AN FPGA IMPLEMENTATION OF SUCCESSIVE CANCELLATION LIST DECODING FOR POLAR CODES

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Altuğ Süral

January 2016

An FPGA Implementation of Successive Cancellation List Decoding for Polar Codes
By Altuğ Süral
January 2016

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Erdal Arıkan(Advisor)

Orhan Arıkan

Ali Ziya Alkar

Approved for the Graduate School of Engineering and Science:

Levent Onural
Director of the Graduate School

ABSTRACT

AN FPGA IMPLEMENTATION OF SUCCESSIVE CANCELLATION LIST DECODING FOR POLAR CODES

Altuğ Süral
M.S. in Electrical and Electronics Engineering
Advisor: Erdal Arıkan
January 2016

Polar Codes are the first asymptotically provably capacity achieving error correction codes under low complexity successive cancellation (SC) decoding for binary discrete memoryless symmetric channels. Although SC is a low complexity algorithm, it does not provide as good performance as a maximum-likelihood (ML) decoder, unless sufficiently large code block is used. SC is a soft decision decoding algorithm such that it employs depth-first searching method with a divide and conquer approach to find a sufficiently perfect estimate of decision vector. Using SC with a list (SCL) improves the performance of SC decoder such that it provides near ML performance. SCL decoder employs beam search method as a greedy algorithm to achieve ML performance without considering all possible codewords. The ML performance of polar codes is not good enough due to the minimum hamming distance of possible codewords. For the purpose of increasing the minimum distance, cyclic redundancy check aided (CRC-SCL) decoding algorithm can be used. This algorithm makes polar codes competitive with state of the art codes by exchanging complexity with performance. In this thesis, we present an FPGA implementation of an adaptive list decoder; consisting of SC, SCL and CRC decoders to meet with the tradeoff between performance and complexity.

Keywords: Polar codes, successive cancellation list decoder, hardware implementation, FPGA.

ÖZET

SIRALI ELEMELİ VE LİSTELİ KUTUPSAL KODÇÖZÜCÜ'NÜN FPGA UYGULAMASI

Altuğ Süral
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Erdal Arıkan
Ocak 2016

Kutupsal kodların ayrık hafızasız kanallarda asimtotik olarak kanal kapasitesine sıralı elemeli (SC) kodçözücü ile eriştiği kanıtlanmıştır. SC düşük karmaşıklı bir algoritmadır ve bu algoritma ile yüksek blok uzunluğunda kodlar kullanılmadığı taktirde azami ihtimaliyet tahmini (ML) performansı elde edilmez. SC algoritması, böl ve fethet yöntemini kullanarak ve derinlik öncelikli arama yaparak karar verir. SC algoritması ile liste yapısı (SCL) birlikte kullanılarak ML performansına yaklaşılır. SCL algoritması açgözlü demet araması yapar. Ancak, kutupsal kodların olası kod sözcüklerinin en yakın hamming uzaklığı ve dolayısıyla ML performansı yeterince iyi değildir. Bu durumun üstesinden gelmek için döngüsel artıklık denetimi ile SCL algoritması (CRC-SCL) birleştirilir. Bu sayede, kutupsal kodlar güncel haberleşme sistemlerinde kullanılan kodlar ile rekabet eder hale gelir. Biz bu tezde, yüksek karmaşıklıklı ve yavaş çalışan CRC-SCL ile düşük karmaşıklıklı ve hızlı çalışan SC algoritmalarını FPGA uygulaması ile birlikte kullanarak performans ve karmaşıklık arasında ödünleşim sağlıyoruz.

Anahtar sözcükler: Kutupsal kodlar, sıralı elemeli ve listeli kodçözücü, donanım uygulaması, FPGA.

Acknowledgement

I would like to thank my supervisor, Prof. Erdal Arıkan for his persistent support, invaluable guidance, encouragement and endless patience during my thesis.

I express deep and sincere gratitude to Prof. Orhan Arıkan and Dr. Ali Ziya Alkar for their valuable suggestions and kindness.

I also thank Bilkent University for providing me an essential opportunity with a sophisticated research environment.

It is my privilege to have a supportive and lovely mother, Defne Süral. Without her supports, I would not complete my thesis.

I am extremely lucky to be with Gökçe Tuncer, who has a big heart and an agile mind. I would like to thank her for some ideas to improve my thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

B-DMC binary discrete memoryless channel.

BAWGNC binary additive white Gaussian noise channel.

BEC binary erasure channel.

BER bit error rate.

BPSK binary phase shift keying.

BRAM block random access memory.

BS bitonic sorter.

BSC binary symmetric channel.

CC clock cycle.

CL control logic.

CRC cyclic redundancy check.

DC decoding cycle.

DU decision unit.

Eb/No energy per bit to noise power spectral density ratio.

FBS fast bitonic sorter.

FEC forward error correction.

FER frame error rate.

FF flip-flop.

FFT fast fourier transform.

FPGA field-programmable gate array.

FRBS fast reduced bitonic sorter.

GF(2) binary glois field.

HRE hard decision router element.

LL log-likelihood.

LLR log-likelihood ratio.

LPE list processing element.

LPSU list partial sum update.

LPU list processing unit.

LR likelihood ratio.

LSB least significant bit.

LUT lookup table.

MAP maximum a-posteriori.

ML maximum likelihood.

MSB most significant bit.

PE processing element.

PSU partial sum update.

PU processing unit.

REP repitition code.

SC successive cancellation.

SCD successive cancellation decoding.

SCL successive cancellation list.

SCLD successive cancellation list decoding.

SM sign-magnitude.

SNR signal to noise ratio.

SPC single parity check code.

SRE soft decision router element.

TC twos complement.

VHDL very high speed integrated circuit hardware description language.

List of Symbols

C latency of the CRC decoder.

K number of information (free) bits, NR.

L list size.

n code block width, $\log_2 N$.

N code block length.

P soft decision bit precision.

R code rate.

V number of list processing units.

Chapter 1

Introduction

Shannon defines channel capacity as the maximum rate of information, which can be reliably transmitted over a communication channel [1]. He also shows that the channel capacity can be achieved by a random code construction method. With a code rate smaller than the channel capacity, a communication system encounters negligible errors. It has always been a challenge to achieve channel capacity with low complexity algorithms. Polar coding is a method that achieves channel capacity with low complexity encoding and decoding.

1.1 What are Polar Codes?

Polar codes are a class of capacity-achieving linear forward error correction (FEC) block codes [2]. The complexity of both encoding and successive cancellation (SC) decoding of polar codes are $O(N \log N)$, where N is the code block length. The recursive construction of both encoder and the SC decoder enables neat processing structures, component sharing and an efficient utilization of the limited sources. Polar codes provide a flexible selection of the code rate with $1/N$ precision such that an arbitrary code rate can be used without reconstructing the code. Polar codes are channel specific codes that means a polar code designed for a particular

channel might not has a good performance for other channels. The important properties of polar codes are:

- Capacity achieving error correction performance.
- $\mathcal{O}(N \log N)$ encoding and SC decoding complexity.
- Enhanced bit error rate (BER) with systematic polar codes.
- Adjustable code rate with $\frac{1}{N}$ precision without generating the code again.
- Channel specific recursive code construction.
- Achieves maximum likelihood (ML) performance by successive cancellation list (SCL) decoding with a sufficiently large list size.
- Block error probability of the SC decoder is asymptotically smaller than $2^{-\sqrt{N}}$ [3].

Decoding of polar codes is an active research problem. There are several decoding methods in literature such that these methods are SC [2], SCL [4], SC stack [5] and belief propagation [6]. The scope of this thesis includes SC and SCL methods. Due to low complexity encoding and decoding methods, implementation of polar codes at long block lengths is feasible for practical communication systems. In contrast, the noticeable concern for polar codes at long block lengths is the decoding latency due to strong data dependencies. In this thesis, we consider moderate code block lengths such as 1024 and 256 in order to enhance finite length performance with limited latency and resource usage. Although the SC decoder asymptotically achieves channel capacity as N increases, the superior performance of the SC decoder decays at short and moderate code block lengths due to poor polarization. For this reason, the SC decoder does not provide as good performance as a maximum likelihood (ML) decoder at short and moderate block lengths. To overcome this issue, it is necessary to add more features to the algorithm such as tracking multiple possible decision paths instead of one such that the SC decoder does. At this point, SCL decoding algorithm emerges

[4]. This algorithm uses beam search method for exploring the possible decoding paths efficiently. It is considerable as a greedy algorithm that approaches ML performance with sufficiently large list size, L. Since considering all 2^{NR} possible decoding paths is impractical and too complex, the SCL algorithm has a restricted complexity such that at most L best possible paths can be traced. In that way, the algorithm operates with $O(L\ N\ \log N)$ computational complexity. The error correction performance is further improved by combining the SCL algorithm with a cyclic redundancy check (CRC) code. At the end of decoding, the SCL decoder selects a CRC valid path from among L surviving paths.

1.2 Summary of Main Results

Polar codes are proved that they achieve channel capacity under SC decoding algorithm for symmetric binary discrete memoryless channels (B-DMCs) [2]. Due to the sequential nature of the SC decoding algorithm, the hardware implementation is significantly challenging. In this thesis, we try to overcome this issue by dividing the algorithm into simpler modules. SC algorithm provides low complexity $O(N\ \log N)$ decoding, however it does not provide as good performance as a ML decoder at short and moderate code block lengths. The performance of the SC algorithm can be improved by using SCL decoding algorithm, which tracks L best decoding paths together. The performance can be further improved by introducing CRC to SCL decoding by selecting a CRC valid path among L best decoding paths at the end of decoding. However, SCL decoding algorithm suffers from long latency and low throughput due to high complexity, $\mathcal{O}(L\ N\ \log N)$ calculations as L and N increases. The throughput of the SCL can be improved by using an adaptive decoder, which provides the SC throughput with the SCL performance. Data flow of the adaptive decoder is shown in Figure 1.1.

The adaptive SCL decoder has three main components, SC, SCL and CRC decoders. Initially, the SC decoder is activated and a hard decision estimate vector is calculated. After that, the CRC decoder controls whether the hard decision vector is correct. If the CRC is valid, the hard decision vector is quite

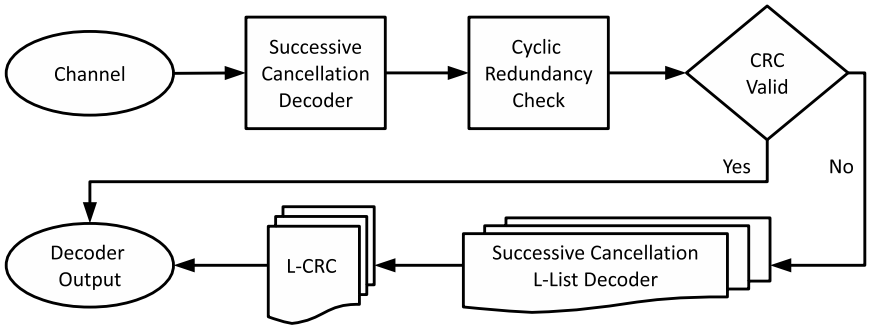


Figure 1.1: Data flow of the adaptive decoder.

likely to be the correct information vector. In this case, the adaptive decoder is immediately terminated without the activation of the SCL decoder. In other case, when the CRC is invalid, the SCL decoder is activated and L information vector candidates are generated. Among these candidates, the CRC decoder selects a candidate, which has a valid CRC vector. If more than one CRC vector candidate is valid, the most probable one among these candidates is selected. Lastly, when none of the candidates has a valid CRC vector, the CRC decoder selects the most probable decision estimation vector to reduce BER.

We have implemented an adaptive SCL decoder on Xilinx Kintex-7 (xc7k325t-2ffg900c) field-programmable gate array (FPGA). We have used Xilinx KC705 evaluation kit to verify our implementation. We have used Xilinx ISE 14.7 XST for synthesis and implementation of our design. To analyze each submodule in detail, we present either the implementation results or synthesis results for the ones which is not implementable as standalone due to large parallelization.

The effect of list size on the frame error rate (FER) performance of polar codes is shown in Figure 1.2. In this simulation, binary phase shift keying (BPSK) modulated symbols are transmitted over binary additive white Gaussian noise channel (BAWGNC). There is significant performance gain, which is more than 1 dB between SC decoder and SCL decoder. When the list size increases the performance improves, however the rate of improvement decreases.

As a result of our FPGA implementation of the adaptive SCL decoder, we have achieved 225 Mb/s data throughput with a reasonable resource usage.

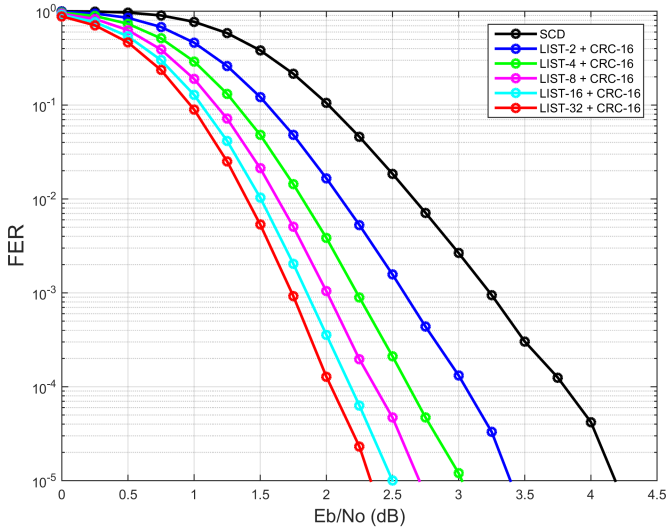


Figure 1.2: FER performance of the SCL decoder $N = 1024$, $K = 512$.

1.3 Outline of Thesis

In Chapter 2, we will review the polar codes in terms of properties, construction, encoding and decoding features. In Chapter 3, we will present details of our adaptive list decoder implementation on FPGAs. Finally, we will summarize main results of this thesis and express future work ideas in Chapter 4.

Chapter 2

Polar Codes

In this chapter, we will summarize the properties, construction, encoding and decoding methods of polar codes.

2.1 Notations

Upper case italic letters, such as X and Y denotes random variables and their realizations are denoted by lower case italic letters (e.g., x, y). A length-N row vector of u is denoted by u^N and its sub-vector $(u_i, u_{i+1}, ..., u_j)$ is denoted by u_i^j. Uppercase calligraphic symbols denote sets (e.g., $\mathcal{X}$, $\mathcal{Y}$). Time complexity of an algorithm is denoted by Υ and space complexity is denoted by ζ. Logarithm of base-2 and natural logarithm is represented by $\log(\cdot)$ and $\ln(\cdot)$, respectively. The logarithmic likelihood information is represented by δ_0 for $\ln(W(y|x=0))$ and δ_1 for $\ln(W(y|x=1))$, where x denotes the encoder output and y denotes the output of channel, W. The ratio of $\frac{\delta_0}{\delta_1}$ is called log-likelihood ratio (LLR) and it is represented by λ.

2.2 Preliminaries

In this section, we review the basic definitions of polar codes in [2].

Definition 1. Binary discrete memoryless channel (B-DMC). A generic memoryless channel W with input alphabet $\mathcal{X}$, output alphabet $\mathcal{Y}$ and transition probabilities $W(y|x),\ x \in \mathcal{X},\ y \in \mathcal{Y}$ is denoted as $W : \mathcal{X} \to \mathcal{Y}$. The input alphabet can take $\{0, 1\}$ binary values, however the output alphabet and the transition probabilities are continues such that they can take any arbitrary values in $[0, 1]$ interval.

Definition 2. A channel W is defined as symmetric if there exists a permutation π such that $\pi^{-1} = \pi$ and $W(y|1) = W(\pi(y)|0), \forall y \in \mathcal{Y}$.

Definition 3. A channel W is defined as memoryless if its transition probability is

$$W_{Y|X}(y|x) = \prod_{\forall i} W_{Y_i|X_i}(y_i|x_i). \tag{2.1}$$

Note that the BSC(p), the BEC(ϵ) and the BAWGNC(σ) are all memoryless.

Definition 4. Symmetric capacity of a B-DMC W is defined as

$$I(W) \triangleq \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)}. \tag{2.2}$$

Note that the symmetric capacity is the measure of rate. For symmetric channels, $I(W)$ equals to the Shannon capacity, which is the upper bound on the code rate to provide reliable communication.

Definition 5. Bhattacharyya parameter of a B-DMC W is defined as

$$Z(W) \triangleq \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)}. \tag{2.3}$$

Note that the Bhattacharyya parameter is the measure of reliability and it is an upper bound on the probability of ML decision.

Definition 6. The Kronecker product of m x n matrix A and q x p matrix B is the (mp) x (nq) block matrix C that defined as

$$C \triangleq A \otimes B = \begin{bmatrix} A_{11}B & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{bmatrix}. \tag{2.4}$$

In addition to that, n^{th} Kronecker power of a matrix A is defined as $A^{\otimes n} \triangleq A \otimes A^{\otimes(n-1)}$.

2.3 Channel Polarization

Channel polarization operation creates N synthetic channels $\{W_N^{(i)} : 1 \leq i \leq N\}$ from N independent copies of the B-DMC W [2]. Polarization phenomenon decomposes the symmetric capacity, $I(W_N^{(i)})$ of these synthetic channels towards to 0 or 1 such that $I(W_N^{(i)}) \simeq 0$ implies that the i^{th} channel is completely noisy and $I(W_N^{(i)}) \simeq 1$ implies that the i^{th} channel is perfectly noiseless. The capacity separation enables to send information (free) bits through the noiseless channels and redundancy (frozen) bits through the noisy channels.

Let $\mathcal{A}$ be the information set and $\mathcal{A}^c$ be the frozen set. The input vector, u_1^N consists of both information bits $u_{\mathcal{A}}$ and frozen bits $u_{\mathcal{A}^c}$ such that $u_{\mathcal{A}} \in \mathcal{X}^K$ and $u_{\mathcal{A}^c} \in \mathcal{X}^{N-K}$.

In the following sections, we present the channel polarization as channel combining, channel splitting and channel construction.

2.3.1 Channel Combining

A B-DMC W_N is generated by combining two independent copies of $W_{N/2}$. Initially, W_2 has the following transition probabilities

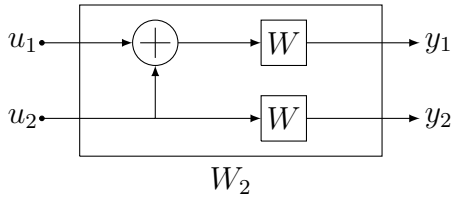


Figure 2.1: Construction of W_2 from W.

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2), \tag{2.5}$$

where W denotes the smallest B-DMC, $\{0, 1\} \to \mathcal{Y}$.

In a similar way, W_N is constructed recursively from $W_{N/2}$, $W_{N/4}$, ..., W_2, W at n steps, where $N = 2^n$.

2.3.2 Channel Splitting

A combined B-DMC W_2 is split back into two channels $W_2^{(\uparrow)}$ and $W_2^{(\downarrow)}$ by channel splitting operation. The transition probabilities of these channels are

$$W_2^{(\uparrow)}(y_1^2|u_1) = \frac{1}{2} \sum_{u_2 \in \{0,1\}} W(y_1|u_1 \oplus u_2)W(y_1|u_2) \tag{2.6}$$

$$W_2^{(\downarrow)}(y_1^2, u_1|u_2) = \frac{1}{2} W(y_1|u_1 \oplus u_2)W(y_2|u_2). \tag{2.7}$$

The transition probabilities are calculated in consecutive order from the top splitting operation to the bottom splitting operation, because the decision bit u_1 must be known before the bottom splitting operation.

2.3.3 Code Construction

The aim of the polar code construction is to determine $\mathcal{A}$ and $\mathcal{A}^c$ sets according to the capacity of individual channels. Since polar codes are channel specific codes, the code construction may differ from channel to channel. Channel parameters, such as σ for BAWGNC and ϵ for binary erasure channel (BEC) are an input to a code construction method. For BEC W, code construction for ($N = 8$, $K = 4$) polar code with an erasure probability $\epsilon = 0.3$ is shown in Figure 2.2.

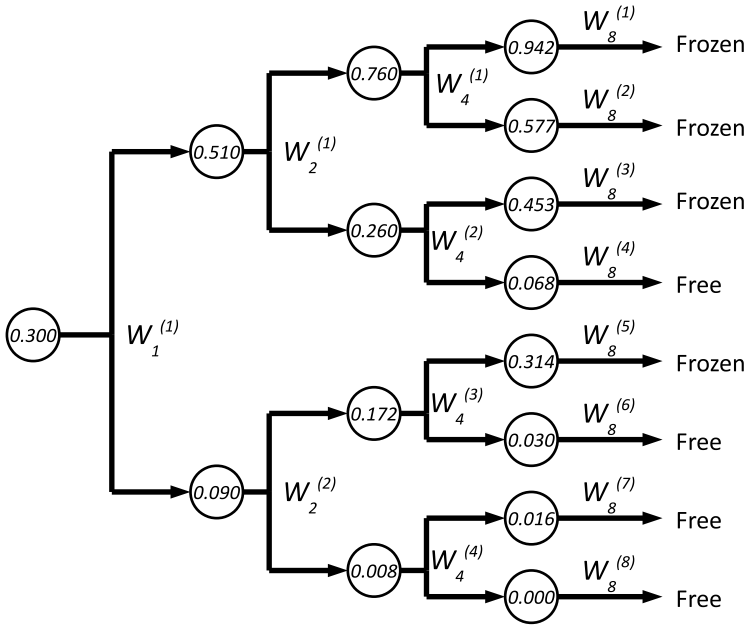


Figure 2.2: Code construction for BEC with $N = 8$, $K = 4$ and $\epsilon = 0.3$.

Initially, the reliability of the smallest channel, $W_1^{(1)}$ sets as $\epsilon = 0.3$. After that the reliability of the first length-2 channel, $W_2^{(1)}$ is calculated as $Z(W_2^{(1)}) = 2Z(W_1^{(1)}) - Z(W_1^{(1)})^2$, where $Z(W_N^{(i)})$ is the erasure probability of the i^{th} length-N channel starting from top. At the same time, the second length-2 channel can be calculated as $Z(W_2^{(2)}) = Z(W_1^{(1)})^2$. In general, the recursive formula for calculating top and bottom channels is

$$Z(W_N^{(2i-1)}) = 2Z(W_{N/2}^{(i)}) - Z(W_{N/2}^{(i)})^2 \tag{2.8}$$

$$Z(W_N^{(2i)}) = Z(W_{N/2}^{(i)})^2. \tag{2.9}$$

At the end of stage $\log N$ (in this case $\log N = 3$), the erasure probability of all length-N channels appears. At this point, the channels which has the lowest K erasure probabilities set as free and others set as frozen. The algorithm has $\log N$ stages and performs $N - 2$ calculations. Polar code construction for symmetric B-DMC is performed by using several methods such as Monte-Carlo simulation [2], density evolution [7] and Gaussian approximation [8]. In this thesis, we use Monte-Carlo simulation method with ten million trial numbers to determine $\mathcal{A}$ and $\mathcal{A}^c$ sets.

2.4 Encoding of Polar Codes

Polar codes can be encoded by using simple linear mapping. For the code block length N the generator matrix, G_N is defined as $G_N = B_N F^{\otimes n}$ for any $N = 2^n$ as $n \geq 1$, where B_N is a bit-reversal matrix and $F^{\otimes n}$ is the n^{th} Kronecker power of the matrix $F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$.

A length-N polar encoder has an input vector u_1^N and an output vector x_1^N. The mapping of $u \mapsto x$ is linear over the binary field, $\mathbb{F}_2$ such that $x_1^N = u_1^N G_N$. The rows of G_N are linearly independent and they form basis for the code space, $\mathcal{C}(N, 2^{\lfloor NR \rfloor})$.

The factor graph representation of 8-bit encoder is shown in Fig.2.3, where $\oplus$ symbol represents binary XOR operation.

As a result, the following equations are obtained at the output of 8-bit encoder.

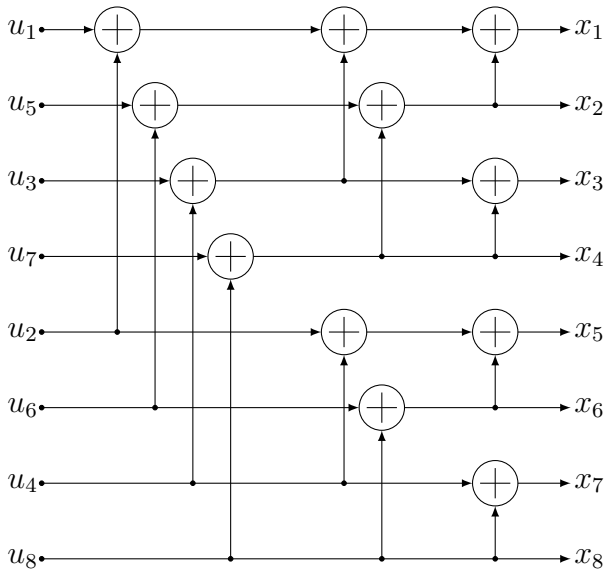


Figure 2.3: The factor graph representation of 8-bit encoder, G_8.

$$
\begin{aligned}
x_1 &= u_1 \oplus u_2 \oplus u_3 \oplus u_4 \oplus u_5 \oplus u_6 \oplus u_7 \oplus u_8 \\
x_2 &= u_5 \oplus u_6 \oplus u_7 \oplus u_8 \\
x_3 &= u_3 \oplus u_4 \oplus u_7 \oplus u_8 \\
x_4 &= u_7 \oplus u_8 \\
x_5 &= u_2 \oplus u_4 \oplus u_6 \oplus u_8 \\
x_6 &= u_6 \oplus u_8 \\
x_7 &= u_4 \oplus u_8 \\
x_8 &= u_8
\end{aligned}
\tag{2.10}
$$

The factor graph representation (Fig.2.3) shows that 8-bit encoder includes twelve XOR operations. In general, N-bit encoder includes $(N/2 \log N)$ XOR operations. Let $\Upsilon_E(N)$ denote the time complexity of encoding. Due to recursive channel combining, a length-N encoder consist of two length-$\frac{N}{2}$ encoders and $\frac{N}{2}$ binary XOR operations. Therefore, $\Upsilon_E(N)$ is

$$\Upsilon_E(N) = 2\Upsilon_E(\frac{N}{2}) + \frac{N}{2} \tag{2.11}$$

$$= 2\Upsilon_E(\frac{N}{2}) + \Theta(N) \tag{2.12}$$

$$\overset{\text{(i)}}{=} \mathcal{O}(N \log N), \tag{2.13}$$

where $\Upsilon_E(2) = 2$ and (i): the master theorem, case 2.

The free bits can be observable at the output of a polar encoder by systematic encoding of polar codes [9].

2.5 Successive Cancellation (SC) Decoding of Polar Codes

SC decoding of polar codes is a search problem [10] such that the target is to reconstruct information data from noisy channel output. The search space consists of all possible codewords, belong to the code space $\mathcal{C}(N, 2^{\lfloor K \rfloor})$, where $K = NR$. The decoding path can be realized as a reduced binary tree with 2^K leafs and N depth. Frozen bits are the cause of the reduction in this binary tree, because there is only one decision option at the i^{th} frozen decision step as $\hat{u}_i = u_i$ for $i \notin \mathcal{A}$.

Calculating likelihood of all possible codewords and finding the most probable codeword could be the first method that minimizes block error probability defined as $P_e = P\{\hat{u}_\mathcal{A} \neq u_\mathcal{A}\}$. This method is called the ML decoding and uses the British Museum procedure for searching all possible codewords. Although the ML decoding method provides decent error correction performance, exponential complexity makes this method impractical to implement.

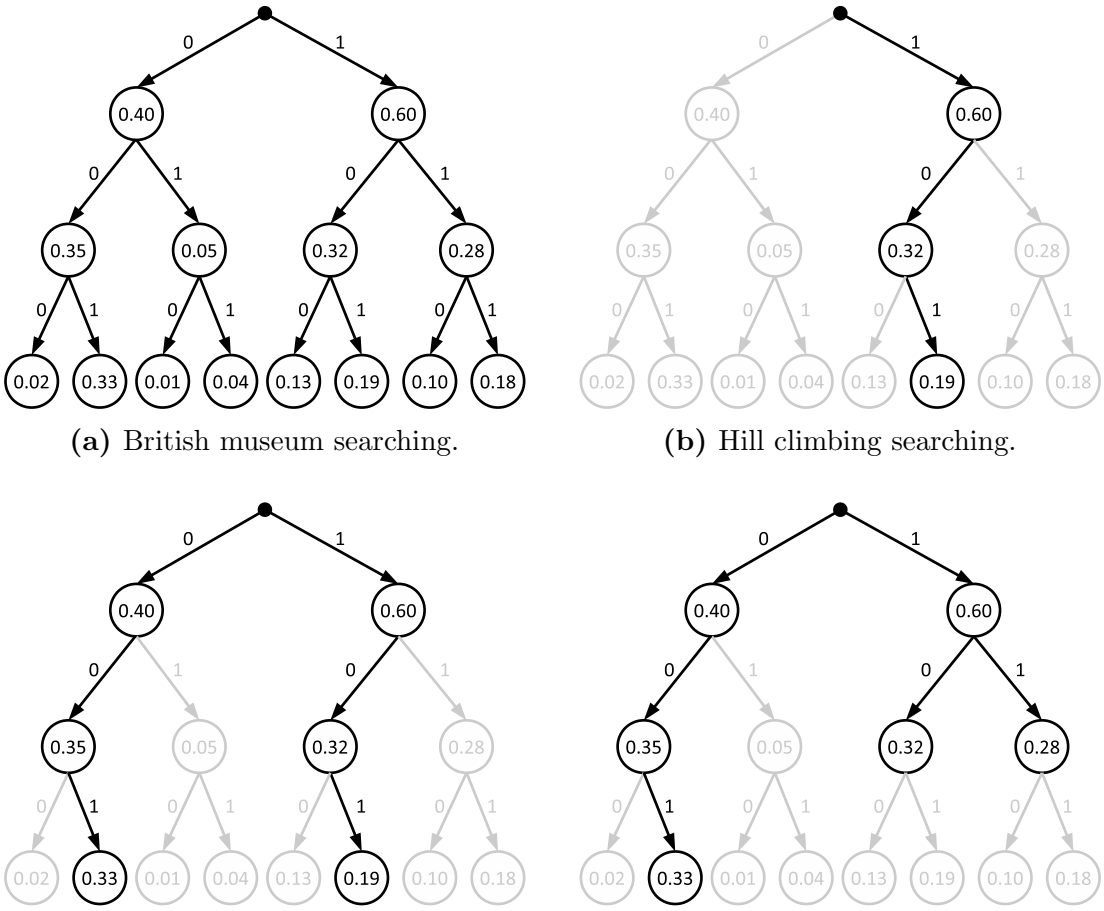
Furthermore, instead of searching for all possible codewords, searching for the

current best decoding path significantly reduces the decoding complexity. In this case, depth-first (hill climbing) search method is useful. At each decision step, there are two decision candidates: $\hat{u}_i = 0$ and $\hat{u}_i = 1$. The decision between these two candidates is made with respect to channel information and all previously decoded bit information. Due to the information gained at each decoding step, depth-first search becomes hill climbing search and the decoder is not allowed to change its previous decisions according to current information bits. Although this restriction reduces the error correction performance of the SC decoder especially in difficult terrains (low signal to noise ratio (SNR) values), the decoder has reasonable $\mathcal{O}(N \log N)$ complexity. The performance loss is caused by local best decoding paths, which misguides the decoder to an incorrect decoding path. The original low complexity SC decoder, [2] uses hill climbing search method for decoding polar codes.

An another method to find a better decoding path is beam searching. It enables to trace L best decoding paths instead of one in depth-first search. The beam search method has restricted complexity compared to breadth-first search, which traces all decoding paths at each decoding layer. The SCL algorithm uses the beam searching method for decoding of polar codes. At each decision level, a sorting algorithm finds the best L decoding paths for among $2L$ possible decoding path candidates. Although the decoder complexity increases to $\mathcal{O}(L\ N\ \log N)$ [4], it has a noticeable performance gain. The performance gain with respect to L is illustrated in Section 2.6.

Lastly, the best search method reduces the complexity of beam search by tracing the best L encountered node so far. This can produce partially developed decoding tree. The best search method is also similar to hill climbing in terms of tracing the best path. The main difference between these methods is the best search method enables the decoder to change its previous decision according to current likelihood information. The stack SC algorithm, [5] uses the best search method to decode polar codes.

A decoding tree example, which consists of all of the mentioned search methods, is shown in Figure 2.4. In this example, the black paths represent the visited



(a) British museum searching.

(b) Hill climbing searching.

(c) Beam searching, $L = 2$.

(d) Best searching, $L = 2$.

Figure 2.4: An example decoding tree representation for searching methods.

paths and the gray paths represent the ignored decoding paths. All decoding paths are possible, because all decisions set as free. Likelihoods of decoding paths are written inside circles and the values on the arrows are the hard decisions at each decision step. By using the British museum search method (Fig. 2.4a), all paths are visited. Therefore, the hard decision is the most probable path at the end of decoding, which is 001 with a probability 0.33. On the other hand, the hill climbing search (Fig. 2.4b) ends up with a different hard decision, 101 that has a lower likelihood probability, 0.19. The reason is the first decoding step such that the SC decoder selects the local best path, however it turns out to be a better path at the end of the binary decision tree. Unlike the British museum searching, beam searching traces only two most probable paths instead of eight and still explores the most probable path (Fig. 2.4c). Lastly, the best search starts with the paths which has 0.60 and 0.40 probabilities, then explores the path which has 0.32 probability. Since it is smaller than the stack, the algorithm explores the other nodes which have 0.28 and 0.35 probabilities respectively. At the end of best searching, the algorithm reveals the most probable decoding path, which has 0.33 probability.

In the following sections, we will introduce SC and SCL algorithms.

2.5.1 Successive Cancellation Decoding of Polar Codes

Polar SC decoder estimates the transmitted bits u_1^N as $\hat{u}_1^N$ by using the received codeword $y \in \mathcal{Y}$ at the B-DMC, $W_N : \mathcal{X}^N \to \mathcal{Y}^N$. The channel likelihood information gained from the received codeword is represented as LLR. The decoder performs soft-decision decoding as computing intermediate LLR values by using channel LLR values. After a sequence of LLR computations, SC decoder computes $\hat{u}_1^N$ hard decisions in a successive order from $\hat{u}_1$ to $\hat{u}_N$. In other words, $\hat{u}_i$ is decided according to $\hat{u}_1^{i-1}$ for $1 < i \leq N$. The time (Υ_{SCD}) and the space complexity (ζ_{SCD}) of the SC algorithm is

$$\Upsilon_{SCD}(N) = \mathcal{O}(N \ \log N), \tag{2.14}$$

$$\zeta_{SCD}(N) = \mathcal{O}(N). \tag{2.15}$$

A high level description of the SC decoding algorithm is illustrated in Algorithm 1. The algorithm takes the received codeword y_1^N, the code block length N and the information set $\mathcal{A}$ as input and calculates the estimated free bits $\hat{u}_{\mathcal{A}}$ as an output vector. There are N decision steps in the algorithm. If a hard decision belongs to the frozen set $\mathcal{A}^c$, the decision will be a frozen decision such that it is known by both encoder and decoder. Otherwise, the decoder sets its hard decision with respect to the soft decision information. After all N decisions are calculated, the output of the decoder is the hard decisions, which belong to the free set.

Algorithm 1: Successive Cancellation Decoding

Input: received codeword, y_1^N
Input: code block length, N
Input: information set, $\mathcal{A}$
Input: frozen bit vector, $u_{\mathcal{A}^c}$
Output: estimated free bits, $\hat{u}_{\mathcal{A}}$

```
begin
    for i ← 1 to N do
        if i ∉ A then
            û_i ⟵ u_i
        else
            if log( W_N^(i)(y_1^N, û_1^{i-1} | û_i=0) / W_N^(i)(y_1^N, û_1^{i-1} | û_i=1) ) ≥ 0 then
                û_i ⟵ 0
            else
                û_i ⟵ 1
    return û_A
```

More specifically, the channel LLR values, γ are calculated for BAWGNC as

$$\gamma = \ln\left(\frac{W(y|x=0)}{W(y|x=1)}\right) \tag{2.16}$$

$$= \ln\left(\frac{e^{\frac{-(y-1)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right) - \ln\left(\frac{e^{\frac{-(y+1)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right) \tag{2.17}$$

$$= \frac{-(y-1)^2}{2\sigma^2} - \frac{-(y+1)^2}{2\sigma^2} \tag{2.18}$$

$$= \frac{2y}{\sigma^2}, \tag{2.19}$$

where BPSK modulation with standard mapping is used to assign an output bit of encoder x to a transmitted symbol s. For $i = \{1 \leq i \leq N\}$, the mapping rule is

$$s_i = \begin{cases} 1, & \text{if } x_i = 0 \\ -1, & \text{if } x_i = 1. \end{cases} \tag{2.20}$$

Three different functions are defined to illustrate the behavior of the SC decoder. These functions are called f, g, and d. Firstly, the f function is responsible for the calculation of top channel splitting operation, defined in Section 2.3.2. The f function, with likelihood ratio (LR) representation is

$$f(LR_a, LR_b) = LR_c \tag{2.21}$$

$$= \frac{W(y_1^2|\hat{u}_c = 0)}{W(y_1^2|\hat{u}_c = 1)} \tag{2.22}$$

$$= \frac{W(y_1|\hat{u}_a = 0)W(y_2|\hat{u}_b = 0) + W(y_1|\hat{u}_a = 1)W(y_2|\hat{u}_b = 1)}{W(y_1|\hat{u}_a = 0)W(y_2|\hat{u}_b = 1) + W(y_1|\hat{u}_a = 1)W(y_2|\hat{u}_b = 0)} \tag{2.23}$$

$$\overset{\text{(ii)}}{=} \frac{LR_a\ LR_b + 1}{LR_a + LR_b}, \tag{2.24}$$

where (ii): both numerator denominator are divided by $W(y_1|\hat{u}_a = 1)W(y_2|\hat{u}_b = 1)$.

The f function, with LLR representation is

$$f(\gamma_a, \gamma_b) = \gamma_c \tag{2.25}$$

$$= 2\tanh^{-1}\left((\tanh\left(\frac{\gamma_a}{2}\right)\tanh\left(\frac{\gamma_b}{2}\right)\right) \tag{2.26}$$

$$\overset{[11],\,[12]}{\approx} \operatorname{sign}(\gamma_a\gamma_b)\min(|\gamma_a|, |\gamma_b|), \tag{2.27}$$

where min-sum approximation is defined for BP decoding of LDPC codes [11] and this approximation is used in SC decoding of polar codes for the first time [12]. The min-sum approximation causes an insignificant performance degradation, which will be shown in Section 2.6.2.

Secondly, the g function computes the bottom channel splitting operation in SC decoder. The g function, with LLR representation of soft decisions is

$$g(\delta_a, \delta_b, \hat{u}) = \delta_c \tag{2.28}$$

$$= \frac{W(y_1^2, \hat{u}|\hat{u}_c = 0)}{W(y_1^2, \hat{u}|\hat{u}_c = 1)} \tag{2.29}$$

$$= \begin{cases} \frac{W(y_1|\hat{u}_a=0)W(y_2|\hat{u}_b=0)}{W(y_1|\hat{u}_a=1)W(y_2|\hat{u}_b=1)}, & \text{if } \hat{u} = 0 \\ \frac{W(y_1|\hat{u}_a=1)W(y_2|\hat{u}_b=0)}{W(y_1|\hat{u}_a=0)W(y_2|\hat{u}_b=1)}, & \text{if } \hat{u} = 1 \end{cases} \tag{2.30}$$

$$= {\delta_a}^{(1-2\hat{u}_c)}\delta_b. \tag{2.31}$$

The g function, with LLR representation is

$$g(\gamma_a, \gamma_b, \hat{u}) = (-1)^{(\hat{u})}\gamma_a + \gamma_b. \tag{2.32}$$

Lastly, the d function is the decision function, which computes hard decisions

from soft decisions such that

$$\hat{u}_i = \begin{cases} u_i, & \text{if } i \notin \mathcal{A} \\ 0, & \text{if } i \in \mathcal{A} \text{ and } \frac{W(y,\hat{u}_1^{i-1}|\hat{u}_i=0)}{W(y,\hat{u}_1^{i-1}|\hat{u}_i=1)} \geq 1 \\ 1, & \text{otherwise.} \end{cases} \tag{2.33}$$

We will present hardware implementations of these functions in Section 3.2.

2.5.2 Successive Cancellation List (SCL) Decoding of Polar Codes

SCL decoding algorithm is proposed to enhance the performance of SC decoding for short and moderate block lengths in [4]. SCL decoding enables tracking L best decoding paths concurrently, unlike an SC decoder can track at most a single decoding path. If L is sufficiently large, ML decoding performance is achieved, since sufficient number of decoding paths are visited. There is a trade-off between complexity and performance of the algorithm, because time complexity (Υ_{SCLD}) and space complexity (ζ_{SCLD}) of the algorithm linearly depends on list size (L) such that

$$\Upsilon_{SCLD}(L, N) = \mathcal{O}(L\ N\ \log N), \tag{2.34}$$

$$\zeta_{SCLD}(L, N) = \mathcal{O}(L\ N). \tag{2.35}$$

A high level description of the algorithm is shown in Algorithm 2. The SCL algorithm takes the received codeword y_1^N, the code block length N, the information set $\mathcal{A}$, the frozen bit vector $u_{\mathcal{A}^c}$ and the maximum list size L as input and calculates the estimated information bits $\hat{u}_{\mathcal{A}}$ as output. The current list size variable, cL set as 1 at the initialization of the algorithm. If the i^{th} hard decision

belongs to the frozen set $\mathcal{A}^c$, the i^{th} hard decisions of all L lists are updated with the frozen decision, u_i. In case of a free decision, the decoder checks whether the current list size is equal to the maximum list size. If they are not equal, the current list size doubles and the decoder can track likelihoods of both decisions. In case of all lists are occupied, the decoder sorts $2L$ likelihoods to continue with the best L decoding paths. At the end of the last decision step, the decoder outputs the free bits from the best list as $\hat{u}_{\mathcal{A}}$.

Algorithm 2: Successive Cancellation List Decoding

Input: received codeword, y_1^N
Input: code block length, N
Input: information set, $\mathcal{A}$
Input: frozen bit vector, $u_{\mathcal{A}^c}$
Input: maximum list size, L
Output: estimated information bits, $\hat{u}_{\mathcal{A}}$
Variable: $cL \longleftarrow 1$ `//current list size`

```
1  begin
2      for i ← 1 to N do
3          if i ∉ A then
4              for l ← 1 to cL do
5                  û_{l,i} ⟵ u_i
6          else
7              if cL ≠ L then
8                  for l ← 1 to cL do
9                      û_{l,i} ⟵ 0
10                     û_{l+cL,i} ⟵ 1
11                 cL ⟵ 2cL
12             else
13                 s ⟵ sort (W_N^(i)(y_1^N, û_1^{i-1} | û_{1,i}^L))
14                 for l ← 1 to cL do
15                     û_{l,i} ⟵ s_l
16     return û_A
```

We will present a hardware implementation of the SCL decoding algorithm in Section 3.3.

2.5.3 Adaptive Successive Cancellation List Decoding of Polar Codes

The adaptive SCL algorithm consists of SC decoding, SCL decoding and CRC decoding algorithms. The aim of the algorithm is to increase the throughput of the SCL decoder in [13], [14]. A high level description of the algorithm is shown in Algorithm 3. Inputs of the adaptive SCL decoding algorithm are the received codeword y_1^N, the code block length N, the information set $\mathcal{A}$, the frozen bit vector $u_{\mathcal{A}^c}$ and the list size L. The output of the algorithm is the free bit vector $\hat{u}_{\mathcal{A}}$. At the beginning of the algorithm, the SC decoder calculates a free bit candidate vector. If the CRC of that vector is true, the algorithm terminates with the output of the SC decoder. In case of incorrect CRC vector, the algorithm calls a SCL algorithm with the list L. At this time, SCL algorithm calculates L hard decision candidate vectors. If one of them has a valid CRC, the algorithm terminates with that output. If none of them has a valid CRC, the algorithm terminates with the most probable hard decision candidate vector.

2.6 Simulation Results

In this section, we present the software simulations of the SC, the SCL and the adaptive SCL decoding algorithms for $N = 1024$ and $K = 512$. Although fixed-point data types and function approximations reduces the complexity of implementation, they cause some performance degradation. This performance degradation is to be insignificant such that the trade-off between performance and complexity is kept. In this manner, we perform fixed-point and approximation simulations to illustrate the performance loss in FPGA implementation. For all simulations, the code is optimized for 0 dB by using Monte-Carlo code construction model with 10000000 trials [2]. The channel model is BAWGNC for all simulations.

Algorithm 3: Adaptive Successive Cancellation List Decoding Algorithm

Input: received codeword, y_1^N
Input: code block length, N
Input: information set, $\mathcal{A}$
Input: frozen bit vector, $u_{\mathcal{A}^c}$
Input: maximum list size, L
Output: estimated information bits, $\hat{u}_{\mathcal{A}}$
Variable: j `//valid CRC vector of SCD`
Variable: k `//valid CRC vector of SCLD`

```
1  begin
2      û_A ⟵ Successive Cancellation Decoding (y_1^N, N, A, u_{A^c})
3      j ⟵ Cyclic Redundancy Check Decoding (û_A)
4      if j is true then
5          return û_A
6      else
7          û_{L,A} ⟵ Successive Cancellation List Decoding (y_1^N, N, A,
           u_{A^c}, L)
8          for l ← 1 to L do
9              k ⟵ Cyclic Redundancy Check Decoding (û_{l,A})
10             if k is true then
11                 û_A ⟵ û_{l,A}
12                 return û_A
13         û_A ⟵ û_{1,A}
14         return û_A
```

2.6.1 Comparison between Floating-point and Fixed-point Simulations of the SC Decoder

In this section, we use P bit precision for both channel input and internal LLR values of the SC decoder. We use $P = 32$ for the floating-point simulations. The BER and FER performance results are shown in Figure 2.5 and 2.6 respectively. The performance difference between 32-bit floating-point precision, 6-bit and 5-bit fixed-point precision is insignificant. When 4-bit LLR precision is used, a noticeable performance degradation up to $1dB$ occurs. This performance degradation becomes significant when energy per bit to noise power spectral density ratio (Eb/No) increases.

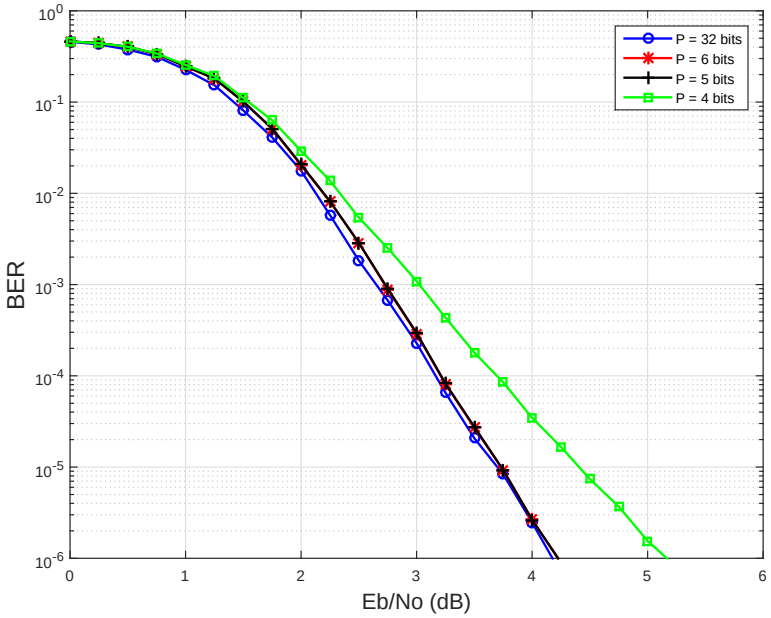


Figure 2.5: BER performance of the SC decoder for different bit precision (P), $N = 1024$, $K = 512$.

2.6.2 Performance Loss due to Min-sum Approximations in the SC Decoder

Although complexity of the f function in an SC decoder reduces by using the min-sum approximation in the Equation 2.25, the performance may also decreases.

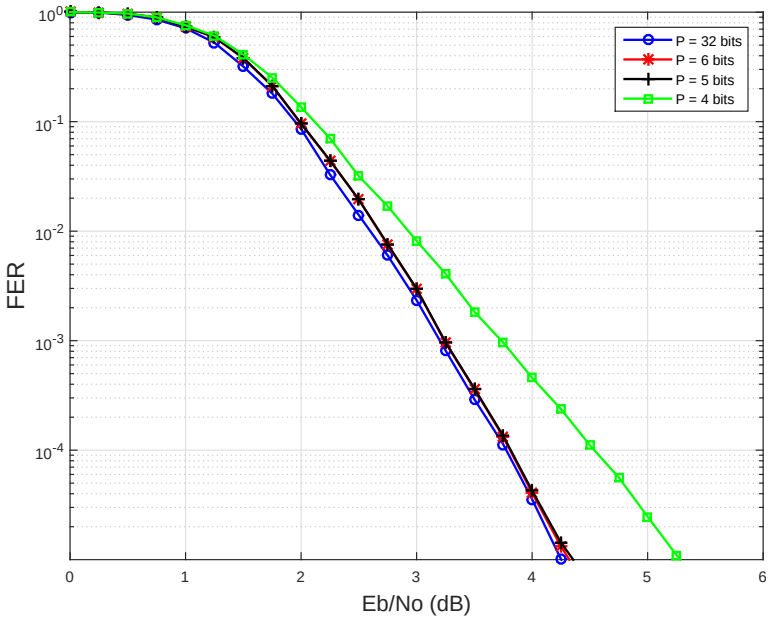


Figure 2.6: FER performance of the SC decoder for different bit precision (P), $N = 1024$, $K = 512$.

The BER performance loss due to min-sum approximation is shown in Figure 2.7. The FER performance loss due to min-sum approximation is shown in Figure 2.8. The results indicate that there is insignificant performance loss due to min-sum approximation of f function in SC decoder.

2.6.3 Fixed-point Simulations of the SCL Decoder

The fixed-point simulations with of the SCL decoder with the soft decision precision, $P = 6$ is shown in Figure 2.9 and 2.10. Note that the SCL decoder does not use CRC in this simulation. After 3 dB Eb/No, the performance improvement from $L = 2$ to $L = 32$ is not observable. However, there is still a performance gap between SC and SCL decoders.

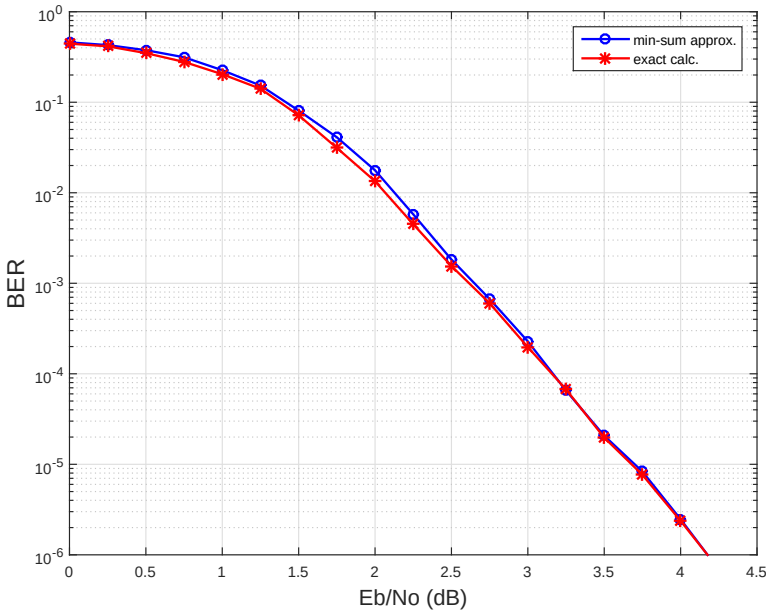


Figure 2.7: BER performance of the SC decoder due to approximations, $N = 1024$, $K = 512$.

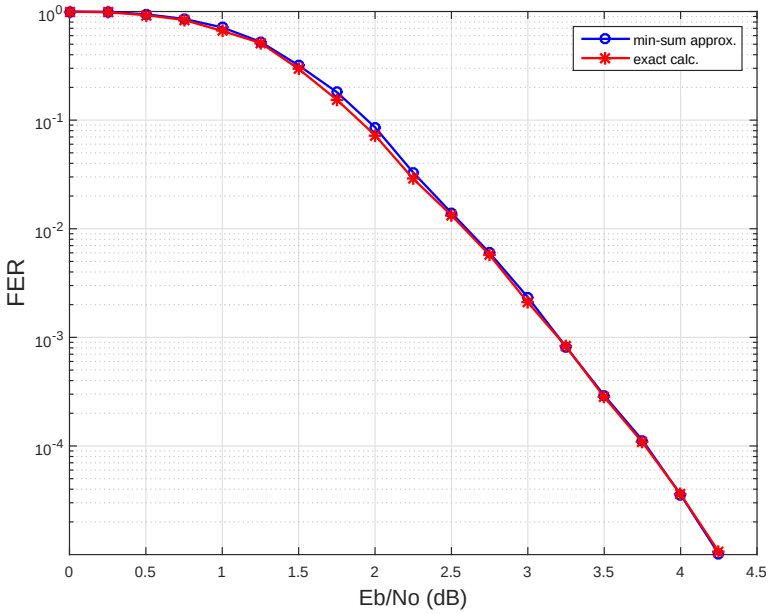


Figure 2.8: FER performance of the SC decoder due to approximations, $N = 1024$, $K = 512$.

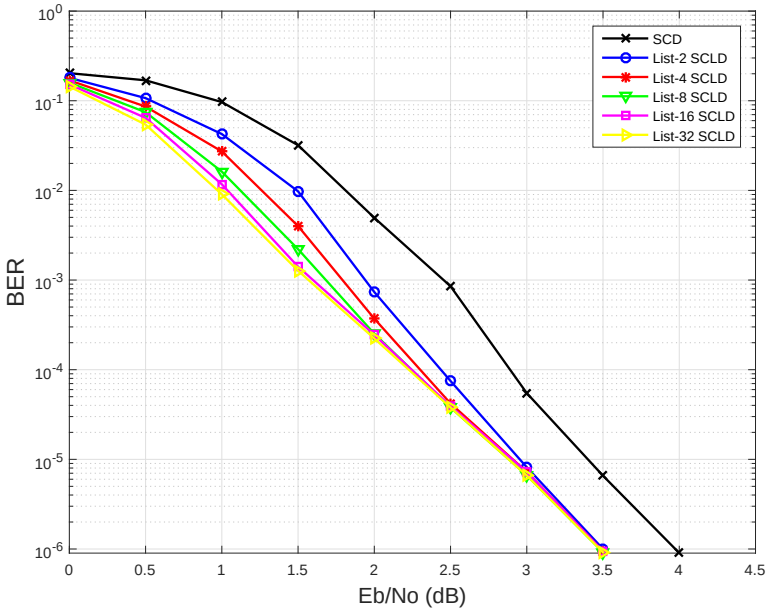


Figure 2.9: BER performance of the SC and the SCL decoders, $N = 1024$, $K = 512$, $P = 6$.

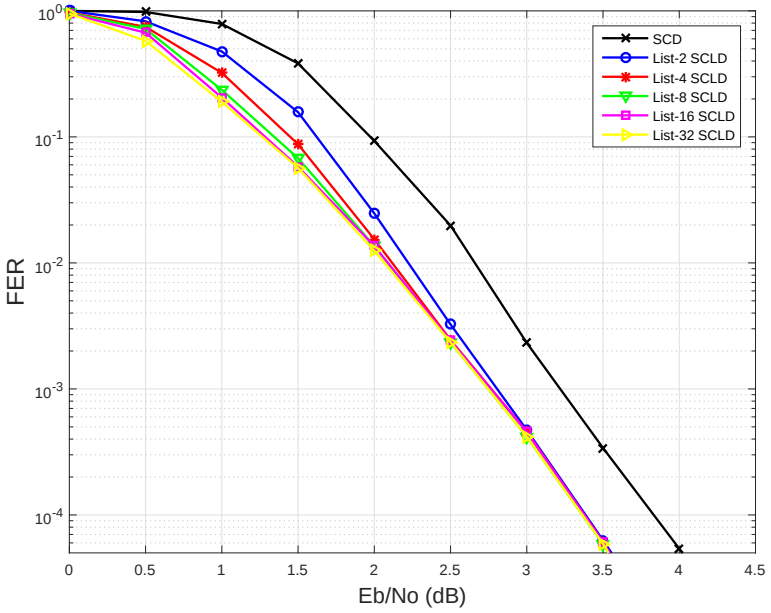


Figure 2.10: FER performance of eh SC and the SCL decoders, $N = 1024$, $K = 512$, $P = 6$.

2.6.4 Fixed-point Simulations of the Adaptive SCL Decoder

For adaptive SCL decoder, we made simulations to determine the input precision P_i of likelihood values. The BER and FER simulation results are shown in Figure 2.11 and 2.12 respectively. There is significant performance loss when the input of adaptive SCL decoder has $P_i = 3$ bits. When $P_i = 4$, there is up to 0.5 db performance loss due to inadequate input precision. For other input bit precisions ($P_i = 5$ and $P_i = 6$), we observed an insignificant performance loss.

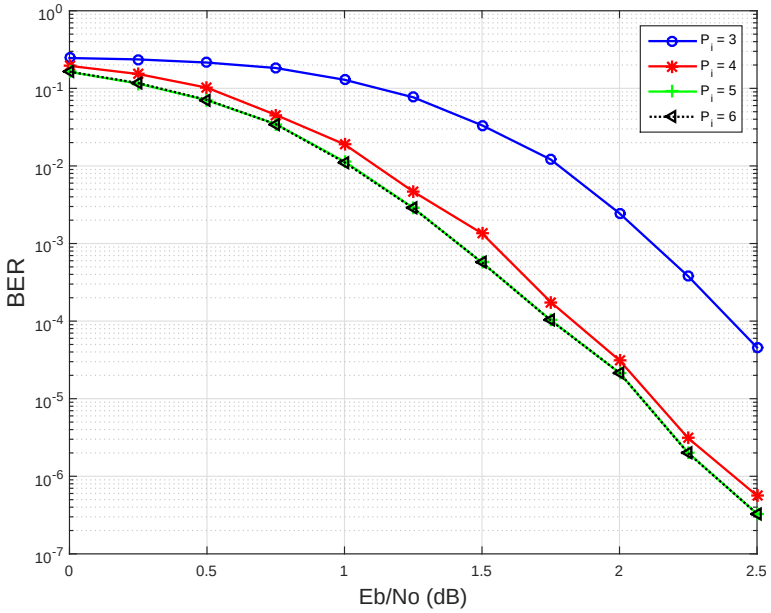


Figure 2.11: BER performance of Adaptive SCL decoder, $N = 1024$, $K = 512$, $L = 16$.

2.6.5 Systematic and Non-systematic Code Simulations of the SC Decoder

In this section, we present some simulation results of the SC decoding with systematic and non-systematic code. The BER performance of the code with $N = 1024$, $K = 512$ is shown in Figure 2.13. According to BER performance results, there is up to 0.5 dB performance gain of systematic codes with respect to

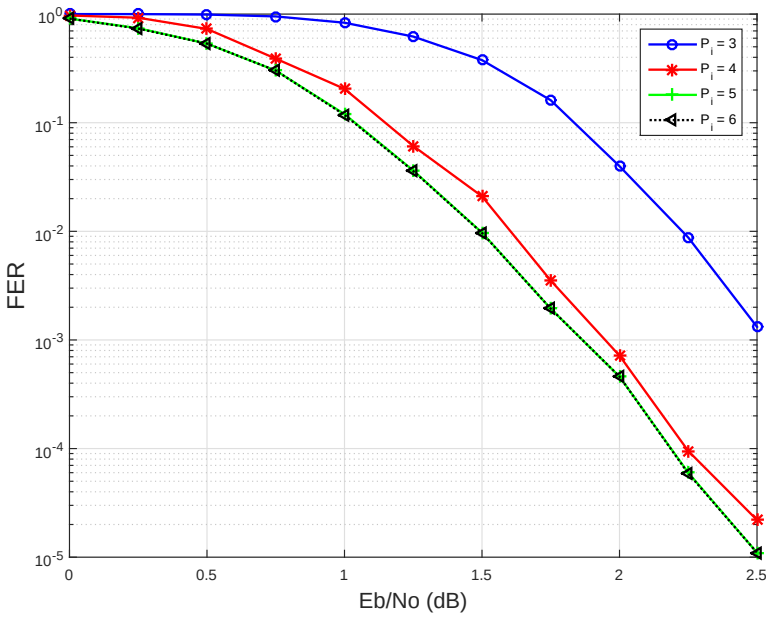


Figure 2.12: FER performance of Adaptive SCL decoder, $N = 1024$, $K = 512$, $L = 16$.

non-systematic polar codes under the SC decoding. The FER performance results is shown in Figure 2.14. We observed that the FER performance of systematic and non-systematic polar codes under SC decoding are almost identical.

2.7 Summary of the Chapter

In this chapter, we presented polar codes in terms of properties, construction, encoding and decoding methods. For encoding of polar codes, we presented two methods as non-systematic encoding and systematic encoding. For a non-systematic encoder, the input consists of free bits and the output consists of parity bits. For the systematic encoder, the input consists of free bits; in this case, the output consist of both free and parity bits. We showed that the systematic polar code has an improved BER performance than the non-systematic code under the SC decoding.

For decoding of polar codes, we presented the SC, the SCL and the adaptive SCL algorithms. We showed some fixed-point software simulation results to

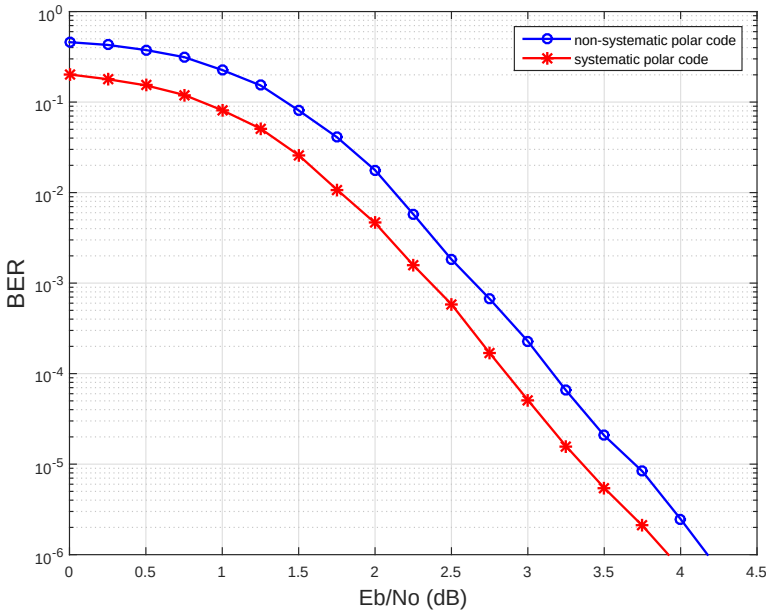


Figure 2.13: BER performance of SC decoder, $N = 1024$, $K = 512$.

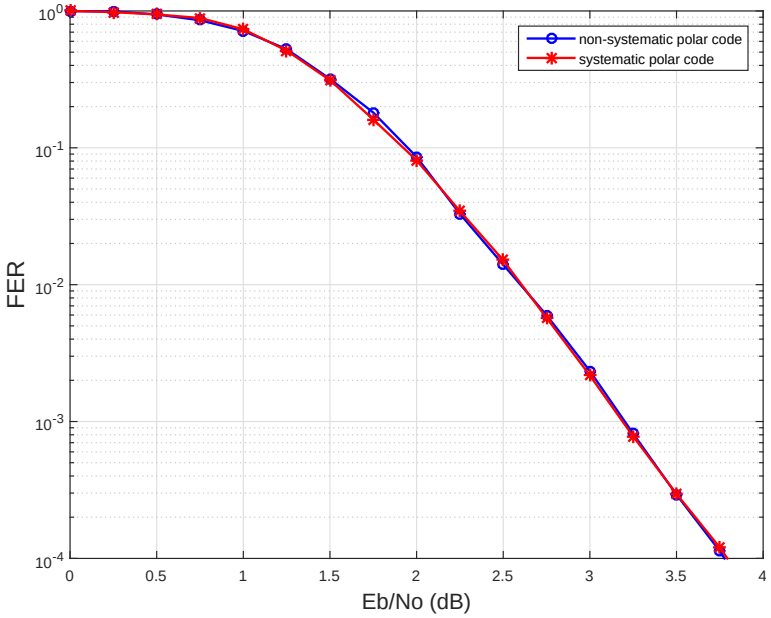


Figure 2.14: BER performance of SC decoder, $N = 1024$, $K = 512$.

demonstrate the performance loss due to approximations, input bit precisions of SC and adaptive SCL decoders. As a result, we observed an insignificant performance loss due to approximations and an input LLR bit precision more than 5 bits. For the adaptive SCL decoder, when the input log-likelihood (LL) bit precision is more than 5 bits, it does not cause an observable performance loss. According to these results, we will use systematic coding with $P = 6$ input LLR and LL bit precisions for our adaptive SCL decoder FPGA implementation, which we will present in the next chapter.

Chapter 3

An Adaptive Polar Successive Cancellation List Decoder Implementation on FPGA

In this chapter, we will present our adaptive decoder implementation. In Section 3.1, we will review previous studies about SC and SCL decoding in terms of algorithm and implementation. As mentioned earlier, the adaptive decoder consists of three decoders, SC, SCL and CRC decoders. In Section 3.2, we will present our SC decoder implementation with the detailed analysis of each submodule in its implementation. In Section 3.3, we will present our SCL decoder implementation and its analysis. In Section 3.4, we will present our CRC decoder with implementation results. Lastly, we will present the implementation results of our adaptive SCL decoder in Section 3.5.

3.1 Literature Survey

In this section, we review previous studies about implementation of SC and SCL decoders of polar codes.

3.1.1 Successive Cancellation Decoder Algorithms and Implementations

SC is a low-complexity decoding algorithm such that it uses soft decision information from channel to calculate hard decision estimations. SC has $\log N$ stage forward and backward processing structure. Internal soft decisions are calculated via forward processing, until a hard decision emerges at the decision stage, which is the last stage of forward processing. After that, backward processing starts and SC calculates partial sums by using hard decisions. The partial sums are used by forward processors as a feedback. SC performs forward and backward processing recursively, until all N hard decisions are estimated.

Implementation of the SC decoding is challenging to get high throughput with a low-complexity, because successive nature of the SC algorithm restricts parallel computations. To overcome this challenge, many different architectures are proposed. In this section, we review previous studies about implementation of the SC decoder in terms of complexity reduction and throughput improvement.

3.1.1.1 Architectures for Reducing Complexity

Implementation of the SC decoder on hardware is still an active research area after polar coding is invented in 2009 [2]. Initially, three different hardware architectures are proposed in [12], [15] and [16]. These are butterfly-based, pipeline tree and line architectures. Firstly, butterfly-based architecture is a resemble of well-known fast fourier transform (FFT) structure, which has $\log N$ stages with N processing operator at each stage. In implementation of polar codes, basic processing blocks are called processing elements (PEs), which performs f (2.25) and g (2.28) functions. Due to the calculation of hard decisions in a successive way, SC decoding introduces strict data dependencies compared to FFT structure. These data dependencies, caused by the forward processing of f and g functions for $N = 8$, are shown in Figure 3.1. In this figure, the circles represents PEs and inside these circles there are values such as $p_{i,j}$, where p is the function

type, i the is stage number and j is the element number in a stage. In general, each stage requires N computation blocks to calculate N hard decisions that takes $2N-2$ clock cycles (CCs). This is the conventional decoding cycle (DC) of the SC algorithm. Using $N \log N$ PEs is a primitive idea to maximize the computation speed, however data dependencies caused by successive nature of the decoder enables using less PEs without spending extra CCs. Since maximum $N/2$ PEs is activated for one CC, it is unnecessary to implement N PEs for each stage. At this point, pipeline tree architecture emerges.

Despite the rectangular shape of the butterfly-based architecture, the pipeline tree architecture has tree shape such that from stage $i = \{1 \rightarrow \log N\}$, at most $\frac{N}{2^i}$ computations can be executed in parallel due to strict data dependencies. Therefore, in this architecture, $\sum_{i=1}^{N} \frac{N}{2^i} = N-1$ PEs are allocated. In that way, a PE is capable of performing either f or g function at one CC. The number of PE can further be decreased to $N/2$ without the necessity of extra CCs. This approach is called the line architecture. With the expense of multiplexers, $\frac{N}{2^i}$ PE is selected from among $N/2$ PEs to increase the utilization for the i^{th} stage. Line architecture utilizes all PEs only for the first stage.

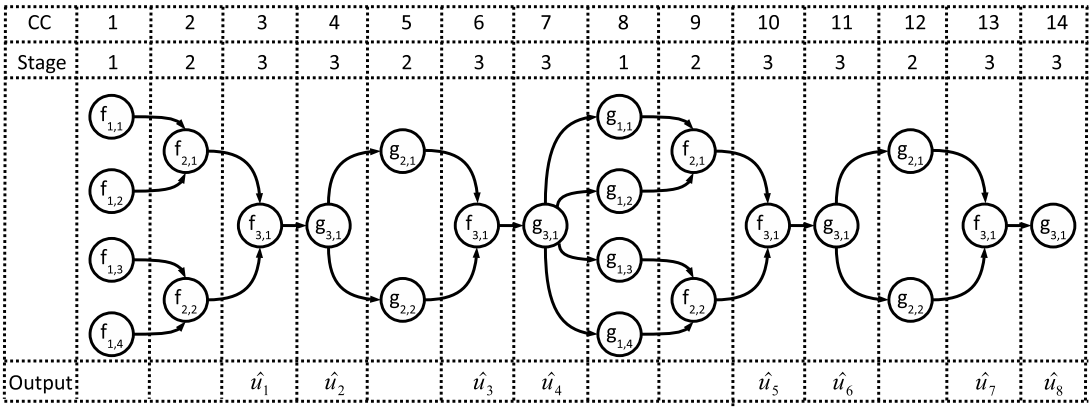


Figure 3.1: Data flow graph of forward processing for successive cancellation decoder, $N = 8$.

The utilization of PEs further increases with the expense of extra CCs in order to reduce complexity by using less PEs. The semi-parallel architecture, in [17], uses v PEs such that $1 < v < N/2$. The stages from 1 to $\log N - \log v - 1$ need

more than v PEs; the stage, $\log N - \log v$ needs v PEs and the remaining stages need less than v PEs to calculate internal soft decisions in one CC. Therefore, more than one CC is spent during the first $\log N - \log v - 1$ stages with the fully utilization of v PEs. Since the activation of these stages are less frequent than the other stages, the time expense is tolerable such that extra $\frac{N}{j} \log \frac{N}{4j} + 2$ CCs are necessary. Thus, the DC increases.

An another approach to reduce the complexity is the two phase decoder [18], which implements $\sqrt{N} - 1$ PEs like a $\sqrt{N}$ decoder with the tree architecture. In two-phase SC decoder architecture, the decoder is divided into two phases: phase-1 for the first $\frac{\log N}{2}$ stages and phase-2 for the remaining stages. Throughout decoding a code block, phase-1 and phase-2 stages are activated $\sqrt{N}$ times. Therefore, $\sqrt{N}$ hard decisions emerge at the end of a phase-2 stage. In total, additional $N + \sqrt{N}\frac{\log N}{2}$ CCs are necessary to calculate all hard decisions without using the 2-bit decoding method, which will be explained in the following section.

3.1.1.2 Architectures for Increasing Throughput

The conventional DC of a polar SC decoder reduces by half by precomputation of g functions [19]. In this approach, a PE performs both f and g functions at the same time. This provides 2-bit decoding [20] during a single CC. Using 2-bit decoding scheme without precomputation reduces the conventional DC by $N/2$, because only the last stage f and g functions are merged. An another approach to the problem is to interleave more than one codeword in the SC decoder presented in [21], [22]. This can address the low utilization problem of PEs by resource constrained implementation and reduces the DC by latency constrained implementation methods. In addition to these, SC decoder uses f and g functions to decompose a code segment into two simpler constituent segments recursively as a divide and conquer paradigm. The constituent segments, all frozen (rate-0) and all free (rate-1) can be decoded easily without further decomposition by using simplified successive cancellation (SSC) decoding [23], [22] and maximum likelihood SSC (ML-SSC) [24]. In addition to rate-0 and rate-1 codes, single parity check code (SPC) and repitition code (REP) code segments are also easy to

decode. At this point fast-SC (FSC) algorithm emerges [25]. In this algorithm, rate-0, rate-1, SPC and REP special code segments are detected and after detection, an ML decoder decodes these constituent code segments. Decomposition of code segments and detection of special code segments are shown in Figure 3.2.

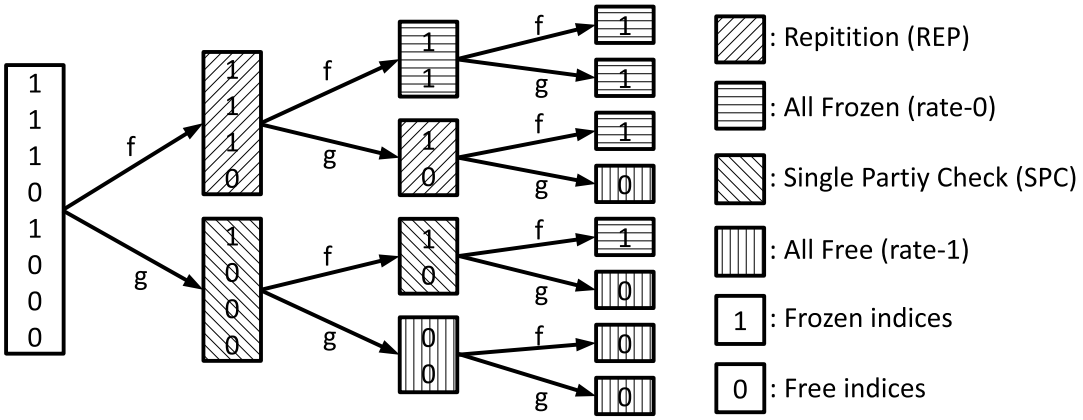


Figure 3.2: Decomposition of code segments and detection of special code segments, $N = 8$.

Code specific designs improve the throughput of SC decoder significantly. Unrolling the DC with sequential pipeline stages consist of f, g, rate-0, rate-1, SPC and REP nodes makes decoder capable of decoding multiple codewords in a reduced DC, presented in [26], [27]. An another method is the use of combinational logic is a way to combine all sequential stages of the SC decoder as one to increases energy efficiency and throughput with the expense of extra computational logic [28].

3.1.2 Successive Cancellation List Decoder Algorithms and Implementations

Hardware implementation of the SCL decoder is an attractive topic to increase the performance of polar codes with reasonable complexity, after the list decoding algorithm is proposed for polar codes in [4]. Combining list decoding algorithm with CRC makes polar codes competitive with Turbo and LDPC codes [4],

[29], [13]. Initial implementation of the SCL decoder focuses on maximizing the throughput with a fast radix sorting algorithm with a limited list size in [30], [31] and [32]. These implementations use log-likelihood LL representation of channel information to calculate hard decisions. A log-likelihood ratio LLR based list decoding implementation is presented to reduce the complexity in [33]. In addition to these, the conventional latency of a SCL decoder can be reduced by reduced latency list decoding (RLLD) algorithm such that it enables the detection of rate-0 and rate-1 constituent codes in [34]. An adaptive list decoding algorithm in software is presented to reduce the latency in [13]. In this algorithm, the decoder uses list-1 decoding at the beginning. If CRC is true as a result of list-i decoding, the algorithm terminates with the output of list-i decoder, otherwise the decoder is relaunched with the list-$2i$ decoder until the maximum list size is achieved.

Moreover, an other adaptive algorithm, which combines a simplified SCL decoder with a fast simplified SC decoder, is presented in [14] to enhance the throughput of SCL decoder in software. For higher list sizes, more efficient implementation of SCL decoder can be made by using a bitonic sorter in [35]. With a recent algorithm, SCL decoder can make multiple decisions [20], [36] in one CC. This can further increase the throughput of polar list decoders. Lastly, double thresholding method is proposed in [37] to decrease the list pruning latency.

3.2 Successive Cancellation Decoder Implementation

In this section, we present our implementation of the SC decoding algorithm. The high level description of the algorithm was shown in Section 2.5. The main modules of the SC decoder are processing unit (PU), decision unit (DU), partial sum update (PSU) and control logic (CL). The data flow between these modules is shown in Figure 3.3. PU is responsible for forward processing operation as computation of likelihood metric. The PU uses channel LLRs to compute internal LLRs. In addition to that the DU uses the internal LLRs of the PU to compute

a length-λ_i ($2 \leq \lambda_i < N$, $2 \leq i < N/2$) constituent of an internal hard decision vector, $\hat{u}$. This constituent hard decision vector is used by PSU to calculate partial sums. At the end of calculation of partial sums, SC decoder completes its one iteration as a part of $\hat{u}$ reveals. Unless all free bits $\hat{u}_{\mathcal{A}}$ reveals, SC decoder starts the its next iteration and the output of the PSU feedbacks to the PU. Lastly, the CL is responsible for all control signals to activate each module and regulate scheduling. Since the SC decoder uses systematic coding 2.4, the output of SC decoder consists of systematic information bits,$\hat{x}_{\mathcal{A}}$. Note that we set all frozen bits to zero, $u_{\mathcal{A}^c} = 0$ to obtain a decision rule.

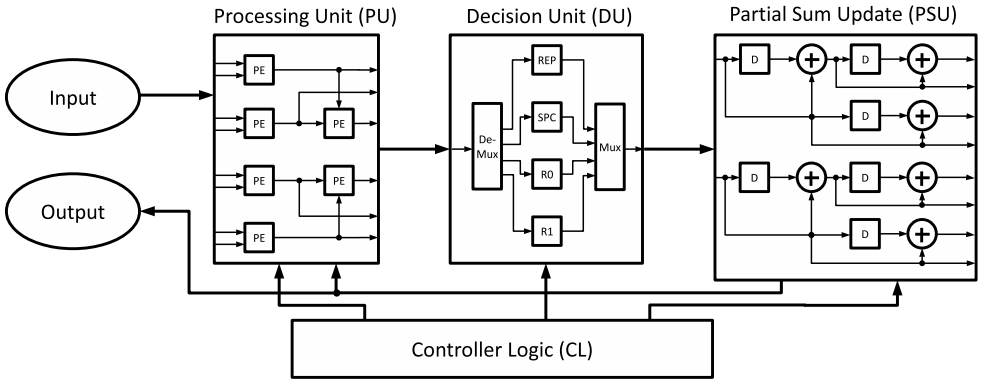


Figure 3.3: Data flow graph of successive cancellation decoder.

To enhance throughput, the SC decoder stores LLR and hard decision information in registers instead of block random access memory (BRAM), which provides slower data access less data width compared to register memory. In this way, processors can access data faster with higher parallelization. State information, free set information and scheduling control information are stored in BRAM, because control sequences do not need high parallelization. In the following sections, we will present the details of our PU, DU, PSU and CL implementation.

3.2.1 Processing Unit (PU)

The PU consists of PEs, which implement f (2.25) and g (2.28) functions. The aim of the PU is to complete at most $N \log N$ computations as fast as possible with the limited resources. The PU employs PEs as pipeline tree architecture

[16], without the PE and the decision element in the last stage of the architecture to provide minimum 2-bit hard decision in a DC. Therefore, the PUs has $\log N-1$ stages and $N-2$ PEs. Although the conventional DC of SC takes $2N-2$ CCs, our implementation takes variable CCs depending on the free set $\mathcal{A}$ and the frozen set $\mathcal{A}^c$. We analyzed $\mathcal{A}$ and all possible subsets of $\mathcal{A}$ to detect rate-0, rate-1, REP and SPC constituent code segments. When these code segments emerge, the PU terminates and gives internal LLRs to the DU.

In the following section, we will present our PE implementation.

3.2.1.1 Processing Element (PE)

PE implements f (2.25) and g (2.28) functions to decompose a length-λ code segment into two length-$\lambda/2$ code segments. It takes two input LLR values, γ_a and γ_b and one partial sum decision, $\hat{u}$. Each of the LLR value has P bit precisions and $\hat{u}$ has single bit precision. CL assigns the control signals, EN to enable the element and SEL to select the function type. As a result of a PE function, an intermediate LLR value γ_c with P bit precision is calculated. Input and output values of a processing element is shown in Figure 3.4 and the truth table of these values are shown in Figure 3.1.

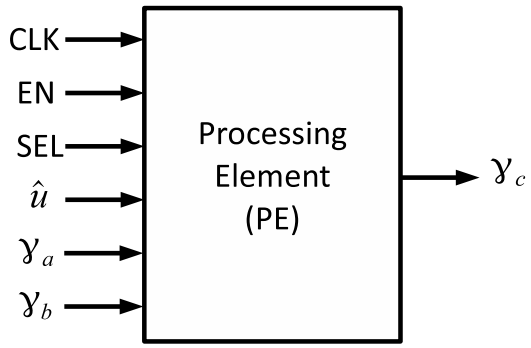


Figure 3.4: Inputs and outputs of a processing element (PE).

We have implemented a PE for both twos complement (TC) and sign-magnitude (SM) representations of γ values. Since the absolute value and the sign value is available in SM representation, f function is easier to compute. In

Table 3.1: The truth table of a PE.

CLK	EN	SEL	$\hat{u}$	γ_a	γ_b	γ_c
-	x	x	x	x	x	no change
↑	0	x	x	x	x	no change
↑	1	0	x	x	x	$\text{sign}(\gamma_a\gamma_b)\ \min(\lvert\gamma_a\rvert, \lvert\gamma_b\rvert)$
↑	1	1	0	x	x	$\gamma_b + \gamma_a$
↑	1	1	1	x	x	$\gamma_b - \gamma_a$

terms of g function, using TC logic minimizes resource usage due to addition and subtraction operations. At the end of a PE, pipeline registers are used for γ_c to meet with the timing requirements of the FPGA. Implementation results of a PE is shown in Figure 3.2. In these results, input values are also registered to measure the latency of the critical path delay from an input to an output. As a result, we choose SM representation to implement the SC decoder.

Table 3.2: Implementation results of a processing element.

LLR Bit Precision	PE Data Type	FFs	LUTs	Slices	Latency (ns)
4	TC	14	67	31	1.96
	SM	14	54	25	1.70
6	TC	20	98	47	2.51
	SM	23	120	48	2.44
8	TC	26	122	53	2.46
	SM	26	94	41	2.62
10	TC	34	145	68	2.85
	SM	32	101	45	3.02
12	TC	38	165	61	3.32
	SM	38	119	51	2.96
14	TC	46	190	73	3.54
	SM	44	139	63	3.08
16	TC	50	217	82	3.75
	SM	50	158	61	3.06

3.2.2 Decision Unit (DU)

The DU creates internal hard decisions from γ soft decisions for a constituent code length λ_i. The CL activates the DU at the end of PU, when a REP, a SPC,

a rate-0 or a rate-1 code segment emerges. We use a decision rule, similar to [25], to decode constituent codes such that the decision rule is shown in Equation 3.1. In this decision rule, r_λ represents the code rate of length-λ constituent code and $\hat{d}^{\lambda}_{1,i}$ is the length-λ internal hard decision vector. Note that this decision rule is only valid when all frozen bits are: $u_{\mathcal{A}^c} = 0$.

$$\hat{d}^{\lambda}_{1,i} = \begin{cases} 0, & \text{if } r_\lambda = 0 \\ 0, & \text{if } r_\lambda = \frac{1}{\lambda} \text{ and } \sum_{j=1}^{\lambda} \gamma_j \geq 0 \\ 1, & \text{if } r_\lambda = \frac{1}{\lambda} \text{ and } \sum_{j=1}^{\lambda} \gamma_j < 0 \\ \text{sign}(\gamma_i), & \text{if } r_\lambda = \frac{\lambda-1}{\lambda} \text{ and } \sum_{j=1}^{\lambda} \text{sign}(\gamma_j) = 0 \\ \text{not sign}(\gamma_i), & \text{if } r_\lambda = \frac{\lambda-1}{\lambda} \text{ and } \sum_{j=1}^{\lambda} \text{sign}(\gamma_j) \neq 0 \text{ and } i = \underset{k}{\text{argmin}}(|\gamma_k|) \\ \text{sign}(\gamma_i), & \text{if } r_\lambda = \frac{\lambda-1}{\lambda} \text{ and } \sum_{j=1}^{\lambda} \text{sign}(\gamma_j) \neq 0 \text{ and } i \neq \underset{k}{\text{argmin}}(|\gamma_k|) \\ \text{sign}(\gamma_i), & \text{otherwise } (r_\lambda = 1). \end{cases} \tag{3.1}$$

Since sign(γ) equals to the most significant bit (MSB) of a γ in SM representation, rate-1 decisions does not use any logic. In addition to that, rate-0 decisions are connected to the ground without using any logic. Thus, the critical path delay of DU is caused by either REP or SPC constitutes. The implementation of length-λ REP and SPC constitutes with $P = 6$, γ bit precision is shown in 3.3. Due to implementation results, we create a constraint for the maximum length of a REP and a SPC constituent as λ^M. As a default, we set $\lambda^M = 16$ to meet with the critical path delay, 5.25 ns. By using this constraint, the SC can decode at most length-16 REP and SPC code segments, however there is no limitation for the length of rate-0 and rate-1 code segments.

Table 3.3: Implementation results of REP and SPC constituent codes, $P = 6$.

Code Type	Code Length	FFs	LUTs	Slices	Latency (ns)
REP	2	13	51	22	1.47
	4	25	104	47	2.81
	8	61	219	98	3.61
	16	125	449	196	5.10
	32	273	921	428	6.45
	64	565	1861	817	7.69
SPC	2	13	40	21	1.07
	4	28	100	47	1.88
	8	56	230	104	3.53
	16	112	484	216	5.25
	32	224	993	496	7.99
	64	451	1992	902	8.95

3.2.3 Partial Sum Update (PSU)

The PSU is responsible for both calculation of the feedback hard decisions and final output of the SC decoder, $\hat{x}_{\mathcal{A}}$. This can be achieved by encoding length-λ_i internal decision vectors blocks, $\hat{d}$ such that λ_i is the length of the i^{th} constituent code. The encoding operation has $\log N$ stages and it is the bit-reversed version of the encoding algorithm that we present in Section 2.4. The last stage decision vector, $\hat{d}_{1,\log N}$ will be the systematic output of the SC decoder. In general, the j^{th} stage has $N/2^j$ decision vectors as $\hat{d}_{i,j}$ of length-2^j. The total number of $\hat{d}$ blocks loaded to PSU is defined as υ such that $\sum_{i=1}^{\upsilon} \lambda_i = N$.

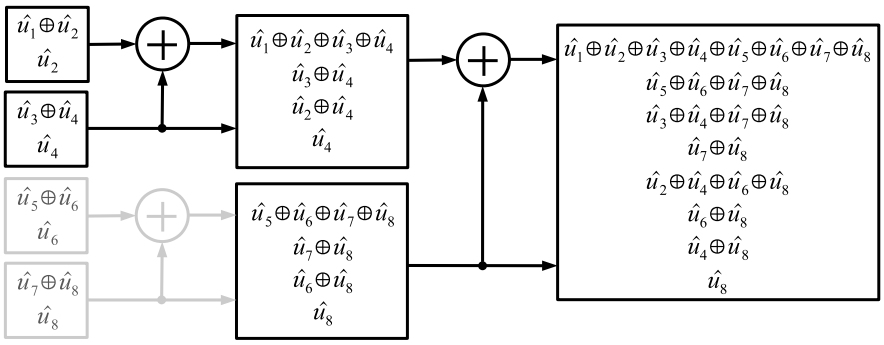


Figure 3.5: PSU with $N = 8$, $\upsilon = 3$, $\lambda_1 = 2$, $\lambda_2 = 2$ and $\lambda_3 = 4$.

The PSU performs logic operations with combinational logic and uses register arrays for keeping $\hat{d}$ values. For instance, let $N = 8$, $\upsilon = 3$, $\lambda_1 = 2$, $\lambda_2 = 2$ and $\lambda_3 = 4$, then the data flow of the PSU is shown in Figure 3.5. Since $\lambda_1 = 2$, the first input of the PSU is $\hat{d}_{1,1} = (\hat{u}_1 \oplus \hat{u}_2,\ \hat{u}_2)$. The PSU saves this input vector in a register array of length-2 and feedbacks these hard decisions to the PU. The second decision block of length-λ_2 is $\hat{d}_{2,1} = (\hat{u}_3 \oplus \hat{u}_4,\ \hat{u}_4)$. This vector and the registered vector are combined as $\hat{d}_{1,2} = (\hat{u}_1 \oplus \hat{u}_2 \oplus \hat{u}_3 \oplus \hat{u}_4,\ \hat{u}_3 \oplus \hat{u}_4,\ \hat{u}_2 \oplus \hat{u}_4,\ \hat{u}_4)$. Similar to the previous encoded hard decision block, this hard decision block is also given to the PU for forward processing operations. Since $\lambda_3 = 4$, so there is no need to calculate $\hat{d}_{3,1}$ and $\hat{d}_{4,1}$. The last decision block is $\hat{d}_{2,2} = (\hat{u}_5 \oplus \hat{u}_6 \oplus \hat{u}_7 \oplus \hat{u}_8,\ \hat{u}_7 \oplus \hat{u}_8,\ \hat{u}_6 \oplus \hat{u}_8,\ \hat{u}_8)$ in the second stage and the PSU combines $\hat{d}_{1,2}$ and $\hat{d}_{2,2}$ as $\hat{d}_{1,3}$ and the decoding is completed. Therefore, the general encoding rule of the PSU is

$$\hat{d}_{i,j} = (\hat{d}_{2i-1,j-1} \oplus \hat{d}_{2i,j-1},\ \hat{d}_{2i,j-1}) \tag{3.2}$$

$$= (\hat{d}^1_{2i-1,j-1} \oplus \hat{d}^1_{2i,j-1},\ \hat{d}^1_{2i,j-1},\ \ ...,\ \hat{d}^{2^{j-1}}_{2i-1,j-1} \oplus \hat{d}^{2^{j-1}}_{2i,j-1},\ \hat{d}^{2^{j-1}}_{2i,j-1}). \tag{3.3}$$

3.2.4 Controller Logic (CL)

CL is responsible for offline detection of REP, SPC, rate-0 and rate-1 constituent codes and scheduling of the SC decoder. Code detection functions are implemented in very high speed integrated circuit hardware description language (VHDL) to make a compact system design. As we present in Section 3.1.1, the conventional DC of the SC decoder is $2N-2$ CCs. Due to the detection of special constituent codes, CL reduces the number of decoding stages as shown in Figure 3.4. In this figure, K is the number of free bits and λ^M is the maximum number of REP and SPC constituent code lengths. The current stage number, i of f and g functions is illustrated as f_i and g_i and the length of the constituent code, j is set as rep_j for repetition, spc_j for single parity check, ai_j for rate-1 and af_j for rate-0 codes. When $K = 4$, the free set is:$\mathcal{A}_4 = \{4, 6, 7, 8\}$. Similarly, other free

sets are $\mathcal{A}_3 = \{4, 6, 8\}$ when $K = 3$ and $\mathcal{A}_7 = \{2, 3, 4, 5, 6, 7, 8\}$ when $K = 7$. If (K, λ^M) is (4,0), CL does not detect the constituent codes, so the conventional DC does not change. In other cases, CL is allowed to detect constituent codes and overall DC reduces. The CL sets the DC such that each stage of PU takes one CC, each iteration of DU takes one CC and PSU operates with combinational logic, which does not contribute to the DC.

Table 3.4: SC decoder latency for $N = 8$.

CC / K,λ^M	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4,0	f_1	f_2	f_3	g_3	g_2	f_3	g_3	g_1	f_2	f_3	g_3	g_2	f_3	g_3
4,2	f_1	f_2	af_2	g_2	rep_2	g_1	f_2	spc_2	g_2	ai_2	-	-	-	-
4,4	f_1	rep_4	g_1	spc_4	-	-	-	-	-	-	-	-	-	-
3,2	f_1	af_4	g_1	f_2	rep_2	g_2	ai_2	-	-	-	-	-	-	-
7,2	f_1	f_2	rep_2	g_2	ai_2	g_1	ai_4	-	-	-	-	-	-	-

3.3 Successive Cancellation List Decoder Implementation

In this section, we present our implementation of the SCL decoding algorithm, introduced in Section 2.5.2. The SCL decoding algorithm is a beam search algorithm that tracks L decoding paths together to increase the error correction performance of SC decoder with some additional complexity. Unlike the SC decoder, the SCL decoder uses the negative of the LL (δ) values as soft decision information such that

$$\begin{aligned} \delta_0 = W(y|x=0) &= -\ln\left(\frac{e^{\frac{-(y-1)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right) \\ \delta_1 = W(y|x=1) &= -\ln\left(\frac{e^{\frac{-(y+1)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right), \end{aligned} \tag{3.4}$$

where the channel is BAWGNC.

Both δ_0 and δ_1 can take values between 0 to $2^P - 1$, represented with P bit precision. Data flow of the SCL decoder is shown in Figure 3.6. Main modules of SCL decoder are list processing unit (LPU), list partial sum update (LPSU) and bitonic sorter (BS). We use an asymmetric BRAM to save δ_0 and δ_1 from channel at the beginning of decoding. After that, the processors use this BRAM to read and write δ_0 and δ_1 values for internal soft decision calculations. There are V LPUs in our SCL decoder implementation and each of LPU has one soft decision router element (SRE) and L list processing elements (LPEs). The aim of the SRE is to route the output of LL asymmetric BRAM to the input of LPEs with respect to the pointer information, which is stored in pointer register array memory. Each LPU takes $4LP$ bits as input and calculates $2LP$ bits of LL information in a pipeline manner. For the last $\log V$ stages, the output of the i^{th} LPU directly feedbacks to the input of the LPU, which has $\lceil i/2 \rceil$ index for $i = \{1 \rightarrow V\}$ without accessing the BRAM to enable pipeline calculations. A data buffer is implemented for the feedback operation of LPUs.

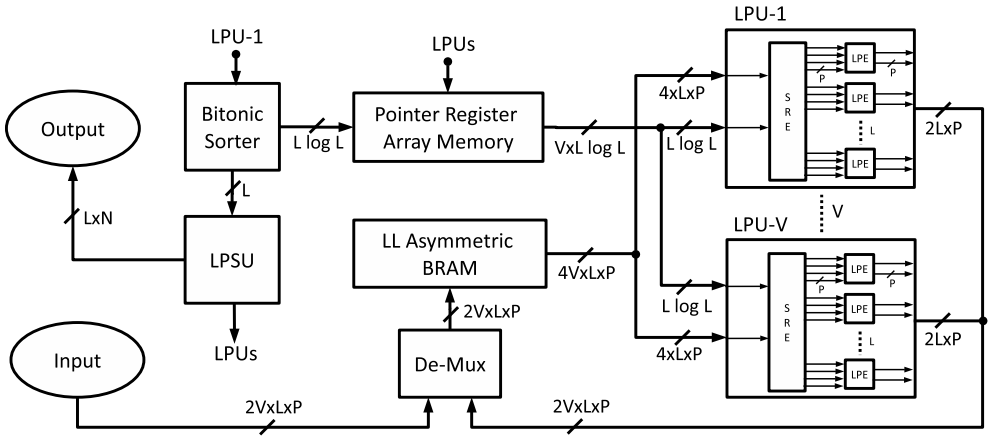


Figure 3.6: Data flow graph of successive cancellation list decoder.

For each valid decoding path, SCL decoder calculates path likelihood information such that the number of the path likelihood information doubles at each free decision step, until $2L$ different valid paths emerge. At this point, the decoder has not adequate resources to track all $2L$ paths. Therefore, it sorts them to find L best paths. If more than one winner path is reproduced from the same ancestor, two different processors have to access to the same memory and one of these processors writes to the same memory. This makes memory conflicts

due to overwriting. To solve this problem, the primitive idea is to copy all data history from one memory of a winner path to the memory of a loser path. The cost of copying information is quite significant, because the amount of data to be copied is $2LP(N-1)$ bits for likelihood values and LN bits for the hard decisions. For the implementation of the SCL decoder, we use P bits LL representation to calculate internal soft decisions. Therefore, $2P$ bits are necessary to keep both likelihood values δ_0 for the likelihood of 0 and δ_1 for the likelihood of 1. In addition to that, we use $N-1$ memory locations to save $2P$ likelihood values for each list to enable internal calculations. During state copying, the SCL decoder pauses the all processing operations such that it creates additional latency after each sorting operation. A better solution is to create a pointer memory, which remembers the valid memory locations for all lists. We use this approach and create $L \log N \times \log L$ bits of pointer array memory. The pointer memory has $L \log N$ depth, because each list needs a pointer for $\log N$ stages. There is L different memory options to access, thus the data width of the pointer memory is $\log L$.

We will explain the implementation details of LPU, LPSU and BS throughout this section.

3.3.1 List Processing Unit (LPU)

The LPU consists of L LPEs and one SRE to compute δ_0 and δ_1 soft decisions. We used semi-parallel architecture for processing to minimize the total complexity of all LPU blocks. Therefore, the SCL decoder has V LPUs. Each LPU reads from and writes to the asymmetric BRAM, which has $2V \times L \times P$ input data width and $4V \times L \times P$ output data width and $(2 * N/V) - 2$ address length. The LL bit precision, P is taken as $P = \log N + Q$, where Q is the channel LL precision, to avoid overflows during forward processing of $\log N$ stages. A LPE can read one of L different memory locations of BRAM or the data buffer between the LPUs, however it can write only its own memory location. During the activation of LPSUs and BS, all LPUs are paused. In case of a loser list after sorting, the SRE

does not route the LL information of that list to LPEs to perform g functions.

SRE reads pointer values from pointer register array and route both of output of the BRAM and data buffer to the input of LPEs. The pointer information is only necessary for g functions, because f functions always operate in the same memory. The synthesis results of LPU, SRE and LPE is shown in Figure 3.5. As L increases, SRE uses more lookup tables (LUTs) and dominates the LPU implementation.

Table 3.5: Synthesis of LPU, SRE and LPE.

		LPU		LPE		SRE	
LL Precision	**List Size**	**LUTs**	**Latency (ns)**	**LUTs (%)**	**Latency (%)**	**LUTs (%)**	**Latency (%)**
8	2	144	3.52	55.6	71.3	44.4	28.7
	4	360	3.98	64.4	71.7	35.6	28.3
	8	976	4.16	47.5	68.6	52.5	31.4
	16	2976	4.30	31.2	66.3	68.8	33.7
	32	12096	4.76	15.3	59.9	84.7	40.1
16	2	312	3.66	59.0	72.3	41.0	27.7
	4	760	4.11	66.3	72.5	33.7	27.5
	8	2032	4.30	49.6	69.3	50.4	30.7
	16	6112	4.46	33.0	66.9	67.0	33.1
	32	24512	4.94	16.4	60.4	83.6	39.6

The RTL schematic of LPU for $N = 1024$, $P = 14$ and $L = 4$ is shown in 3.7.

3.3.1.1 List Processing Element (LPE)

LPEs compute f_0, f_1, g_0 and g_1 functions to obtain $\log N$ stage LL values. At each stage, these functions are activated $N/2$ times. The f_0 and f_1 functions are LL representation of the f function; g_0 and g_1 are the LL representation of the g function such that

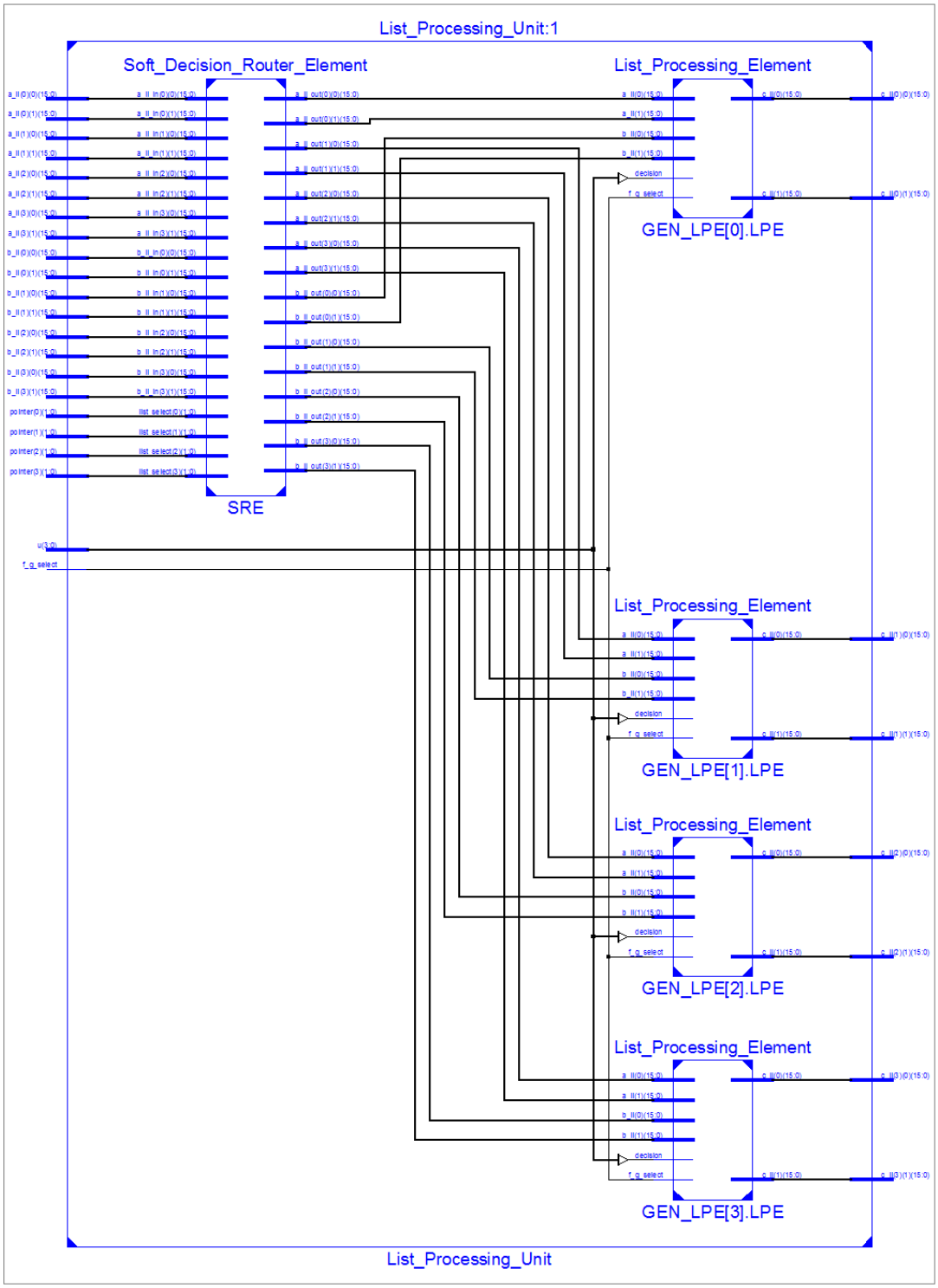


Figure 3.7: The RTL schematic of list processing unit for $N = 1024$, $P = 16$ and $L = 4$.

$$
\begin{aligned}
f_0(\delta_{a,0}, \delta_{a,1}, \delta_{b,0}, \delta_{b,1}) &= \delta_{c,0} \\
&= \min(\delta_{a,0} + \delta_{b,0},\ \delta_{a,1} + \delta_{b,1}) + \ln(1 + e^{-|\delta_{a,0}+\delta_{b,0}-\delta_{a,1}-\delta_{b,1}|}) - \ln 2 \\
&\approx \min(\delta_{a,0} + \delta_{b,0},\ \delta_{a,1} + \delta_{b,1}),
\end{aligned} \tag{3.5}
$$

$$
\begin{aligned}
f_1(\delta_{a,0}, \delta_{a,1}, \delta_{b,0}, \delta_{b,1}) &= \delta_{c,1} \\
&= \min(\delta_{a,0} + \delta_{b,1},\ \delta_{a,1} + \delta_{b,0}) + \ln(1 + e^{-|\delta_{a,0}+\delta_{b,1}-\delta_{a,1}-\delta_{b,0}|}) - \ln 2 \\
&\approx \min(\delta_{a,0} + \delta_{b,1},\ \delta_{a,1} + \delta_{b,0}),
\end{aligned} \tag{3.6}
$$

$$
\begin{aligned}
g_0(\delta^*_{a,0}, \delta^*_{a,1}, \delta^*_{b,0}, \delta^*_{b,1}, \hat{u}) &= \delta_{d,0} \\
&= \delta_{a,0} + \delta_{b,0} + (\delta_{a,1} - \delta_{a,0})\ \hat{u} - \ln 2 \\
&\approx \delta_{a,0} + \delta_{b,0} + (\delta_{a,1} - \delta_{a,0})\hat{u},
\end{aligned} \tag{3.7}
$$

$$
\begin{aligned}
g_1(\delta^*_{a,0}, \delta^*_{a,1}, \delta^*_{b,0}, \delta^*_{b,1}, \hat{u}) &= \delta_{d,1} \\
&= \delta_{a,1} + \delta_{b,1} + (\delta_{a,0} - \delta_{a,1})\ \hat{u} - \ln 2 \\
&\approx \delta_{a,1} + \delta_{b,1} + (\delta_{a,0} - \delta_{a,1})\hat{u},
\end{aligned} \tag{3.8}
$$

where $\delta^*_{a,0}, \delta^*_{a,1}, \delta^*_{b,0}, \delta^*_{b,1}$ are the pointed LL vectors.

SCL decoder performs g_0 and g_1 functions after $\hat{u}$ hard decisions are made.

3.3.1.2 Soft Decision Router Element (SRE)

The SRE reads the pointer information from pointer register array. With respect to this information, the SRE routes the output of the asymmetric LL BRAM to L LPEs. This routing operation has $P\ L^2$ complexity, because a SRE needs $4PL$ $L - to - 1$ demultiplexers to map $\{\delta_{a,0}, \delta_{a,1}, \delta_{b,0}, \delta_{b,1}\}$ LLs to $\{\delta^*_{a,0}, \delta^*_{a,1}, \delta^*_{b,0}, \delta^*_{b,1}\}$ pointed LLs for all lists.

3.3.2 List Partial Sum Update Logic (LPSU)

The LPSU updates hard decision partial sums for all lists as a feedback to g_0 and g_1 functions and creates the systematic data output of the SCL decoder. The feedback decisions have $L(N-1)$ bits and the systematic output has LN bits. The LPSU uses registers as memory instead of BRAM to decrease latency. This enables to calculate all necessary partial sums in one CC. The LPSU accesses registers through $N-1$ hard decision router elements (HREs) with respect to pointer information to avoid recalculation of previous hard decisions. The data flow of the LPSU for $N=4$, $L=2$ is shown in Figure 3.8. At each decision step from $i=\{1\leq i\leq N\}$, L hard decisions appear for the l^{th} list $l=\{1\leq l\leq L\}$ as $\hat{u}_{l,i}$. If the outputs of l^{th} list has lower probability than the other lists, we do not need the decisions of this list anymore. In addition to that, a list can operate with the previous hard decisions of an other list. For that operation, the LPSU accesses the decision memory by using de-multiplexers and calculates the partial sums.

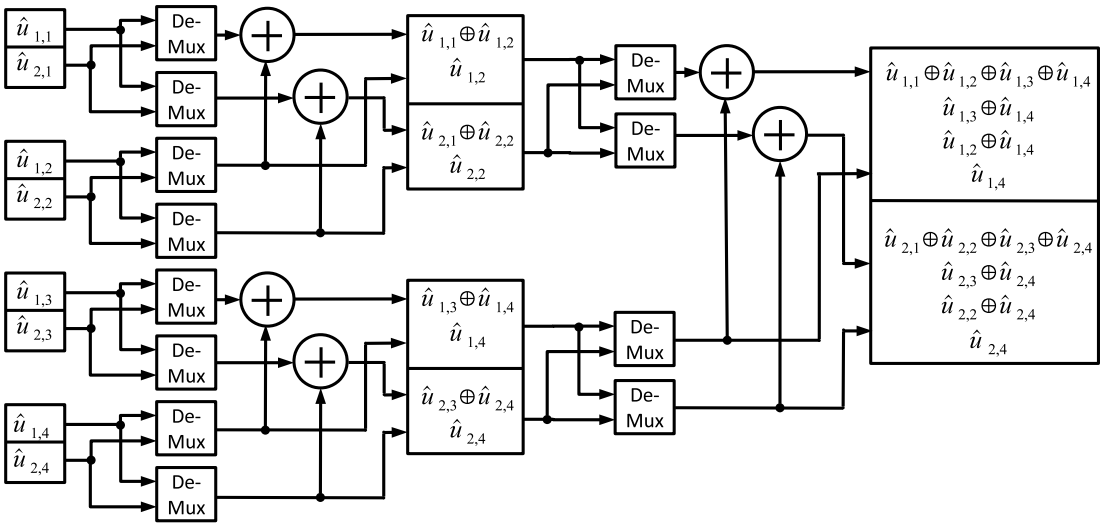


Figure 3.8: List partial sum update logic (LPSU) for $N=4$, $L=2$.

3.3.2.1 Hard Decision Router Element (HRE)

The operation of the HREs and SREs in Section 3.3.1.2 is similar as they access to the same pointer memory at different times. The only difference is the HRE takes hard decisions as input. Therefore, the complexity of each HRE decreases to L^2 from $P\ L^2$ such that each HRE has $L\ L-to-1$ demultiplexers. Accessing to the pointer memory does not cause memory conflicts, because read operations occur before write operations.

3.3.3 Sorter

Our SCL decoder needs a sorter to find best L decoding path among $2L$ at each free decision stage. That means, we activate the sorter module K times for a code block. During sorting, the decoder core waits for the best decoding paths without performing any operations. Therefore, latency of a sorter module has an important influence on total latency of the SCL decoder. In addition to that, the probability of each decoding path is represented by two LLs with each of them has P bit precision. Although these $2L$ LLs need to be sorted to find the highest L LLs, we only need L winner indices and their list number. The remaining Lloser paths are discarded. The output information of sorter is used for making hard decisions and updating pointer list values.

3.3.3.1 Bitonic Sorter

To limit latency and also resource usage, we implemented BS algorithm [38]. The BS algorithm is a low complexity semi-parallel sorting algorithm, which is suitable for FPGA implementation of polar list decoders [35]. The complexity of bitonic sorting scales with L efficiently.

Let $\Upsilon_B(L)$ denote the time complexity of the bitonic sorter with the list size

L such that

$$\Upsilon_B(L) = \Upsilon_B(\frac{L}{2}) + \log L + 1 \tag{3.9}$$

$$= \Upsilon_B(\frac{L}{2}) + \Theta(\log L) \tag{3.10}$$

$$\overset{\text{(i)}}{=} \mathcal{O}(\log^2 L). \tag{3.11}$$

Let $\zeta_B(L)$ denote the space complexity of the bitonic sorter with the list size L such that

$$\zeta_B(L)) = 2\zeta_B(\frac{L}{2}) + 2L(\log L + 1) \tag{3.12}$$

$$= 2\zeta_B(\frac{L}{2}) + \Theta(L \log L) \tag{3.13}$$

$$\overset{\text{(i)}}{=} \mathcal{O}(L\log^2 L), \tag{3.14}$$

where $\Upsilon_B(1) = 1$, $\zeta_B(1) = 2$ and (i): the master theorem, case 2.

For instance, let $L = 4$, the bitonic sorter circuit is shown in Figure 3.9. There are three stages as s_1, s_2, s_3 and these stages has substages such as $s_{1,1}, s_{2,2}, s_{3,2}$. Then, total stage number is $\log(L) + 1$ and total substage number is $\frac{(\log(L)+1)(\log(L)+2)}{2}$. Each list processing core provides two LLs to the BS. In addition to that, the input indices are represented by $\log 2L$ bit precision from one to eight such that $LL_{1,1}$ is the first, $LL_{1,2}$ is the second and $LL_{4,2}$ is the eighth input indices. These indices follow the LL data, throughout the sorter to keep list history information. The elements in each substage are simple comparators, which compares the left upper LL value with the left bottom one. The higher LL value is the output of diamond connection and the lower LL value is the output of the square connection with their initial index information. Comparators does not use initial index information, but they use LL values for comparison.

We have implemented three different bitonic sorters as BS, fast bitonic sorter (FBS) and fast reduced bitonic sorter (FRBS). For the BS implementation, we use pipeline registers at the end of each substage. For the FBS implementation, we use pipeline registers for every two substage of the bitonic sorter. For the

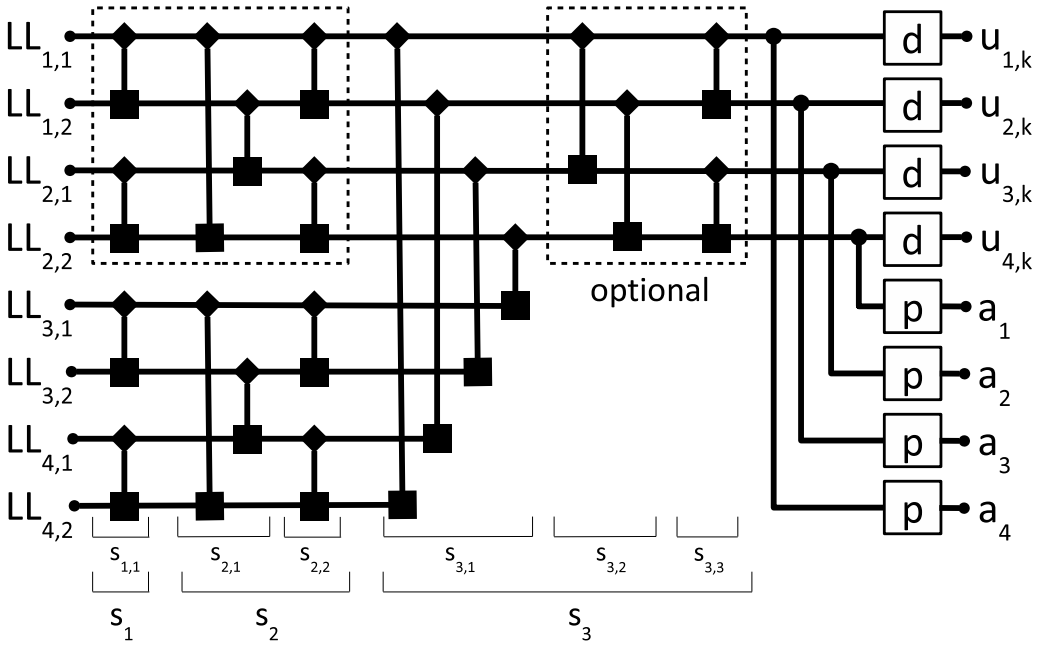


Figure 3.9: Bitonic sorter circuit for $L = 4$.

FRBS implementation, the last $\log L$ optional substages, $s_{3,2}$ and $s_{3,3}$ are not used. Thus, the output list information is not sorted from the best list to the worse list. This may introduce some performance degradation to the algorithm as we will be discussed in Section 3.5.

At the end of s_3, there are four identical decision units (d) to make hard decisions with respect to indices of sorted LLs. Since we activate the sorter $K - \log L$ times as $k = \{1 \rightarrow K - \log L\}$ for all lists $l = \{1 \rightarrow L = 4\}$, the output hard decisions are saved as $u_{l,k}$. Let the input of a decision unit is t, then the decision rule is

$$\hat{u}_{l,k} = \begin{cases} 0, & \text{if } t \bmod 2 = 1 \\ 1, & \text{otherwise.} \end{cases} \tag{3.15}$$

In addition to that, there are four pointer units (p) to calculate the list number of output indices. Let the input of a pointer unit is l and the output is a such that $a = \lceil \frac{l}{2} \rceil$. The output of a pointer unit determines a routing rule for a LPU. Similar to the example of $L = 4$ sorter, the $L = 2$ sorter is shown as the dotted square box in the upper left corner of the $L = 4$ bitonic sorter. A recursive

algorithm is used to generate a sorter with respect to L.

The resource usage of the bitonic sorter is shown in Table 3.6. Since FRBS has advantages in terms of both resource usage and latency, we choose FRBS to implement the adaptive SCL decoder. RTL schematic of the FBS for $L = 2$ is shown in Figure 3.10. For the FRBS implementation, the last stage bitonic comparator in this RTL schematic is not used.

Table 3.6: Implementation results of the bitonic sorter for $P = 8$.

List Size	Sorter Type	FFs	LUTs	Slices	Period (ns)	Latency (CC)	Latency (ns)
2	FRBS	34	147	71	0.51	1	0.51
	FBS	54	229	108	2.95	1	2.95
	BS	96	189	84	1.68	3	5.03
4	FRBS	146	507	209	4.21	1	4.21
	FBS	203	873	345	3.24	2	6.47
	BS	401	635	261	2.03	6	12.20
8	FRBS	500	1521	624	4.50	3	13.49
	FBS	776	2492	970	3.43	4	13.71
	BS	1446	2054	835	2.13	10	21.33
16	FRBS	1781	4796	2019	4.44	5	22.22
	FBS	2469	6660	2824	3.75	7	26.25
	BS	4694	6174	2521	2.24	15	33.54
32	FRBS	5893	16170	6070	5.09	7	35.61
	FBS	7110	21406	8200	4.43	10	44.25
	BS	14151	17209	6763	2.42	21	50.88

3.4 CRC Decoder Implementation

The CRC decoder is activated at the end of both SC and SCL decoders for detecting whether a decision vector is valid. We use CRC-16-CCITT, $x^{16} + x^{12} + x^5 + 1$ polynomial for the implementation. The input of the CRC decoder is $\hat{x}_{\mathcal{A}}$ at the end of SC decoder and $\hat{x}_{l,\mathcal{A}}$ at the end of SCL decoder for $l = \{1 \rightarrow L\}$. The output of the CRC decoder is boolean such that it is equal to '1' if the CRC is valid, '0' otherwise. The CRC decoder is shown in Figure 3.11. In this figure, there are 16 pipeline registers and 3 XOR gates. At the initial stage, the output

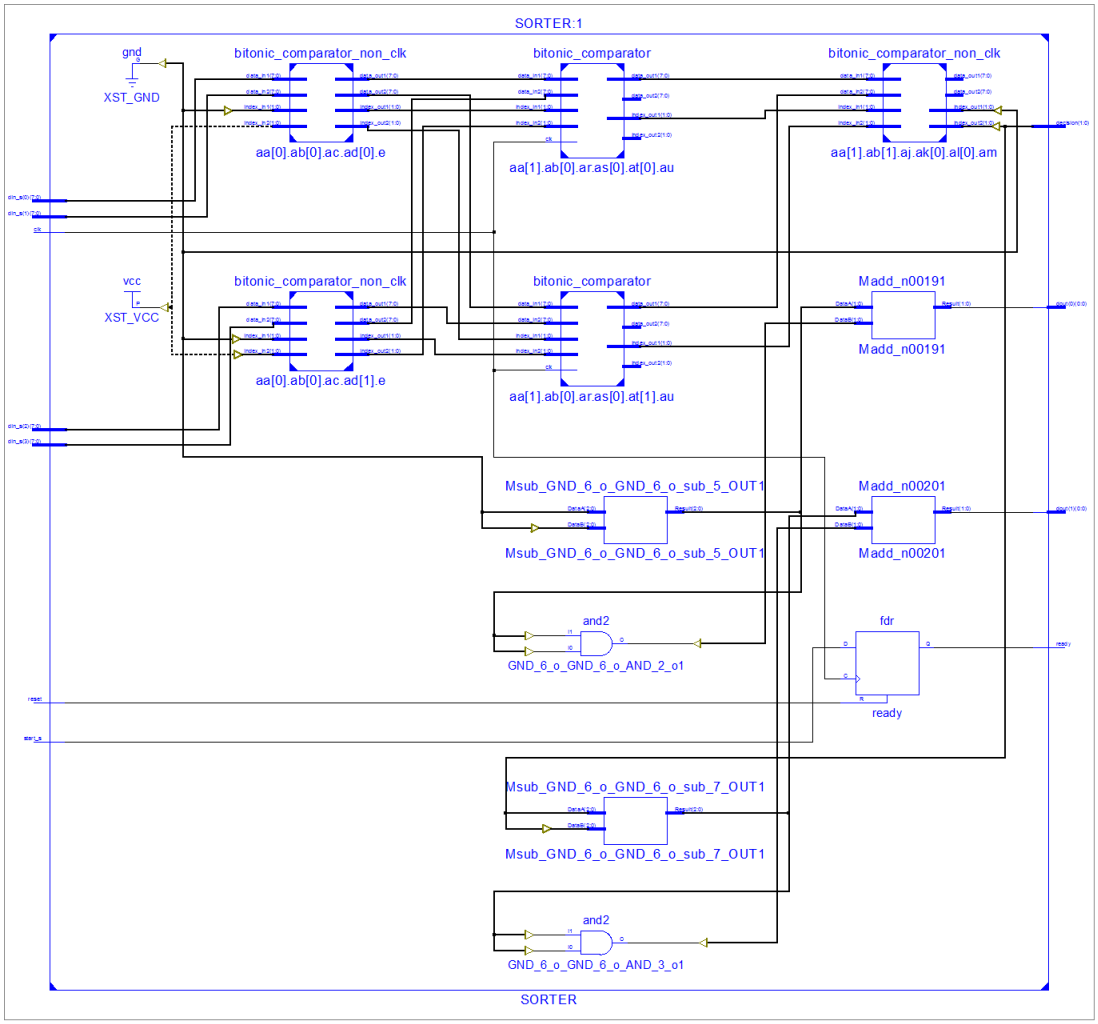


Figure 3.10: RTL schematic of the fast bitonic sorter with $L = 2$.

of all registers are zero. After that, an input vector is loaded to *din* pin from least significant bit (LSB) to MSB, where the last 16 MSB bits are the CRC bits. Loading an input vector to the CRC decoder takes K CCs. The final CRC output is valid, if and only if the output of all 16 flip-flops (FFs) are zero after the K^{th} CC.

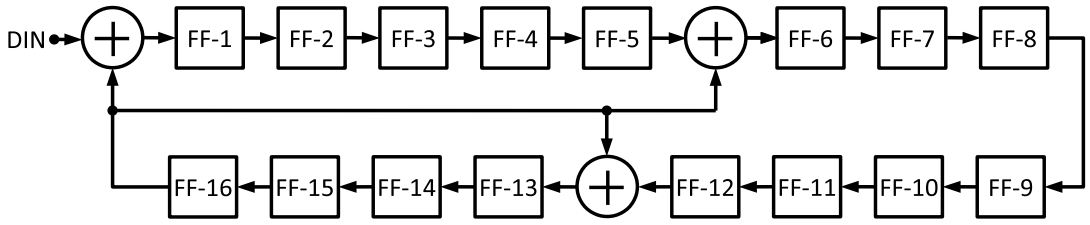


Figure 3.11: CRC decoder circuit.

In our implementation, we reduced the latency of the CRC decoder from K to C by using C pipeline stages. C can take any integer value, if it is dividable by K. The implementation results of CRC decoder for $K = 512$ with respect to various C values are shown in Table 3.7. According to implementation results, we choose C as two to limit the maximum period as 2.56 ns.

In case of CRC encoding, the last 16 bits set to zero. After all input is loaded, the output of 16 FFs will be the CRC vector. Therefore, the same circuit can be used for both CRC encoding and decoding.

Table 3.7: Implementation results of CRC decoder for $K = 512$.

FFs	LUTs	Slices	Period (ns)	Latency (CC)	Latency (ns)
17	2711	1113	18.11	1	18.11
34	1572	713	2.56	2	5.13
67	1526	617	2.20	4	8.81
132	1501	619	1.85	8	14.81
261	1571	658	1.73	16	27.68

The RTL schematic of CRC for $K = 512$ is shown in Figure 3.12.

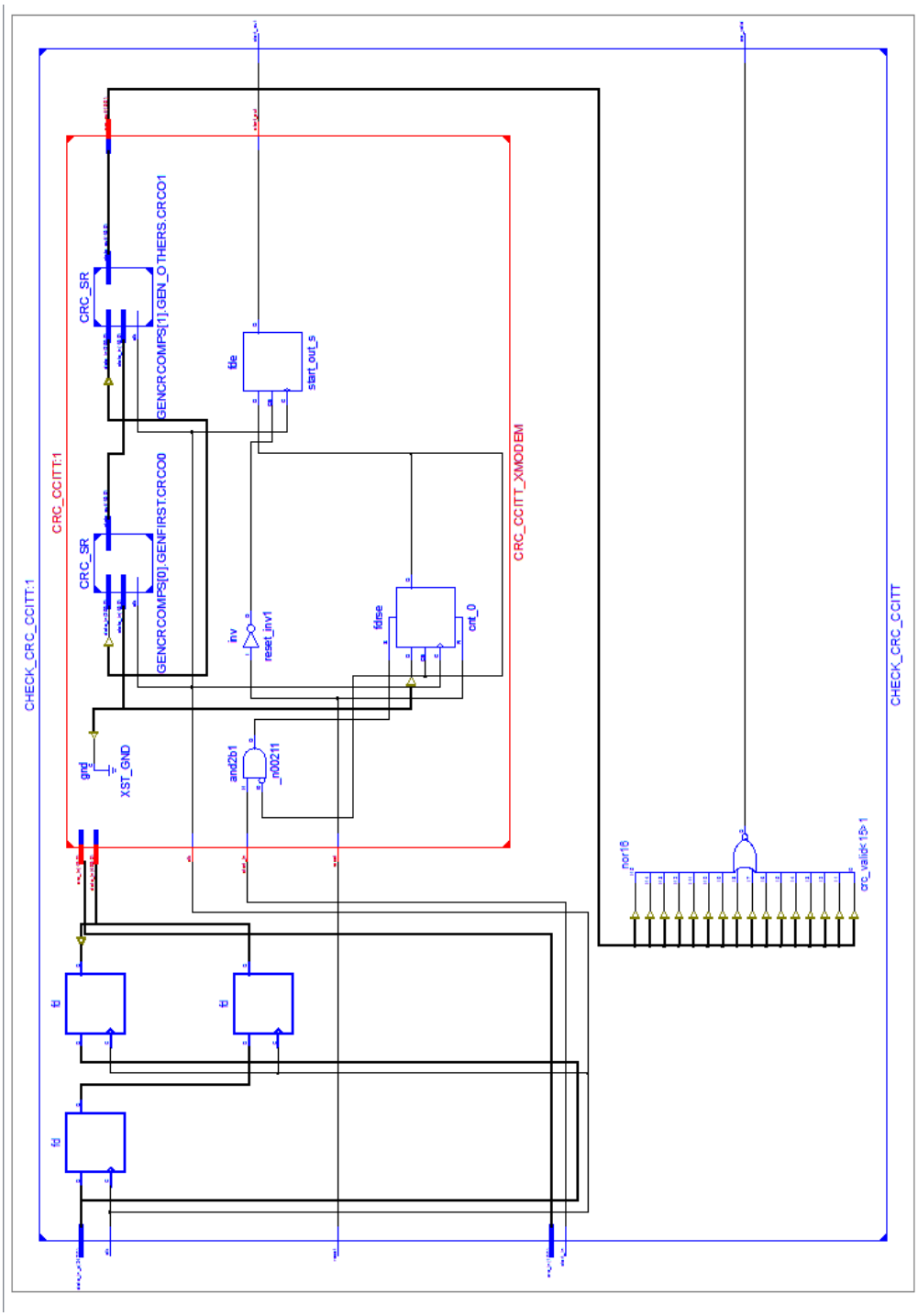


Figure 3.12: RTL schematic of the CRC for $K = 512$ with two CCs latency.

3.5 Adaptive SCL Decoder Implementation Results

We implement the adaptive decoder as the combination of SC, SCL and CRC decoders on the Xilinx Kintex-7 (xc7k325t-2ffg900c) FPGA. The resource usage, latency and throughput of the adaptive decoder is shown in Table 3.8. For this implementation, we set input LLR precision of SC decoder to $P_{SCD} = 6$ bits. Although the input LL precision of the SCL decoder is $P_i = 6$ bits, we add extra $\log N$ bits to ensure that the decoder does not overflow. Thus, the internal LL bit precision of the SCL decoder is $P_{SCLD} = 6 + \log N$ bits. The number of LPU set as $V = 4$ and we use FRBS to sort LL values in the SCL decoder. The adaptive SCL decoder performs its maximum latency and minimum throughput when the CRC is invalid at the end of SC decoder and SCL decoder is activated. The adaptive SCL decoder performs min. latency and max. throughput, when the output of the SC decoder has a valid CRC as a result of the CRC decoder. The increase on the BRAM usage is caused by the input data width of the asymmetric BRAM in the SCL decoder.

Table 3.8: Implementation results of adaptive successive cancellation list decoder.

N	L	FFs	LUTs	Slices	Block RAMs	Period (ns)	Min.-Max. Lat. (CC)	Min.-Max. Tp. (Mbps)
256	2	7809	12589	4886	14	7.38	152 - 1534	12 - 197
	4	10217	16565	6314	28	8.23	152 - 1539	11 - 177
	8	15241	28438	10344	56	9.67	152 - 1800	8 - 150
1024	2	27690	45021	12576	16	7.78	548 - 6640	10 - 225
	4	34099	53972	17219	32	8.43	548 - 6645	10 - 208

The resource percentage of SC, SCL and CRC decoders in the adaptive SCL decoder implementation is shown in Figure 3.9. As the list size increases, SCL decoder uses more resources compared to other decoders. For all cases, CRC decoder uses insignificant amount of resources.

Throughput results of our implementations varies with respect to Eb/No. At low Eb/No values, both the SC and the SCL decoders do not performs well. Therefore, the SCL decoder is frequently activated, after the CRC fails at the

Table 3.9: The resource usage percentage of SC, SCL and CRC decoders.

N	L	Decoder Type	FFs (%)	LUTs (%)	Slices (%)	BRAMs (%)
256	2	SCD	67.2	72.2	75.8	6.7
		SCLD	31.9	25.1	22.1	93.3
		CRC	0.9	2.7	2.1	0.0
	4	SCD	51.0	53.5	57.8	3.4
		SCLD	48.3	44.4	40.3	96.6
		CRC	0.7	2.1	1.9	0.0
	8	SCD	34.0	30.9	30.5	1.8
		SCLD	65.5	67.9	68.4	98.2
		CRC	0.5	1.1	1.1	0.0
1024	2	SCD	74.6	80.2	79.4	5.3
		SCLD	25.1	16.2	17.2	94.7
		CRC	0.3	3.6	3.5	0.0
	4	SCD	60.4	66.8	67.3	2.9
		SCLD	39.4	30.2	29.6	97.1
		CRC	0.2	3.0	3.1	0.0

end of the SC decoder. As Eb/No increases, the SCL decoder is less frequently activated and the throughput of adaptive SCL decoder improves. The throughput improvement with respect to Eb/No is shown in Figure 3.13 and 3.14.

The performance loss due to the implementation of the FRBS is shown in Figure 3.15 and 3.16 for $N = 256$ and $K = 128$. It is observed that there is not significant performance loss in both BER and FER curves with the list size $L = 32$ and $L = 16$.

For $N = 1024$ and $K = 512$, the performance of the adaptive SCL decoder implementation with bitonic sorter is shown in Figure 3.17, 3.18. At this point, the FRBS causes more than 0.5 dB performance loss when $L = 32$. The reason behind this performance loss might be the hard decision candidates, which has more than one valid CRC at the end of the SCL decoder.

In addition to these, the BER performance of the adaptive SCL with respect to different internal bit precisions is shown in Figure 3.19 and the FER performance is shown in Figure 3.20. Although the SCL decoder uses $P = P_i + \log N$ internal

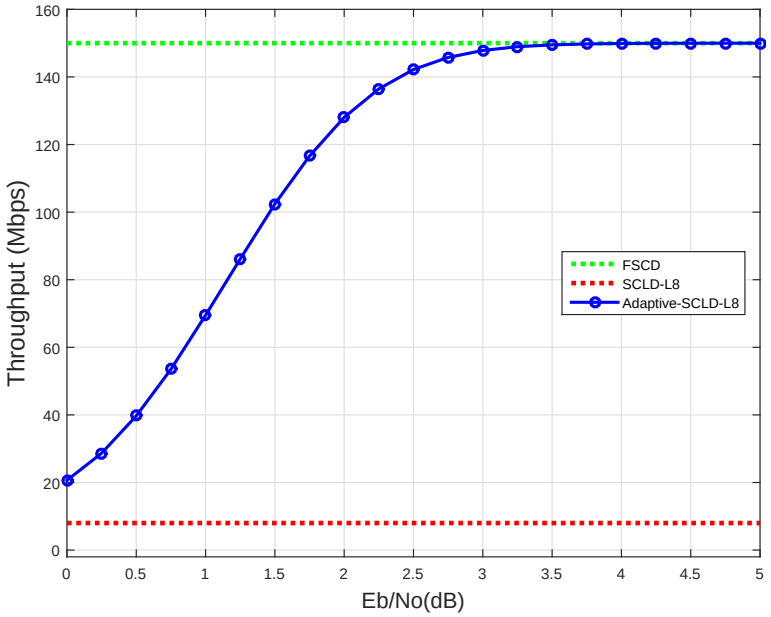


Figure 3.13: Throughput of the adaptive SCL decoder, $N = 256$, $K = 128$, $L = 8$.

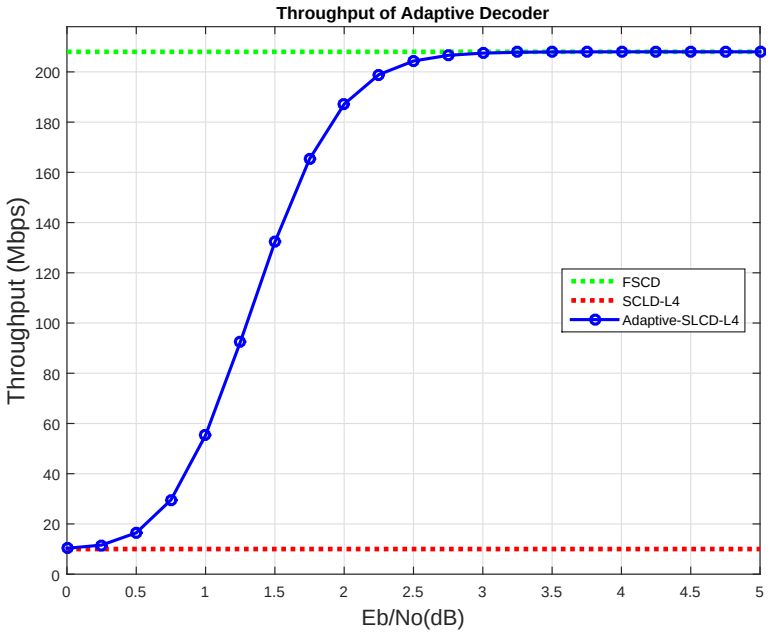


Figure 3.14: Throughput of the adaptive SCL decoder, $N = 1024$, $K = 512$, $L = 4$.

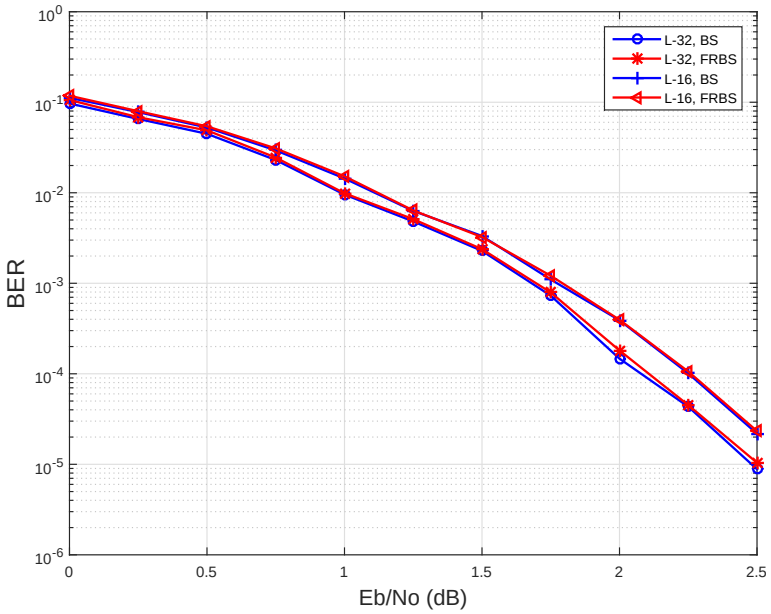


Figure 3.15: BER performance of the adaptive SCL decoder with bitonic sorter, $N = 256$, $K = 128$.

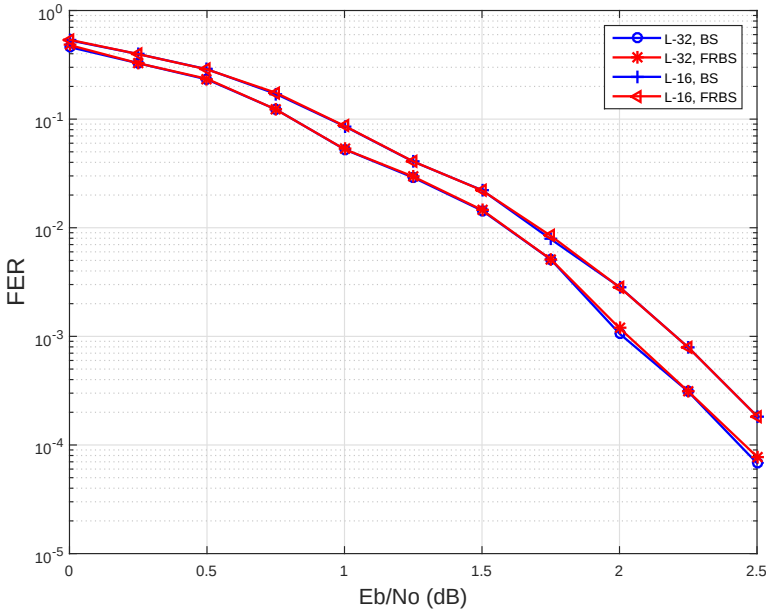


Figure 3.16: FER performance of the adaptive SCL decoder with bitonic sorter, $N = 256$, $K = 128$.

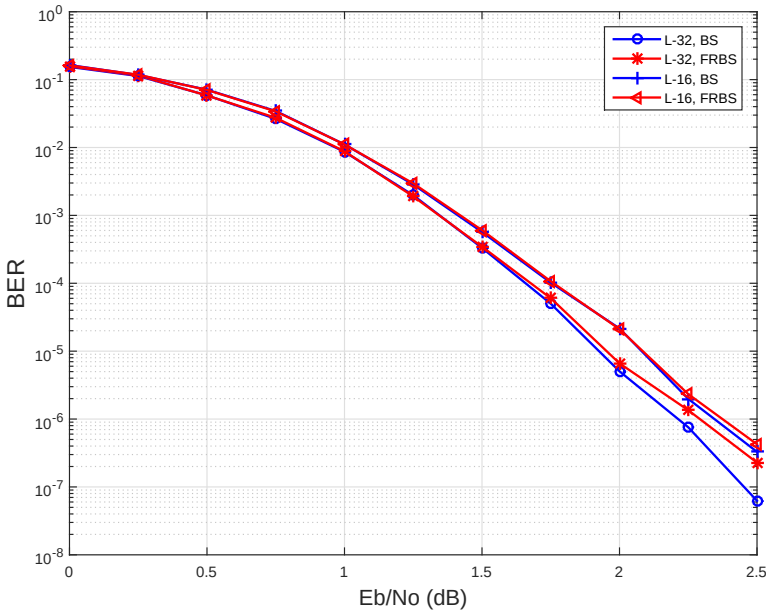


Figure 3.17: BER performance of the adaptive SCL decoder with bitonic sorter, $N = 1024$, $K = 512$.

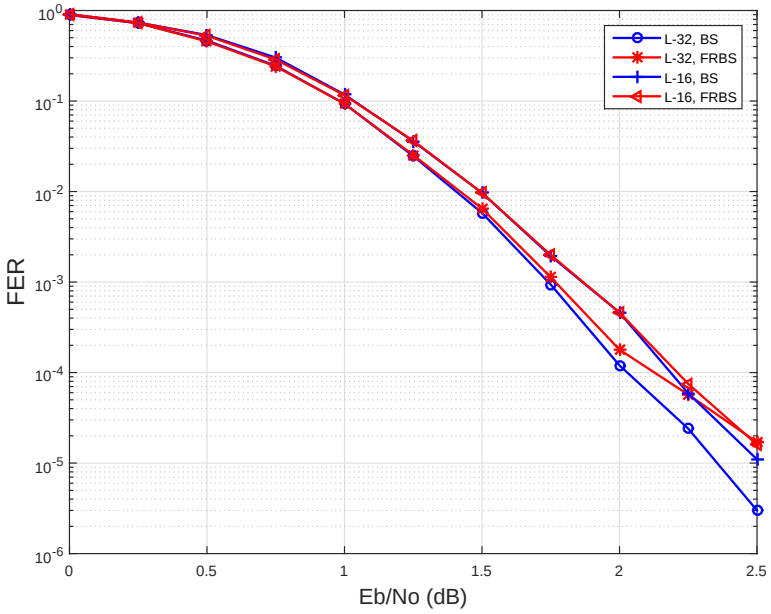


Figure 3.18: FER performance of the adaptive SCL decoder with bitonic sorter, $N = 1024$, $K = 512$.

LL precision, the $P = 11$ bit precision is adequate for $N = 1024$, $K = 512$, $L = 16$, $P_i = 6$ due to BER and FER performance results.

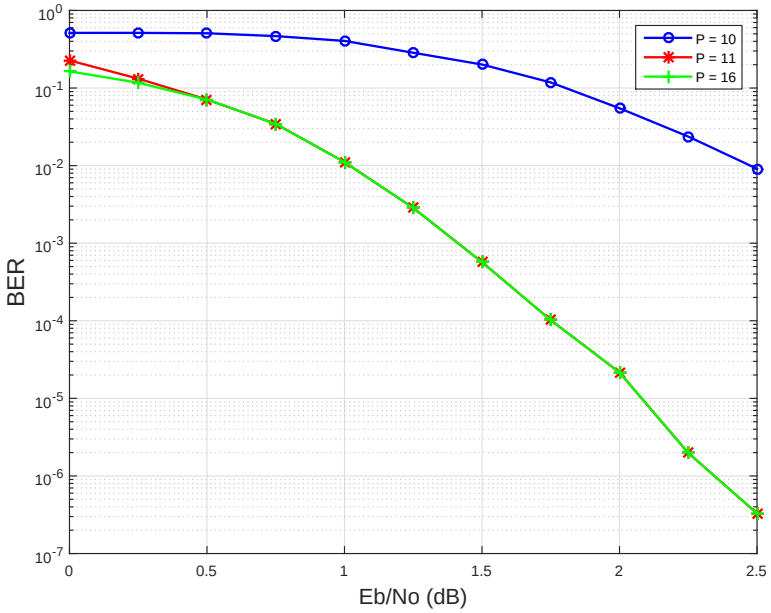


Figure 3.19: BER performance of the adaptive SCL decoder with different internal bit precisions, $N = 1024$, $K = 512$, $L = 16$, $P_i = 6$

3.6 Summary of the Chapter

In this chapter, we made a literature survey about SC and SCL decoder algorithms and implementations. We analyzed the SC decoding architectures in terms of reducing complexity and increasing throughput. Furthermore, we presented our adaptive successive cancellation list decoder implementation consisting SC, SCL and CRC decoders.

The SC decoder has four main modules as processing unit (PU), decision unit (DU), partial sum update (PSU) and controller logic (CL). For the implementation of the SC decoder, we use fast SC decoding method in [25]. This provides the detection of all frozen (rate-0), all free (rate-1), single partiy check (SPC) and repetition (REP) code segments to increase the throughput of the SC decoder. These code segments can be decoded with a maximum likelihood (ML) decoder

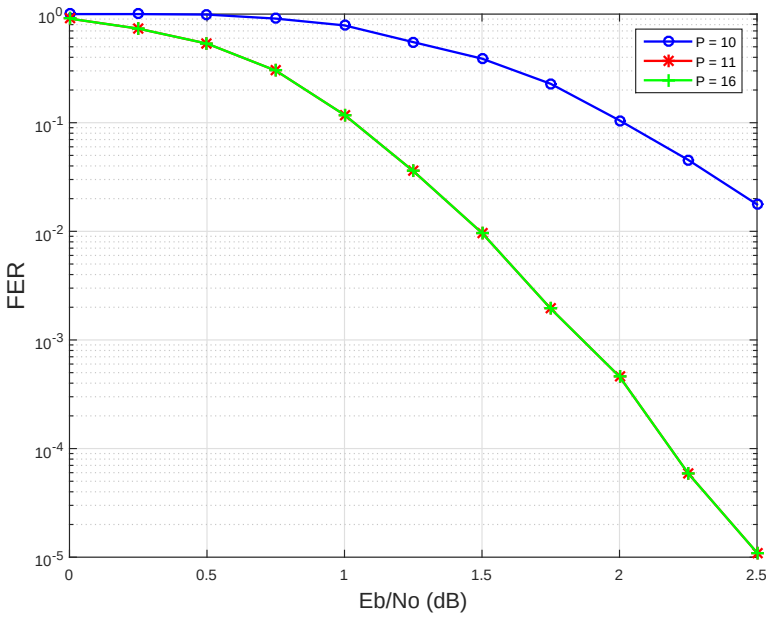


Figure 3.20: FER performance of the adaptive SCL decoder with different internal bit precisions, $N = 1024$, $K = 512$, $L = 16$

in a single clock cycle.

The SCL decoder consists of three main modules as list processing unit (LPU), list partial sum update (LPSU) and sorter. We use the semi-parallel architecture in [17] for the implementation of LPU. By this way, we define V LPUs, consist of total V soft decision router elements (SRE) and VL list processing elements (LPE). In the implementation of adaptive SCL decoder, we set $V = 4$. For the LPSU module, we use tree structure to minimize the latency. Thus, $N - 1$ hard decision router elements (HRE) are used for the LPSU module. For the sorter module, we have implemented there different sorters as bitonic sorter (BS), fast bitonic sorter (FBS) and fast reduced bitonic sorter (FRBS). We selected FRBS as our sorter to minimize the latency and resource usage caused by the sorter in SCL decoder. The FRBS introduces some insignificant performance loss that we showed in this chapter.

We considered a latency oriented design of the CRC decoder to maximize the throughput. The latency of the CRC decoder sets as 2, for the implementation of the adaptive SCL decoder. As a result, we achived an FPGA implementation

of the adaptive SCL decoder up to 225 Mb/s data throughput.

Chapter 4

Conclusion

In this thesis, we focused on the implementation of polar codes on standard commercial FPGA chips. An adaptive SCL algorithm was developed by combining the SC, the SCL and the CRC decoders. The implementation of the adaptive SCL decoder on FPGA was challenging for large list sizes due to increasing complexity and the excessive demand on routing paths. Thus, we achieved an implementation for the code block length $N = 256$ up to list size $L = 8$ and for $N = 1024$ up to list size $L = 4$. Although we had tried to implement with longer list sizes, the implementation failed due to routing congestion in FPGA. As a result, we achieved an adaptive SCL decoder, which has a throughput up to 225 Mb/s.

In Chapter 2, we reviewed the polar codes in terms of properties, code construction, encoding and decoding methods. We discussed the systematic and non-systematic polar coding. We presented the SC, the SCL and the adaptive SCL algorithms with their high level description. We presented both floating-point and fixed-point simulation results to get intuitive about hardware performance results. In Chapter 3, we presented important details about our adaptive list decoder implementation. Our implementation target is to design the SC decoder as throughput oriented and the SCL decoder as complexity oriented. In this manner, we preferred fast architecture for the SC decoder; semi-parallel architecture for the SCL decoder. In the SC decoder, we analyzed special code segments, REP,

SPC, R-0 and R-1 in detail and presented its implementation results. In SCL decoder, we developed FRBS to reduce sorting latency significantly, compared to a conventional BS. The FRBS implementation results were analyzed in Section 3.3.3.1. After that, we showed the implementation logic of the CRC decoder. Since it has insignificant contribution to the total implementation complexity, we reduced the pipeline stages in Section 3.4. Lastly, we showed implementation results of the adaptive SCL decoder in terms of resource usage, maximum clock frequency, memory usage, latency and throughput. The latency and the throughput of the adaptive SCL decoder varies with respect to operating SNR value. As a result, we achieved 100 Mb/s throughput approximately at 1.5 dB Eb/No in $N = 256$, $K = 128$, $L = 8$ parameters. For the length $N = 1024$, $K = 512$, $L = 4$ decoder at 1.25 dB Eb/No, our implementation works faster than 100 Mb/s data throughput.

For future work, we will reduce the complexity of our implementation and make it possible to implement adaptive $L = 32$ SCL decoders on standard commercial FPGA chips.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, Jul., Oct. 1948.

[2] E. Arıkan, "Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inform. Theory*, vol. 55, pp. 3051–3073, Jul. 2009.

[3] E. Arıkan and E. Telatar, "On the rate of channel polarization," *in Proc. IEEE Int. Sym. on Inform. Theory (ISIT)*, pp. 1493–1495, Jul. 2009.

[4] I. Tal and A. Vardy, "List decoding of polar codes," *in Proc. IEEE Int. Sym. Inf. Theory (ISIT)*, pp. 1–5, 2011.

[5] K. Niu and K. Chen, "Stack decoding of polar codes," *Elect. Lett.*, vol. 48, pp. 695–596, Jun. 2012.

[6] E. Arıkan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Comm. Lett.*, vol. 12, pp. 447–449, Jun. 2008.

[7] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," *in Proc. IEEE Int. Sym. on Inform. Theory (ISIT)*, pp. 1496–1500, 2009.

[8] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Trans. on Comm.*, vol. 60, pp. 3221–3227, Nov. 2012.

[9] E. Arıkan, "Systematic polar coding," *IEEE Comm. Lett.*, vol. 15, pp. 860–862, Aug. 2011.

[10] P. H. Winston, *Artificial Intelligence*. Addison-Wesley publishing company, 1993.

[11] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Comm.*, vol. 47, pp. 673–680, May. 1999.

[12] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," *in Proc. IEEE Int. Conf. on Acou. Speech and Signal Process. (ICASSP)*, pp. 1665–1668, Jun. 2011.

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

[14] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Increasing the speed of polar list decoders," *in Proc. Sig. Process. Sys. (SiPs) IEEE Workshop*, pp. 1–6, Oct. 2014.

[15] 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*, pp. 305–315, Dec. 2012.

[16] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "A successive cancellation decoder asic for a 1024-bit polar code in 180 nm cmos," *in Proc. IEEE Asian Solid-State Circuits Conf.*, pp. 205–508, Nov. 2012.

[17] C. Leroux, A. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. on Signal Process.*, vol. 61, pp. 289–299, Jan. 2013.

[18] A. Pamuk and E. Arıkan, "A two phase successive cancellation decoder architecture for polar codes," *in Proc. IEEE Int. Sym. on Inform. Theory (ISIT)*, pp. 957–961, Jul. 2013.

[19] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency sc polar decoder architectures," *in Proc. IEEE Int. Conf. on Comm. (ICC)*, pp. 3471–3475, Jun. 2012.

[20] C. Zhang and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 61, pp. 1241–1254, Apr. 2014.

[21] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Trans. on Signal Process.*, vol. 61, pp. 2429–2441, Jan. 2013.

[22] C. Zhang and K. K. Parhi, "Interleaved successive cancellation polar decoders," *in Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, pp. 401–404, Jun. 2014.

[23] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Comm. Lett.*, vol. 15, pp. 1378–1380, Dec. 2011.

[24] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Comm. Lett.*, vol. 17, pp. 725–728, Apr. 2013.

[25] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: algorithm and implementation," *IEEE Journal on Selec. Areas in Comm.*, vol. 32, pp. 946–957, May 2014.

[26] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 gbit/s unrolled hardware polar decoder," *Elect. Lett.*, vol. 51, pp. 762–763, May 2015.

[27] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Unrolled polar decoders, part i: hardware architectures." May 2015.

[28] O. Dizdar and E. Arıkan, "A high-throughput energy-efficient implementation of successive-cancellation decoder for polar codes using combinational logic." Mar. 2015.

[29] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Comm. Lett.*, vol. 16, pp. 1668–1671, Oct. 2012.

[30] A. Balatsoukas-Stimming and A. Burg, "Tree search architecture for list sc decoding of polar codes." Mar. 2013.

[31] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Tran. on Circuits and Systems II: Express Briefs*, vol. 61, pp. 609–613, May. 2014.

[32] C. Zhang, X. You, and J. Sha, "Hardware architecture for list successive cancellation polar decoder," *in Proc. IEEE Int. Sym. on Circuits and Systems (ISCAS)*, pp. 209–212, Jun. 2014.

[33] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "Llr-based successive cancellation list decoding of polar codes," *in Proc. IEEE Int. Conf. on Acou. Speech and Signal Process. (ICASSP)*, pp. 3903–3907, May 2014.

[34] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes." Oct. 2014.

[35] J. Lin and Z. Yan, "Efficient list decoder architecture for polar codes," *in Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, pp. 1022–1025, Jun. 2014.

[36] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Trans. on VLSI Systems*, vol. 23, pp. 2268–2280, Oct. 2015.

[37] Y. Fan, J. Chen, C. Xia, C. Tsui, J. Jin, H. Shen, and B. Li, "Low-latency list decoding of polar codes with double thresholding," *CoRR*, Apr. 2015.

[38] E. K. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pp. 307–314, ACM, 1968.