding process. Its decoding schedule is derived as shown in Table 5.2. It can be observed that its decoding latency is 8 clock cycles.

Table 5.2 The SSC decoding schedule for the (8, 5) polar decoder.

Clock cycle	**1**	**2**	**3**	**4**
Active nodes	$A_{1,\ldots,4}$	$C_{1,2}$	$\times G_2$	$B_{1,\ldots,4}$
Output bits	—	—	$\hat{u}_2, \hat{u}_3$	—
Clock cycle	**5**	**6**	**7**	**8**
Active nodes	$C_{3,4}$	F_3	$D_{3,4}$	$\times G_2$
Output bits	—	$\hat{u}_6$	—	$\hat{u}_7, \hat{u}_8$

Meanwhile, the latency can also be calculated with the proposed approach. According to Eq. (5-2), the SSC decoding latency is:

$$\begin{aligned} T_{(8,5)} &= 2\times(8-1)-[(2^2-1)+(2^1-1)]-\{[(2^1-1)-(0+1)]+2\times[(2^2-1)-(1+1)]\} \\ &= 8. \end{aligned} \tag{5-4}$$

Two results match again. This approach is *universal* that can be applied to the SSC polar decoder with any code length, code rate, or *frozen bit* location. Without laboriously following the entire decoding process like before, the decoding latency can be derived easily in a systematic manner.

5.4 Proposed SSC Decoder Architecture

Now we present the design steps for the SSC decoder. The $(8,3,\{5,6,7\},(0,0,0,0,0))$ polar code example used in [60] is employed here to illustrate the procedure.

5.4.1 DFG Analysis of the SSC Decoding Process

For the conventional SC algorithm, its DFG analysis [58] has been carried out in Chapter 3. Now we show that several modifications are needed to obtain the revised DFG for the SSC decoding process. According to the latency analysis, all PEs corresponding to *zero node* trees can be safely removed. All PEs corresponding to *one node* trees can be replaced by matrix multiplications ($\times G_{2^{n-d_v}}$). In Figure 5.8, by removing all the shaded nodes, a total of 5 clock cycles are required to output all code bits, compared 14 clock cycles for the conventional ones.

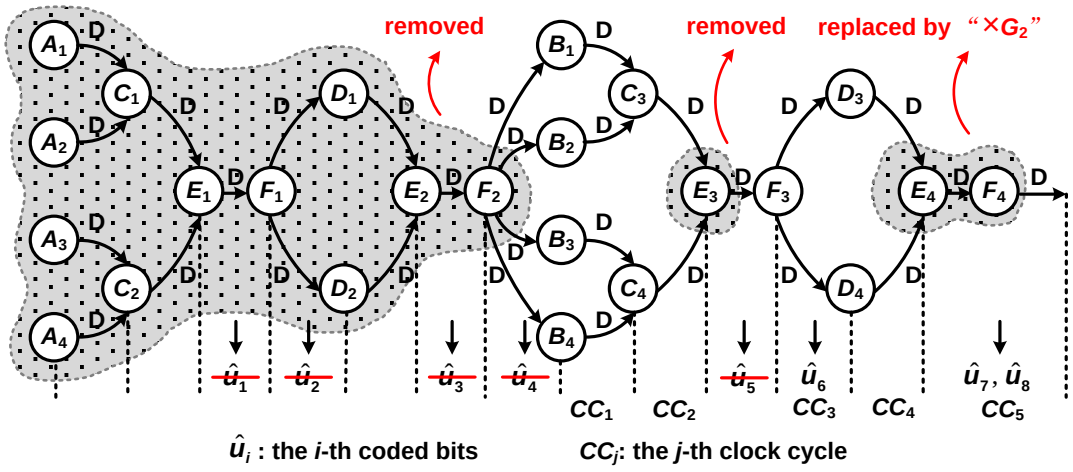


Figure 5.8 The DFG for the SSC decoding approach.

5.4.2 The SSC Decoder Architecture Design

With the revised DFG available, we can derive the corresponding decoder architecture now. Interpreting the DFG in Figure 5.8 with hardware implementations, we can derive

the SSC polar decoder architecture illustrated in Figure 5.9. Since all *frozen bits* are set to 0, PEs $B_{1,\ldots,4}$ and F_1 are nothing but W-bit adders, where W is the quantization length. $C_{1,2}$ and $D_{1,2}$ can be implemented with Type I and Type II PE proposed in Chapter 3, respectively.

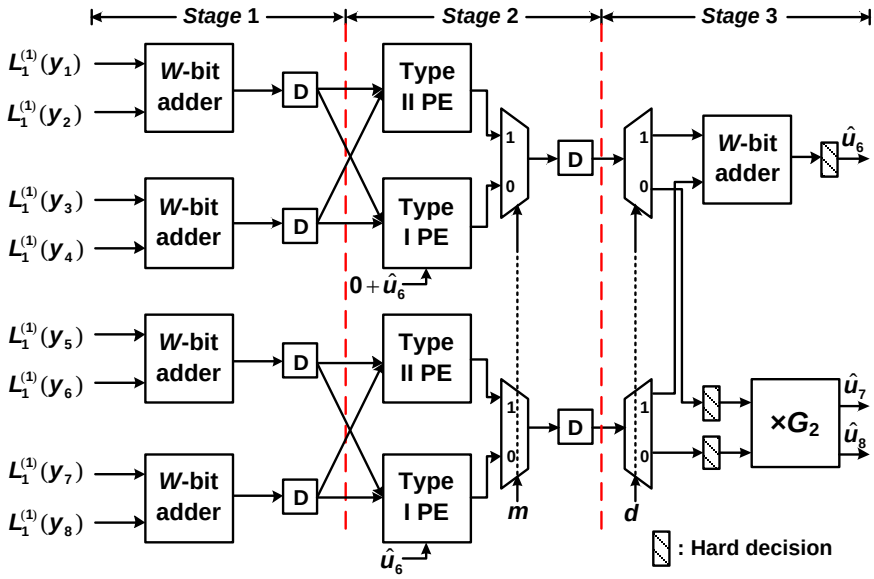


Figure 5.9 The SSC decoder architecture for the (8, 3) codes.

For the matrix multiplication block, it is known that the generation matrix can always be written in the form:

$$G_N = B_N F^{\otimes n} \text{ with } F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \tag{5-5}$$

where N is the code length, n equals $\log_2 N$, $\otimes$ is the *Kronecker* power operation, and B_N is the *bit-reversal* matrix.

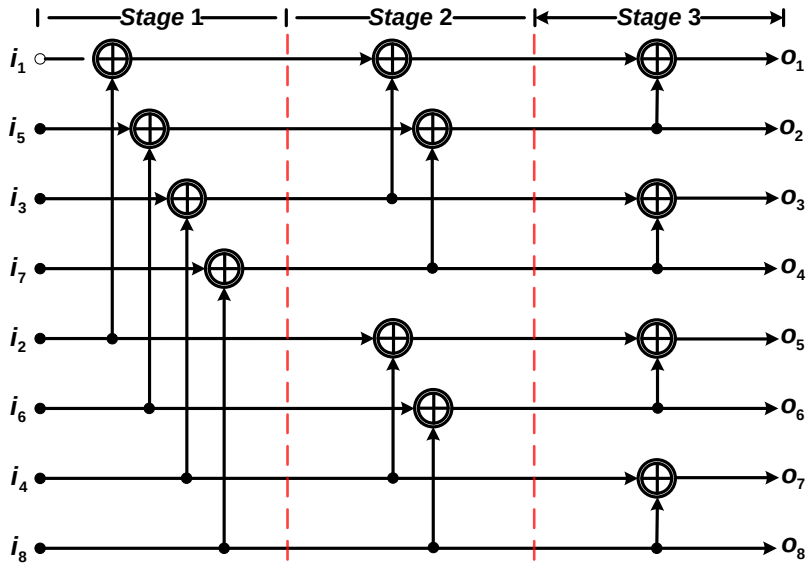


Figure 5.10 The fully-pipelined implementation for $F^{\otimes 3}$.

The matrix multiplication block is composed of two parts: the B_N part and the $F^{\otimes n}$ part. Because all inputs to the matrix multiplication block are available at the same time, the B_N part can always be implemented with simple routing. The fully pipelined design example for $F^{\otimes 3}$ is illustrated in Figure 5.10.

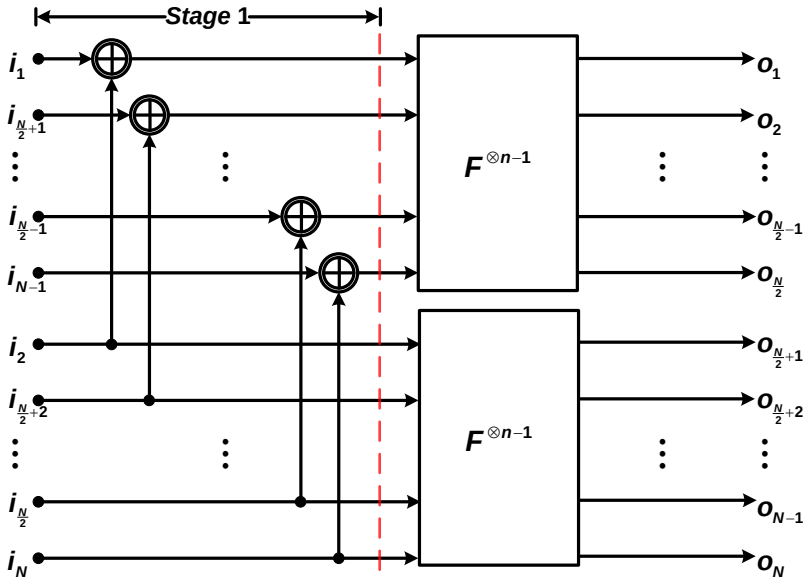


Figure 5.11 Recursive construction of the fully-pipelined implementation.

The general form of the $F^{\otimes n}$ part can be implemented in a recursion fashion, which is shown in Figure 5.11. With this approach, two copies of $F^{\otimes n-1}$ implementation and $N/2$ XOR gates are combined together to produce the fully pipelined implementation of the $F^{\otimes n}$ part.

5.5 Pre-Computation Look-Ahead SSC Polar Decoder

5.5.1 Look-Ahead SSC Polar Decoder Architecture

Though the SSC algorithm can effectively reduce the latency, in some occasions its latency can still be greater than that of the pre-computation look-ahead SC approach [36, 58, 59]. This is because the SSC decoding latency highly depends on the specific code which is being processed. Following our prior approach in [36, 58, 59], we can further reduce the decoding latency for SSC decoder by using pre-computation look-ahead technique [99]. Back to our (8, 3) decoder example, it can be noted that $C_{3,4}$ and $D_{3,4}$ are actually dual PEs which can be merged together. The corresponding decoding schedule is shown in Table 5.3:

Table 5.3 Pre-computation schedule for the (8, 3) decoder.

Clock cycle	**1**	**2**	**3**	**4**
Active nodes	$B_{1,\ldots,4}$	$C_{3,4}$, $D_{3,4}$	F_3	$\times G_2$
Output bits	—	—	$\hat{u}_6$	$\hat{u}_7, \hat{u}_8$

Since the pre-computation look-ahead technique is used, the same *merged* PE proposed in Chapter 3 is employed here to construct the new architecture shown in Figure 5.12. This *merged* PE is used instead of Type I and Type II PEs to carry out both operations efficiently in the same clock cycle.

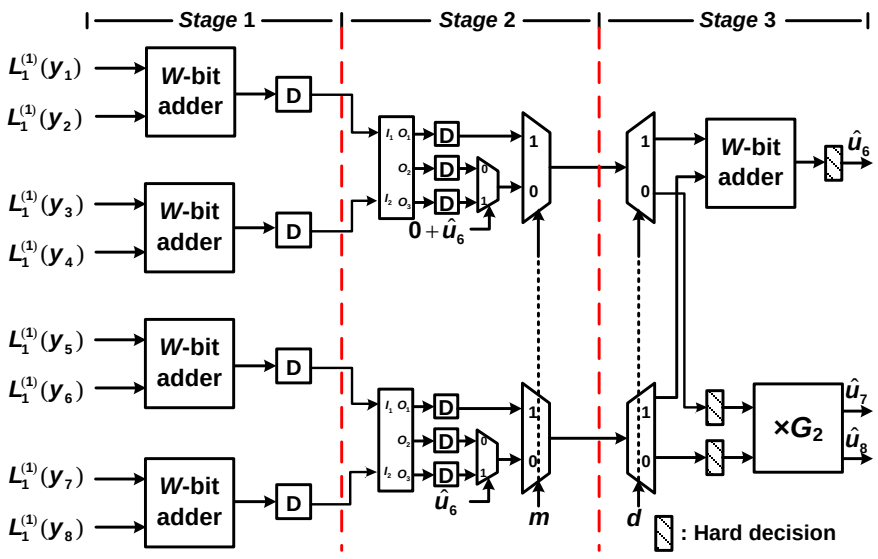


Figure 5.12 Pre-computation SSC decoder architecture for the (8, 3) polar codes.

5.5.2 Latency of the Look-Ahead SSC Polar Decoder

However, for the general case, how to calculate the reduced latency remains undetermined. Again, we will exploit the nodes categorization tree to extend pre-computation scheme to the realm of *N*-bit decoder condition.

The method to compute the decoding latency of the pre-computation look-ahead SSC decoders is given in the following theorem.

Theorem 2 *The latency T' for an N-bit pre-computation SSC polar decoder can be calculated by:*

$$T' = 2(N-1) - \sum_{\{i\}} (2^{d_i+1} - 1) - \sum_{\{j\}} [(2^{d_j+1} - 1) - (d_j + 1)] - k, \tag{5-6}$$

where k is the number of mixed nodes which have one node descendants in both left and right sub-trees.

Proof According to Figure 5.8, a *mixed node* will be activated twice during one decoding procedure if and only if it has *one node* descendants in both its left and right sub-trees.

However, with pre-computation technique this *mixed node* can be activated only once. Hence, one clock cycle is saved for each qualified *mixed node*. This completes the proof: since the above argument applies to any pre-computation SSC decoder, its decoding latency T' is k clock cycles less than T (see Eq. (5-2)), where k is the number of the qualified *mixed nodes*.■

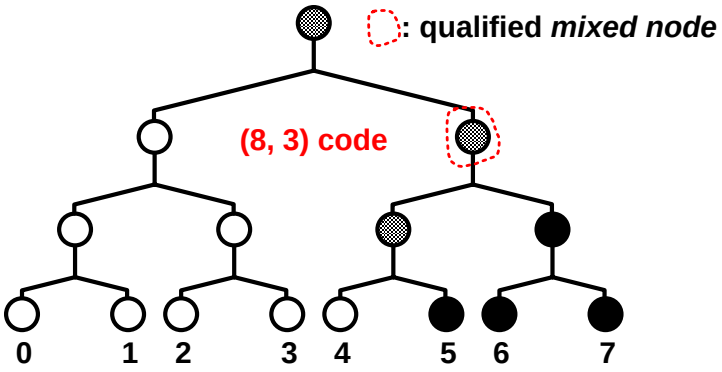


Figure 5.13 The qualified *mixed node* for the (8, 3) code.

We can apply this method back to our previous (8, 3) polar code example. As illustrated in Figure 5.13, the number of qualified *mixed node(s)* is one. We apply **Theorem 2** to calculate its decoding latency. We get the result which matches Table 5.3:

$$T'_{(8,3)} = 5 - 1 = 4. \tag{5-7}$$

For the previously mentioned (8, 5) polar code, we have its pre-computation look-ahead decoding schedule listed in Figure 5.4 as follows:

Table 5.4 Pre-computation schedule for the (8, 5) decoder.

Clock cycle	**1**	**2**	**3**
Active nodes	$A_{1,\ldots,4}$, $B_{1,\ldots,4}$	$C_{1,2}$	$\times G_2$
Output bits	—	—	$\hat{u}_2, \hat{u}_3$
Clock cycle	**4**	**5**	**6**
Active nodes	$C_{3,4}$, $D_{3,4}$	F_3	$\times G_2$
Output bits	—	$\hat{u}_6$	$\hat{u}_7, \hat{u}_8$

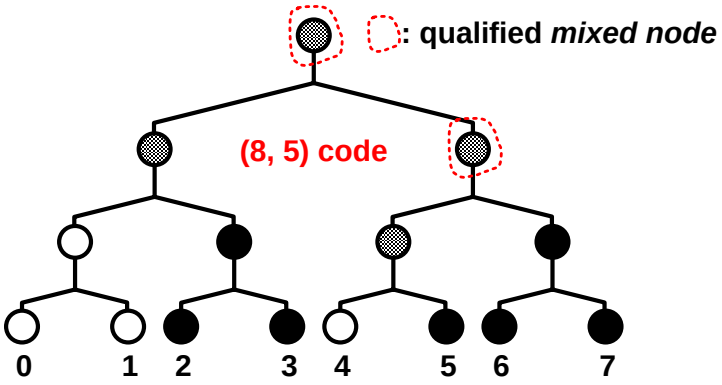


Figure 5.14 The qualified *mixed node* for the (8, 5) code.

According to Figure 5.14, the number of qualified *mixed node(s)* within the binary tree is two. By applying **Theorem 2** again, we get the result which matches Table 5.4:

$$T'_{(8,5)} = 8 - 2 = 6. \tag{5-8}$$

Although the saving here is only one or two clock cycle(s), when N becomes larger, the positions of *one nodes* become more scattered, the advantage brought by the pre-computation method will be more prominent. Please see the 1024-bit polar code in Section 5.6 as an example.

5.6 Latency and Complexity Comparison

In this section, we compare the proposed SSC polar decoder architectures with state-of-the-art references. Comparison results of the decoding latency and the hardware efficiency are listed in Table 5.5 in detail. Since the SSC algorithm is code-aware, here, a common used (1024, 512) polar code is employed as an example to conduction the comparison in Table 5.5.

Table 5.5 Comparison between the proposed SSC decoders and others.

Different designs		Conventional SC decoder†			Pre-computation SC decoder‡			Proposed SSC decoder			Proposed pre-computation SSC decoder		
Hardware complexity													
# of PEs		Type I	Type II	Merged	Type I	Type II	Merged	Type I	Type II	Merged	Type I	Type II	Merged
		1023	1023	—	—	—	1023	1023	1022	—	7	6	1016
# of matrix mult. block		—			—			$\times G_2$	$\times G_4$		$\times G_2$	$\times G_4$	
								1	1		1	1	
# of other REGs		1023W			3068W			1023W			3055W		
# of other MUXs		0			2045W			12W			2032W		
Total	**XOR**	**16368W-3069**			**17390W**			**16371W-3067**			**17375W-19**		
	REG	**1023W**			**3068W**			**1023W+4**			**3055W+4**		
Decoding latency													
Latency		**2046**			**1023**			**1235**			**866**		

†Tree SC decoder proposed in [92].
‡Pre-computation SC decoder proposed in [58].

For the proposed (1024, 512) polar code, there are 258 *one node* trees with depth 0, 97 with depth 1, and 15 with depth 2, respectively. Meanwhile, there are 258 *zero node* trees with depth 0, 97 with depth 1, and 15 with depth 2, respectively. According to **Theorem 1**, the latency for the corresponding 1024-bit SSC polar decoder is calculated as 1235 clock cycles. The number of qualified *mixed nodes* defined in **Theorem 2** is 369. Hence, the decoding latency for 1024-bit pre-computation SSC polar decoder is 866 clock cycles.

Before the comparison, it is mentioned that the counterpart of the proposed SSC decoder is the conventional SC tree decoder [92], and the counterpart of the proposed pre-computation SSC decoder is the pre-computation SC decoder [58]. For hardware complexity, the word-length of the soft message is W. It is assumed that each 1-bit 2-to-1 multiplexer requires the same silicon area as an XOR gate.

Compared with the conventional SC polar decoder, the proposed SSC one can achieve 39.6% decoding latency reduction with the same amount of hardware. The proposed pre-computation SSC decoder requires only 84.6% decoding latency compared with its SC counterpart, with even lower hardware consumption. Therefore, proposed SSC polar decoder designs are shown to have advantages in decoding latency compared with the state-of-the-art designs [36, 58, 59, 92, 94]. For polar codes with long code lengths and deep *one* (*zero*) *node* trees, this advantage can become more apparent.

5.7 Conclusion

In this chapter, a newly proposed decoding algorithm named SSC algorithm is investigated. A method to calculate the SSC decoding latency is proposed. Based on the revised DFG, a design methodology for the SSC polar decoder is given. In order to attain further reduced decoding latency, the pre-computation look-ahead SSC decoder

architecture is also proposed. Comparison results have shown that the proposed architectures turn out to be very attractive for real-time applications.

Chapter 6

Conclusions and Future Research Directions

6.1 Conclusions

This thesis focuses on developing approaches for designing low-latency low-complexity channel decoder architectures for modern communication systems. Decoding latency (or throughput) and hardware complexity are the two main focuses of channel decoder design. How to design a decoder architecture which can well balance the latency and the hardware is still a challenging problem. In this thesis, we are not only satisfied with specific decoder designs. Our main goal is to propose a design methodology which is applicable for general case. With the help of this methodology, we are able to construct the decoder architecture to meet various design requirements formally.

With the help of VLSI DSP techniques [37], two kinds of channel decoders are investigated. First, this thesis studies the non-binary QC-LDPC codes. With the given geometry properties, the low-complexity non-binary QC-LDPC decoder architectures are proposed. We are able to construct the decoder architecture to meet various design requirements formally. As a newly proposed error correction code, polar code has shown

advantages in several aspects. The rest chapters of this thesis derive several polar decoder design methodologies for different real-time communication applications.

6.1.1 Non-Binary LDPC Decoders with Efficient Networks

In Chapter 2, the intrinsic shifting and symmetry properties of non-binary QC-LDPC codes have been exploited. These unique features lead to significant hardware reduction of the switch networks. With those networks, two low-complexity decoder architectures are proposed for Class-I and Class-II non-binary QC-LDPC codes, respectively. This design approach is general and can be applied to other non-binary QC-LDPC decoders.

In order to demonstrate the merits of the proposed method, two design examples are given. Comparison results with the state-of-the-art designs show that for the code example of the 64-ary (1260, 630) rate-0.5 Class-I code, the proposed scheme can save up to 70.6% hardware required by switch network, which demonstrates the efficiency of the proposed technique. The proposed design for the 32-ary (992, 496) rate-0.5 Class-II code can achieve a 93.8% switch network complexity reduction compared with conventional approaches. Furthermore, with the help of a generator for possible solution sequences, both forward and backward steps can be eliminated to offer processing convenience of check node unit (CNU) blocks. Results show that the proposed 32-ary (992, 496) rate-0.5 Class-II decoder can achieve 4.47 Mb/s decoding throughput.

6.1.2 Low-Latency Sequential SC Polar Decoder

Due to their low encoding and decoding complexity, polar codes have recently emerged as one of the most favorable capacity-achieving error correction codes. However, because of the large code length required by practical applications, the few existing SC decoder implementations still suffer from not only high hardware cost but also long decoding latency.

In Chapter 3, a data-flow graph (DFG) for the SC decoder is derived for the first time. A complete hardware architecture is first derived for the conventional tree SC decoder and the *feedback part* is presented next. It is worth noting that the *feedback part* can be generated in a systematic way. In order to reduce the decoding latency, pre-computation look-ahead techniques are proposed based on the DFG analysis. Sub-structure sharing is used to design a *merged* PE for higher hardware utilization. Gate-level design details of the *merged* PE are provided also. Compared with the conventional *N*-bit tree SC decoder, the proposed pre-computation look-ahead decoder architecture can achieve twice speedup with similar hardware consumption.

6.1.3 High-Throughput SC Polar Decoder Architectures

Two meet the requirements of the modern communication systems, two new SC polar decoder architectures are proposed in Chapter 4 to achieve higher throughput.

First, a family of TC interleaved pre-computation look-ahead SC polar decoders is proposed. Several properties of the decoder architectures have been proposed. In order to meet different throughput requirements for a diverse set of application scenarios, different versions of the proposed TC interleaved decoders are required. With the help of those properties, we are able to construct the desired decoder architecture in a systematic way. Furthermore, when the concurrent parameter $M = N$-1, the 100% hardware utilization is achieved. Compared with the conventional *N*-bit tree SC decoder [28, 92], the proposed TC interleaved architectures can achieve as high as 2(*N*-1) times speedup with only 50% decoding latency and $(N \cdot \log_2 N)/2$ *merged* PEs.

Secondly, a RC interleaved processing approach is also introduced to design polar decoders. Based on the DFG analysis and the folding technique, the RC *L*-interleaved pre-computation look-ahead SC decoder is proposed. The RC 2-interleaved version is

used to show the advantage of this approach. Compared with the newly proposed line decoder design [94], the RC 2-interleaved design can achieve 4 times decoding throughput with the same hardware cost.

6.1.4 Design of Simplified SC Polar Decoders

The *simplified* successive cancellation (SSC) algorithm is a revised version of the conventional SC decoding algorithm. Though it is mathematically equivalent to the SC decoding algorithm, its decoding latency and hardware cost have been reduced. In Chapter 5, some problems regarding the SSC polar decoder design issues are discussed.

Since the SSC decoding process highly depends on the specific polar code it is dealing with, its decoding latency is not trivial to calculate. In this chapter, we present the first systematic approach to formally derive the SSC decoding latency for any given polar code. By revising the DFG for the conventional SC decoder, we can easily get the DFG for the SSC decoder. With this revised DFG, the method to derive various SSC polar decoder architectures for any specific code is also presented. In order to further reduce the latency, the pre-computation look-ahead technique is also employed. Similarly, the latency-reduced SSC decoder's latency can also be calculated with a simple equation. Compared with the state-of-the-art SC decoder designs, the two SSC polar decoders can save up to 39.6% decoding latency with the same hardware cost.

Recently, a low-latency decoding scheme referred as the *simplified* successive cancellation (SSC) algorithm has been proposed for the decoding of polar codes. In this brief, we present the first systematic approach to formally derive the SSC decoding latency for any given polar code. The method to derive various SSC polar decoder architectures for any specific code is also presented. Moreover, it is shown that with the pre-computation technique, the decoding latency can be further reduced. Similarly, the

latency-reduced SSC decoder's latency can also be calculated with a simple equation. Compared with the state-of-the-art SC decoder designs, the two SSC polar decoders can save up to 39.6% decoding latency with the same hardware cost.

6.2 Future Research Directions

In the future, we would like to extend our research in the following aspects: (1) extending our switch network designing methodology to other non-binary QC-LDPC codes; (2) designing low-latency low-complexity CNU architecture for further non-binary LDPC decoders; (3) implementing the pre-computation look-ahead SC polar decoder with real chip; (4) investigating the hardware architectures for the *list decoding* polar decoders; and (5) designing the polar decoder architectures for the *belief propagation* (BP) algorithm.

6.2.1 Switch Networks for Other Non-Binary QC-LDPC Codes

In Chapter 2, the switch networks construction methodology for both Class-I and Class-II codes. This method leads to significant reduction of memory size and routing complexity. However, now this method is only applicable for Class-I and Class-II codes.

In the next step, we would like to apply it to other non-binary QC-LDPC codes. The non-binary QC-LDPC codes form a big family. The previously stated two classes of non-binary QC-LDPC codes can be categorized into codes constructed in [49]. Usually, the techniques such as *array masking* and *array dispersion* are employed to construct non-binary QC-LDPC codes. Several useful codes have been proposed by Lingqi's [46] and Bo's [54] PhD dissertations as references. Table 6.1 gives a brief overview of those popular non-binary QC-LDPC codes. According to the listed construction approaches, those codes are highly related. Therefore, we can expand our current approach to other kinds of non-binary QC-LDPC to make this approach more general.

Table 6.1 Summary of major non-binary QC-LDPC codes.

Reference	Contribution	Comment
[47]	A general ***matrix dispersion*** construction of non-binary QC-LDPC codes based on ***finite fields*** is given.	Constriction details can be found in ***Chapter 3*** of [46].
[48]	A ***non-binary incidence-vector*** construction of non-binary QC-LDPC codes based on ***finite geometries*** is proposed.	Constriction details can be found in ***Chapter 4*** of [46]. This is a prior work of [52].
[51]	A ***unified approach*** for constructing binary and non-binary QC-LDPC codes is presented.	This is a prior work of [49].
[52]	Construction of non-binary QC-LDPC codes based on flats of finite ***Euclidean geometries*** and ***array masking*** is presented.	Constriction details can be found in ***Chapter 5*** of [54].
[53]	Construction of non-binary QC-LDPC codes based on ***arrays of circulant matrices*** and ***multi-fold array dispersions*** is presented.	Constriction details can be found in ***Chapter 4*** of [54].
[50]	Four specific algebraic constructions of RS-based QC-LDPC codes (***construction based on RS codes***) are presented.	Constriction details can be found in ***Chapter 6*** of [54].
[49]	Construction of non-binary QC-LDPC codes based on ***dispersing*** matrices over ***subgroups*** of non-binary finite fields is proposed.	Constriction details can be found in ***Chapter 8*** of [54].

6.2.2 Low-Latency Low-Complexity CNU Architectures

Similar to their binary counterparts, in the non-binary LDPC decoders, CNU has the highest hardware complexity among all units. Therefore, low-latency low-complexity CNU design is the most favorable topic for non-binary decoders [38, 39, 42-44, 68]. Although in Chapter 2, by introducing the generator for possible solution sequences, the CNU process is simplified, this method is not suitable for 64-ary (or higher order) LDPC codes since the throughput is too slow to be practical. Also, Because of the inherent high decoding complexity, the hardware cost of CNU block will increase drastically with the increase of the finite field order.

Therefore, CNU architectures with low latency and low complexity are still required. In the next design, we are going to refine the previous *path-finding* approach to present an efficient architecture design for the CNU block.

6.2.3 Chip Implementation of the Proposed Polar Decoders

According to Chapter 3 up to Chapter 5, the proposed polar decoder architectures have shown advantages over previous works. However, the best way to evaluate the decoding throughput and hardware consumption of one design is to implement it. Also we know that there are at two published works that provide detailed hardware complexity reports in FPGA or ASIC [93, 94]. Implementing the proposed designs with prototype would give a much more clear view of the potential of those works. This would also alleviate the somehow problematic complexity comparison with the *feedback part* omitted.

6.2.4 List Decoder Architectures for Polar Codes

In this thesis, all the polar decoders are dealing with SC decoding algorithm, or its variants, such as Min-Sum algorithm and SSC decoding algorithm. However, the SC

decoding algorithm family could not achieve the optimal decoding performance [107]. Recently, the list decoding approach is reported to show better performance compared with other codes [57]. Until now, no hardware implementation of the list polar decoder has been reported. Actually, the list decoding algorithm for polar code is nothing but a multi-way version of the conventional SC decoding followed by a selecting step. Therefore, the SC decoding algorithm and the list decoding algorithm are highly related. Our previous design approaches on SC polar decoder architectures can be employed.

6.2.5 Polar Decoders Using Belief Propagation Algorithm

Aside from the SC decoding algorithm, there is another decoding algorithm for polar codes: the *belief propagation* (BP) algorithm. This algorithm was first used to give a performance comparison of polar codes and Reed-Muller codes [34]. In [107], Korada and Urbanke have proved that SC decoding as a particular instance of BP algorithm. The performance of BP decoding algorithm is in general superior to that of the SC algorithm. Also, the corresponding decoding latency is much shorter than that of the SC decoder. Although the computation complexity of BP decoder is higher than the SC decoder, its hardware implementation is very attractive for hardware designers.

Bibliography

[1] Wikipedia contributors. (13 November 2012 20:48 UTC). *Mobile phone*. Available: http://en.wikipedia.org/w/index.php?title=Mobile_phone&oldid=521359663

[2] Wikipedia contributors. (13 November 2012 21:10 UTC). *China Mobile*. Available: http://en.wikipedia.org/w/index.php?title=China_Mobile&oldid=521929749

[3] "IEEE Standard for Air Interface for Broadband Wireless Access Systems," *IEEE Std 802.16-2012 (Revision of IEEE Std 802.16-2009)*, pp. 1-2542, 2012.

[4] Wikipedia contributors. (13 November 2012 21:54 UTC). *LTE standard*. Available: http://en.wikipedia.org/w/index.php?title=LTE_(telecommunication)&oldid=522856142

[5] Wikipedia contributors. (14 November 2012 23:13 UTC). *3G technologies*. Available: http://en.wikipedia.org/w/index.php?title=3G&oldid=522631449

[6] Wikipedia contributors. (13 November 2012 21:44 UTC). *5G technologies*. Available: http://en.wikipedia.org/w/index.php?title=5G&oldid=518883254

[7] X. Li, A. Gani, R. Salleh, and O. Zakaria, "The future of mobile wireless communication networks," in *Proc. International Conference on Communication Software and Networks (ICCSN)*, 2009, pp. 554-557.

[8] Wikipedia contributors. (13 November 2012 21:14 UTC). *Commun. system*. Available: http://en.wikipedia.org/w/index.php?title=Communications_system&oldid=513526649

[9] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*: Cambridge University Press, 2009.

[10] S. Lin and D. J. Costello, *Error Control Coding*: Prentice-Hall, Inc., 2004.

[11] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*: Univ. Illinois Press, 1949.

[12] T. M. Cover and J. A. Thomas, *Elements of Information Theory*: Wiley-Interscience, 2006.

[13] F. Gray, "Pulse code communication," United States Patent, 1953.

[14] R. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 26, pp. 147-160, 1950.

[15] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300-304, 1960.

[16] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147-156, 1959 1959.

[17] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68-79, 3// 1960.

[18] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes (1)," in *Proc. IEEE International Conference on Communications*, 1993, pp. 1064-1070 vol.2.

[19] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 56-67, 1998.

[20] R. G. Gallager, *Low-Density Parity-Check Codes*: M.I.T. Press, 1963.

[21] "IEEE Standard for Information Technology--Local and Metropolitan Area Networks--Specific Requirements Part 3," *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)*, pp. c1-597, 2008.

[22] European Telecommunication Standardization Institute. (14 November 2012 23:13 UTC). *Digital Video Broadcasting (DVB)- A guideline for the use of the DVB specifications and standards*. Available: http://broadcasting.ru/pdf-standard-specifications/cookbook/dvb-cook/a020r1.pdf

[23] "IEEE Standard for Information technology--Local and metropolitan area networks--Specific requirements Part 11," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1-2793, 2012.

[24] F. Kienle, N. Wehn, and H. Meyr, "On Complexity, Energy- and Implementation-Efficiency of Channel Decoders," *IEEE Transactions on Communications*, vol. 59, pp. 3301-3310, 2011.

[25] Wikipedia contributors. (15 November 2012 02:14 UTC). *Communication*. Available: http://en.wikipedia.org/w/index.php?title=Communication&oldid=523065008

[26] M. C. Davey and D. MacKay, "Low-density parity check codes over GF(*q*)," *IEEE Communications Letters*, vol. 2, pp. 165-167, 1998.

[27] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(*q*)," *IEEE Transactions on Communications*, vol. 55, pp. 633-643, 2007.

[28] E. Arikan, "Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, pp. 3051-3073, 2009.

[29] S. B. Korada, E. Şaşoğlu, and R. Urbanke, "Polar codes: characterization of exponent, bounds, and constructions," *IEEE Transactions on Information Theory*, vol. 56, pp. 6253-6264, 2010.

[30] E. Arikan, "Systematic polar coding," *IEEE Communications Letters*, vol. 15, pp. 860-862, 2011.

[31] R. Mori and T. Tanaka, "Performance of polar codes with the construction using density evolution," *IEEE Communications Letters*, vol. 13, pp. 519-521, 2009.

[32] S. B. Korada and R. L. Urbanke, "Polar codes are optimal for lossy source coding," *IEEE Transactions on Information Theory*, vol. 56, pp. 1751-1768, 2010.

[33] O. O. Koyluoglu and H. El Gamal, "Polar coding for secure transmission and key agreement," *IEEE Transactions on Information Forensics and Security*, vol. 7, pp. 1472-1483, 2012.

[34] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Communications Letters*, vol. 12, pp. 447-449, 2008.

[35] H. Mahdavifar and A. Vardy, "Achieving the secrecy capacity of wiretap channels using Polar codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2010, pp. 913-917.

[36] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency SC polar decoder architectures," in *Proc. IEEE International Conference on Communications (ICC)*, 2012, pp. 3520-3524.

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

[38] J. Lin, J. Sha, Z. Wang, and L. Li, "An efficient VLSI architecture for nonbinary LDPC decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 57, pp. 51-55, 2010.

[39] J. Lin, J. Sha, Z. Wang, and L. Li, "Efficient decoder design for nonbinary quasicyclic LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 57, pp. 1071-1082, 2010.

[40] F. Naessens, A. Bourdoux, and A. Dejonghe, "A flexible ASIP decoder for combined binary and non-binary LDPC codes," in *Proc. IEEE Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, 2010, pp. 1-5.

[41] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, and P. Urard, "Architecture of a low-complexity non-binary LDPC decoder for high order fields," in *Proc. International Symposium on Communications and Information Technologies (ISCIT)*, 2007, pp. 1201-1206.

[42] X. Zhang and F. Cai, "Partial-parallel decoder architecture for quasi-cyclic non-binary LDPC codes," in *Proc. IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, 2010, pp. 1506-1509.

[43] X. Zhang and F. Cai, "Efficient partial-parallel decoder architecture for quasi-cyclic nonbinary LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 58, pp. 402-414, 2011.

[44] X. Zhang and F. Cai, "Reduced-complexity decoder architecture for non-binary LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 19, pp. 1229-1238, 2011.

[45] C. Zhang and K. K. Parhi, "A network-efficient nonbinary QC-LDPC decoder architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 59, pp. 1359-1371, 2012.

[46] L. Zeng, "Algebraic Constructions of Nonbinary Quasi-Cyclic LDPC Codes and Efficient Encoding," University of California, Davis, 2006.

[47] L. Zeng, L. Lan, Y. Y. Tai, S. Song, S. Lin, and K. Abdel-Ghaffar, "Constructions of nonbinary quasi-cyclic LDPC codes: a finite field approach," *IEEE Transactions on Communications,* vol. 56, pp. 545-554, 2008.

[48] L. Zeng, L. Lan, Y. Y. Tai, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "Construction of nonbinary cyclic, quasi-cyclic and regular LDPC codes: a finite geometry approach," *IEEE Transactions on Communications,* vol. 56, pp. 378-387, 2008.

[49] B. Zhou, L. Zhang, J. Kang, Q. Huang, S. Lin, and K. Abdel-Ghaffar, "Array dispersions of matrices and constructions of quasi-cyclic LDPC codes over non-binary fields," in *Proc. IEEE International Symposium on Information Theory*, 2008, pp. 1158-1162.

[50] B. Zhou, L. Zhang, J. Kang, Q. Huang, Y. Y. Tai, S. Lin, *et al.*, "Non-binary LDPC codes vs. Reed-Solomon codes," in *Proc. Information Theory and Applications Workshop*, 2008, pp. 175-184.

[51] S. Song, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "A unified approach to the construction of binary and nonbinary quasi-cyclic LDPC codes based on finite fields," *IEEE Transactions on Communications,* vol. 57, pp. 84-93, 2009.

[52] B. Zhou, J. Kang, Y. Tai, S. Lin, Lin, and Z. Ding, "High performance non-binary quasi-cyclic LDPC codes on Euclidean geometries," *IEEE Transactions on Communications,* vol. 57, pp. 1298-1311, 2009.

[53] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. Xu, "Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions," *IEEE Transactions on Communications*, vol. 57, pp. 1652-1662, 2009.

[54] B. Zhou, "Algebraic Constructions of High Performance and Efficiently Encodable Non-binary Quasi-cyclic LDPC Codes," University of California, Davis, 2008.

[55] K. Jingyu, H. Qin, Z. Li, Z. Bo, and L. Shu, "Quasi-cyclic LDPC codes: an algebraic construction," *IEEE Transactions on Communications*, vol. 58, pp. 1383-1396, 2010.

[56] Z. Bo, Y. Y. Tai, L. Lan, S. Song, Z. Lingqi, and L. Shu, "Construction of high performance and efficiently encodable nonbinary quasi-cyclic LDPC codes," in *Proc. Global Telecommunications Conference (GLOBECOM)*, 2006, pp. 1-6.

[57] I. Tal and A. Vardy. (2012, May 1, 2012). List decoding of polar codes. *ArXiv e-prints 1206*, 50. Available: http://adsabs.harvard.edu/abs/2012arXiv1206.0050T

[58] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Transactions on Signal Processing*, in revision.

[59] C. Zhang and K. K. Parhi, "Interleaved successive cancellation polar decoders," in *Proc. IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, 2013, submitted.

[60] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, pp. 1378-1380, 2011.

[61] C. Zhang and K. K. Parhi, "Latency analysis and architecture design of simplified SC polar decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, in revision.

[62] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, pp. 457-458, 1997.

[63] "IEEE Standard for Information technology--Specific requirements-- Part 11:," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1-565, 2009.

[64] "IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2," *IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005 (Amendment and Corrigendum to IEEE Std 802.16-2004)*, pp. 0_1-822, 2006.

[65] SARFT. (15 November 2012 22:14 UTC). *China Mobile Multimedia Broadcasting (CMMB)*. Available: http://www.cmmb.org.cn/

[66] DTMB. (15 November 2012 22:14 UTC). *Framing Structure, Channel Coding and Modulation for Digital Television Tterrestrial Broadcasting System (DTTV)*. Available: http://www.sac.gov.cn/SACSearch/outlinetemplet/gjcxjg_qwyd.jsp?bzNum=GB20600-2006

[67] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *Proc. IEEE International Conference on Communications*, 2004, pp. 772-776.

[68] C. Spagnol, E. M. Popovici, and W. P. Marnane, "Hardware implementation of GF(2^m) LDPC decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, pp. 2609-2620, 2009.

[69] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low-complexity decoding for non-binary LDPC codes in high order fields," *IEEE Transactions on Communications*, vol. 58, pp. 1365-1375, 2010.

[70] E. Boutillon and L. Conde-Canencia, "Bubble check: a simplified algorithm for elementary check node processing in extended min-sum non-binary LDPC decoders," *Electronics Letters*, vol. 46, pp. 633-634, 2010.

[71] V. Savin, "Min-Max decoding for non binary LDPC codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2008, pp. 960-964.

[72] D. Oh and K. K. Parhi, "Min-Sum decoder architectures with reduced word length for LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, pp. 105-115, 2010.

[73] D. Oh and K. K. Parhi, "Low-complexity switch network for reconfigurable LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 85-94, 2010.

[74] D. Oh and K. Parhi, "Low complexity decoder architecture for low-density parity-check codes," *Journal of Signal Processing Systems*, vol. 56, pp. 217-228, 2009.

[75] Z. Cui, Z. Wang, and Y. Liu, "High-throughput layered LDPC decoding architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 582-587, 2009.

[76] Z. Cui and Z. Wang, "Extended layered decoding of LDPC codes," in *Proc. ACM Great Lakes symposium on VLSI*, Orlando, Florida, USA, 2008, pp. 457-462.

[77] T. Lehnigk-Emden and N. Wehn, "Complexity evaluation of non-binary Galois field LDPC code decoders," in *Proc. International Symposium on Turbo Codes and Iterative Information*, 2010, pp. 53-57.

[78] S. Zhou, J. Sha, L. Li, and Z. Wang, "Layered decoding for non-binary LDPC codes," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 481-484.

[79] D. Oh and K. K. Parhi, "Area efficient controller design of barrel shifters for reconfigurable LDPC decoders," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2008, pp. 240-243.

[80] J. Tang, T. Bhatt, V. Sundaramurthy, and K. K. Parhi, "Reconfigurable shuffle network design in LDPC decoders," in *Proc. International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2006, pp. 81-86.

[81] C. Y. Lee and J. M. Tsai, "A shift register architecture for high-speed data sorting," *Journal of VLSI Signal Processing*, vol. 11, pp. 273-280, Dec 1995.

[82] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261-1271, 1996.

[83] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, 1994, pp. 1298-1303 vol.3.

[84] D. Arnold and G. Meyerhans, *The Realization of the Turbo-Coding System*: Swiss Fed. Inst. of Tech., 1995.

[85] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533-547, 1981.

[86] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," in *Proc. IEEE International Symposium on Information Theory*, 1997, p. 113.

[87] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 56, pp. 2634-2643, 2009.

[88] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE International Conference on Communications (ICC)*, 1995, pp. 1009-1013 vol.2.

[89] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory,* vol. 47, pp. 498-519, 2001.

[90] Z. Wang and K. K. Parhi, "High performance, high throughput turbo/SOVA decoder design," *IEEE Transactions on Communications,* vol. 51, pp. 570-579, 2003.

[91] Z. Wang, Z. Chi, and K. K. Parhi, "Area-efficient high-speed decoding schemes for turbo decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 10, pp. 902-912, 2002.

[92] 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)*, 2011, pp. 1665-1668.

[93] A. Pamuk, "An FPGA implementation architecture for decoding of polar codes," in *International Symposium on Wireless Communication Systems (ISWCS)*, 2011, pp. 437-441.

[94] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *Journal of Signal Processing Systems for Signal Image and Video Technology,* vol. 69, pp. 305-315, Dec 2012.

[95] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers,* vol. C-36, pp. 24-35, 1987.

[96] K. Ito and K. K. Parhi, "Determining the minimum iteration period of an algorithm," *J. VLSI Signal Process. Syst.,* vol. 11, pp. 229-244, 1995.

[97] K. Ito, L. E. Lucke, and K. K. Parhi, "ILP-based cost-optimal DSP synthesis with module selection and data format conversion," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 6, pp. 582-594, 1998.

[98] T. C. Denk and K. K. Parhi, "Synthesis of folded pipelined architectures for multirate DSP algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 6, pp. 595-607, 1998.

[99] K. K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Transactions on Circuits and Systems,* vol. 38, pp. 745-754, 1991.

[100] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters, part I: Pipelining using scattered look-ahead and decomposition," *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. 37, pp. 1099-1117, 1989.

[101] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters, part II: Pipelined incremental block filtering," *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. 37, pp. 1118-1134, 1989.

[102] Wikipedia contributors. (14 November 2012 20:33 UTC). *Adder-subtractor*. Available: http://en.wikipedia.org/w/index.php?title=Adder%E2%80%93subtractor&oldid=515321310

[103] Wikipedia contributors. (14 November 2012 20:41 UTC). *Kronecker product*. Available: http://en.wikipedia.org/w/index.php?title=Kronecker_product&oldid=509224913

[104] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Transactions on Computers,* vol. 40, pp. 178-195, 1991.

[105] M. Ayinala, M. Brown, and K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 20, pp. 1068-1081, 2012.

[106] V. Sundararajan and K. K. Parhi, "Synthesis of low power folded programmable coefficient FIR digital filters," in *Proc. Asia and South Pacific Design Automation Conference*, 2000, pp. 153-156.

[107] S. B. Korada, "Polar Codes for Channel and Source Coding," EPFL, 2009.