HARDWARE IMPLEMENTATION OF A SIMPLIFIED RADIX-4 SUCCESSIVE CANCELLATION DECODER FOR POLAR CODES

By

Hussein Galal Hussein Hassan

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING , CAIRO UNIVERSITY
GIZA, EGYPT
2019

HARDWARE IMPLEMENTATION OF A SIMPLIFIED RADIX-4 SUCCESSIVE CANCELLATION DECODER FOR POLAR CODES

By

Hussein Galal Hussein Hassan

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Prof. Hossam Aly Hassan Fahmy
Professor of Electronics and Communications Engineering
Electronics and Communications Engineering Department
Faculty of Engineering , Cairo University

FACULTY OF ENGINEERING , CAIRO UNIVERSITY
GIZA, EGYPT
2019

HARDWARE IMPLEMENTATION OF A SIMPLIFIED RADIX-4 SUCCESSIVE CANCELLATION DECODER FOR POLAR CODES

By

Hussein Galal Hussein Hassan

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by the
Examining Committee

Prof. Hossam Aly Hassan Fahmy, Thesis Main Advisor

Prof. Mohamed Mohamed Khairy, Internal Examiner

Assoc. Prof. Karim Gomaa Seddik, External Examiner
The Electronics and Communications Engineering Department,
School of Sciences and Engineering, AUC

FACULTY OF ENGINEERING , CAIRO UNIVERSITY
GIZA, EGYPT
2019

Engineer's Name:	Hussein Galal Hussein Hassan
Date of Birth:	16/09/1989
Nationality:	Egyptian
E-mail:	hussein.galal@ieee.org
Phone:	01066397911
Address:	11433
Registration Date:	01/10/2013
Awarding Date:	2019
Degree:	Master of Science
Department:	Electronics and Communications Engineering

Supervisors:

Prof. Hossam Aly Hassan Fahmy

Examiners:

Prof. Hossam Aly Hassan Fahmy	(Thesis main advisor)
Prof. Mohamed Mohamed Khairy	(Internal examiner)
Assoc. Prof. Karim Gomaa Seddik Electronics and Communications Engineering Department, School of Sciences and Engineering, AUC	(External examiner)

Title of Thesis:

HARDWARE IMPLEMENTATION OF A SIMPLIFIED RADIX-4 SUCCESSIVE CANCELLATION DECODER FOR POLAR CODES

Key Words:

Polar codes; Radix-4; partial sum lookahead; simplified successive cancellation decoding; special subcodes;

Summary:

In this thesis, we proposed a latency reduced decoder for the polar codes, this decoder is based on the successive cancellation decoding. The main idea is based on the usage of a radix-4 processing unit to calculate intermediate LLR values, a new last stage processing unit that is capable of decoding more than 4 bits in a cycle and the usage of the partial sum lookahead technique to improve hardware utilization and decrease the overall latency.

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute. I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Name: Hussein Galal Hussein Hassan Date:

Signature:

Dedication

I would like to dedicate this thesis for to my parents, my sister, my wife, and my daughters: Malak and Yara for their care, support and extreme patience.

Acknowledgments

I would like to express my gratitude to my advisor Dr. Hossam A. H. Fahmy for his continuous help and patient guidance.
I would like also to thank Dr. Amr M. Hussien for his detailed support and extreme patience till reaching this point.
I would like to thank my family for their support and patience throughout the long journey of my Masters studies.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

C	**C**hannel capacity
SC	**S**uccessive **C**ancellation decoder
SCL	**S**uccessive **C**ancellation **L**ist decoder
LR	**L**ikelihood **R**atio
LLR	**L**og **L**ikelihood **R**atio
LTE	**L**ong **T**erm **E**volution
BER	**B**it **E**rror **R**ate
BEC	**B**inary **E**rasure **C**hannel
B-DMC	**B**inary **D**iscrete **M**emoryless **C**hannel
AWGN	**A**dditive **W**hite **G**aussian **N**oise
BP	**B**elief **P**ropagation decoder
Z(W)	Bhattacharyya coefficient
I(W)	Mutual information
W	Transition probability
PU	**P**rocessing **U**nit
LSPU	**L**ast **S**tage **P**rocessing **U**nit

Abstract

Polar codes recently received high attention by researchers as proven to approach channel capacity at higher codeword length. However, the decoding latency grows significantly with codeword length, rendering implementation for latency constrained applications impossible. To tackle this problem, this thesis proposes a polar decoder architecture based on radix-4 processing units with a special last stage processing unit to decode up to 16 bits in the same clock. In addition, it proposes decoding extended special sub-codes to reduce latency. Moreover, it uses partial sum look-ahead technique, resulting in a high throughput with low latency decoding architecture.

Publications

Hassan, Hussein GH, Amr MA Hussien, and Hossam AH Fahmy. "Radix-4 successive cancellation decoding of polar codes with partial sum lookahead." Microelectronics (ICM), 2017 29th International Conference on. IEEE, 2017.

Hassan, Hussein GH, Amr MA Hussien, and Hossam AH Fahmy. "A simplified radix-4 successive cancellation decoder with partial sum lookahead." AEU-International Journal of Electronics and Communications 96 (2018): 267-272.

Chapter 1: Introduction

1.1 Introduction

The ever going improvement in the information technology applications and usage is adding a severe demand to a cost efficient reliable high data rate communication system. Those applications range in different fields from the wireless communication systems to other wired yet low power systems to digital storage systems. The wireless mobile communications field for example have witnessed lots of improvements which added extra complexity and demand on the systems starting from the first wireless mobile call using analog signals on the 1980s to the second-generation mobile communication (2G) in the 1990s, the data rates erupted after that by the deployment of the (3G) networks in the 2000s ,in 2010 the long Term Evolution (LTE-4G) was used in commercial networks boosting the data rates to limits we have never reached. Currently the next technology (5G) is being developed, the main characteristics will be fiber-like data rates along with "zero-like" latencies, while being able to connect about 100 billion devices.
Another example is the digital storage systems as they typically write and read huge chunks of data at very high rates , applying error correction techniques before writing the data is necessary to minimize bit error rate (BER) when reading, yet the error correction technique used must have good waterfall region performance with very low error-floors and moderate throughput of several gigabits per second (GbpS).
Another aspect is the huge breakthrough in wearables field that requires ultra low power communication systems. Fulfilling their low power requirements may decrease the SNR and increase consequents. Therefore, powerful error correction techniques shall be adopted. Those challenges made a need to squeeze more data and to increase the data rates while using the same bandwidth and same power or even using the same old data rate but with much less power consumption figures.
That leads the recent researches to try achieving the maximum allowable rate of reliable data communication stated by Shannon in [21].

1.2 Channel Coding

Since the ultimate goal of a communication system is to provide a cost efficient technique for communicating information between two point with acceptable rate and reliability from the system point of view. The key aspect affecting the reliability is the signal energy per bit to noise power spectral density ratio (bit-to-noise). This ratio along with the modulation scheme defines the BER. For a fixed bit-to-noise ratio and a defined modulation scheme, the only way to improve the reliable data rate (number of true transmitted bits per second) is to use some error-control coding techniques [22].
Even if the current reliable data rate without error-control coding is acceptable the designer can still make use of error-control coding to reduce the bit-to-noise ratio while maintaining the rate thus reducing both transmitted power and hardware costs.

Most of error-control techniques mainly introduce redundancy in the message in order to withstand the noise introduced by the channel in the receiver and to correct some errors that may happen in reception like forward error correction (FEC).
The communication systems usually contains source encoding in the transmitter and then a channel encoder at receiver side , the signal will have some noise and distortion that introduce error in sent message , in order to correct this message to some extent, introducing some redundancy before the transmitter will be handy.

Figure 1.1 contains a typical communication system showing source coding / decoding and channel coding / decoding. The Data source generates the raw data bits, then they are formatted to match the communication system , this formatting is done by compressing the data without risking the information loss, the last stage before uploading the data to transmission medium -whether wired or wireless- is to add the error correction redundancy in the channel coding block.
On the receiver side, the error detection and correction is handled by the channel decoder, the data is then returned to the application format by decompressing in the source decoder.

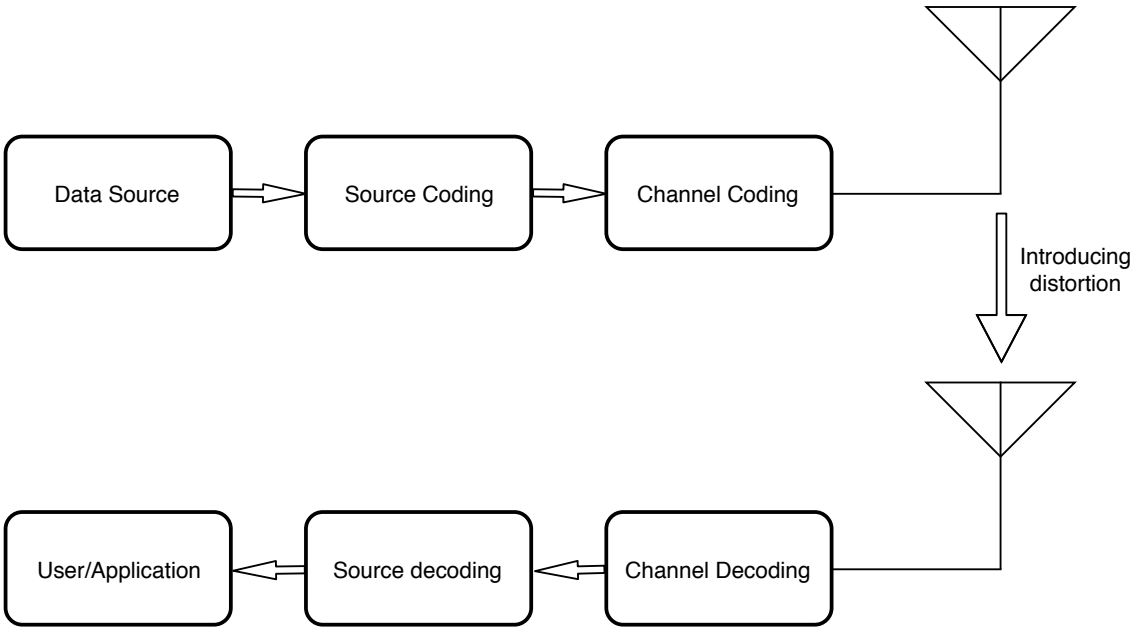


Figure 1.1: A simple communication system

It is important for the system designer though to trade-off between the data rate improvement using channel coding technique and the increased transmission bandwidth along with the added complexity for implementing the channel decoder operation.

The error-correcting channel coding techniques can be classified into block codes and convolutional codes. The main difference between the two types of codes is the presence of a memory in the encoders of those codes. In the block encoding the whole data block is saved into a memory member. The encoder operates on all of them simultaneously. In the convolutional encoder a sliding window technique is used so it can accept a continuous sequence of data bits and generates a continuous

sequence of the encoded bits at a higher rate than the block encoder.

1.3 Channel Capacity

An Important property of a communication channel is the Channel capacity [21]. The Channel Capacity C is the number of channels that have the maximum mutual information I. Since Shannon only stated the channel capacity but not a way to design a coding technique to reach this capacity, reaching the channel capacity was the target of lots of researchers to get the most reliable rate of the used channel. Lots of research theoretically reached the channel capacity but there were more problems in implementation. Claude Berrou was able to introduce turbo codes [7] that achieved high performance of error correction over the used channel at a capacity within a small margin of channel capacity. Also Low-Density Parity-Check (LDPC) was developed by Robert Gallager in [12] and it showed to have similar performance to that of turbo codes. Those techniques have been used widely in communication systems, having small differences may bias the designer towards one or another when designing a new system.

1.4 Is coding dead?

The channel capacity is achieved and one can think that there is no room for further research in this field. In 1974 [17] stated the motto reputable this period "Coding is dead except for the deep-space channel" and illustrated this motto.

Several researches targeted reaching the channel capacity upper boundary. The LDPC codes [12] reached a near Shannon capacity. Then the turbo codes [7] reached a better performance. For long time, lots of researchers stuck with those coding techniques. This caused the saying that channel coding is died to get famous.
In 2009 E. Arikan introduced the channel polarization along with a successive cancellation decoding technique for this codes which was provably the first capacity achieving codes with a linear simple form.

1.5 Research opportunity in polar codes

The main issue with the polar codes were that the decoding performance was achieved when the codeword length reaches infinity, further research proved that the successive cancellation decoding can hold a decent bit error correction performance with low complexity in codeword length that vary in the range 2^{10} to 2^{20}.
The successive cancellation decoder presented by Arikan was of complexity $O(NlogN)$ while its decoding latency was equal to $2N-2$. This latency was very high that it was a show stopper in the usage of polar codes in real life applications.
The remaining problem was being able to implement this low complex yet high latency decoding technique in a feasible latency for its application.

1.6 Conclusion

The current demand of the recent technology is pushing the information theory specialists to adopt more efficient error detection and correction techniques. The channel coding means the addition of extra redundancy to the source message to enable the detection and correction of error. Channel coding was said to be dead. After the invention of LDPC and turbo codes, there where no valuable improvements till the invention of polar codes by E. Arikan. The Polar codes suffered a very high latency (initially equal $2N - 2$). This offered a huge research opportunity to tackle this problem.

Chapter 2: Literature Review

2.1 Introduction

Channel polarization is presented and proved by E.Arikan to be the first channel capacity achieving coding technique that is simple and with a linear form. It depends on combining a set of binary discrete memoryless channels into another set of channels where the polarization starts to appear, the polarization appears in the sense that after the combination of these channels, some of them reaches higher reliability than others. Polar codes have a high resemblance to the Reed-Muller coding, yet the error performance of polar codes is better. Several encoder implementations are presented. Current state of the art decoding techniques were surveyed in this chapter.

2.2 Channel polarization

E. Arikan [3] noticed the effect of channel polarization when combining a set of N symmetric binary discrete memoryless channels (BDMC) using modulo-2 sum of bits in a flow graph like the one used with Reed-Muller codes into another set of channels, it was found that after the polarization some channels will be able to transmit data with better performance than others.
The new set of channels will maintain the total Mutual information(i.e the overall mutual information will be $N * I(W)$). Yet the mutual information of some channels will be $< I(W)$ so it will be marked unreliable, other channels will have mutual information $> I(W)$ and will be marked reliable.
As N tends to ∞ the channels will be either completely reliable or completely unreliable, the ratio of the completely reliable channels will be equal to Shannon's channel capacity.

2.2.1 Mutual information

The mutual information term in information theory defines a metric to calculate the dependence between two variables. Consider a channel with input denoted as x and the output as y. If the mutual information of x and y is large, this means there is a strong relation which helps regaining x from y.
Mutual information generally measures the similarity of $P(x, y)$ to $P(x)P(y)$. The mutual information is calculated for discrete values as:

$I(X;Y) = \sum_Y \sum_X p(x,y) log(\frac{p(x,y)}{p(x)p(y)})$
For Channel coding :

$$I(W) = \sum_y \sum_x \frac{1}{2} W(y|x) log \frac{W(y|x)}{\frac{1}{2}W(y|0)+\frac{1}{2}W(y|1)}$$

For B-DMC W:

$I(W) \geqslant log \frac{2}{1+Z(W)}$

$I(W) \leqslant \sqrt{1 - Z(W)^2}$
where $Z(W)$ is the Bhattacharyya parameter.

2.2.2 Bhattacharyya parameter

The Bhattacharyya coefficient(parameter) [3] is a metric to measure the closeness of two probability distributions. For the distributions of different values of a several bits in a decoder, the closeness is a sign of being not able to separate the two cases from each other, thus not being able to decide the bit correctly.
The Bhattacharyya coefficient is calculated as:

$Z(W) = \sum_y \sqrt{W(y|0)W(y|1)}$

The Bhattacharyya coefficient is the inverse of the Bhattacharyya distance. Bhattacharyya coefficient takes a value in $[0, 1]$.
Arikan [3] used the Bhattacharyya coefficient instead of the Hamming distance to differentiate the reliable channels(least Bhattacharyya coefficient) from non reliable ones.
An important property of the Bhattacharyya coefficient is that when the number of channels to combine increase the mutual information gets larger and the Bhattacharyya coefficient gets smaller.
As $N \to \infty$ $Z(W) \to 0$ $I(W) \to 1$
or
$Z(W) \to 1$ $I(W) \to 0$

2.3 Polar encoders

The channel polarization is carried out with an encoder similar to that needed for Reed-Muller encoders, yet the reliable channel selection is done by comparing the Bhattacharyya parameter instead of using the hamming weight selection [2].

2.3.1 Reed-Muller vs Polar codes

Reed-Muller is a channel coding technique that has a great similarity with polar codes from the encoder point of view, both techniques use the same channel combining way by the use of Kronecker power of G.
Yet the only difference is using Hamming distance to differentiate the reliable channels instead of Bhattacharyya coefficient in polar codes.

2.3.2 Reed-Muller vs Polar codes performance on BEC

Table 2.1 and Figure 2.1 shows the difference in performance between Polar codes and Reed-Muller codes on a binary erasure channel(BEC) with a rate $\frac{1}{2}$ and a codeword length 32 [3].

Table 2.1: BER for polar codes and Reed-Muller(N,K) = (32,16) Codes

	Erasure probability				
Type of Code	0.05	0.15	0.25	0.35	0.45
RM(2,5)	0.00000	0.00039	0.01169	0.06525	0.24507
Polar	0.00001	0.00056	0.00702	0.05005	0.16722

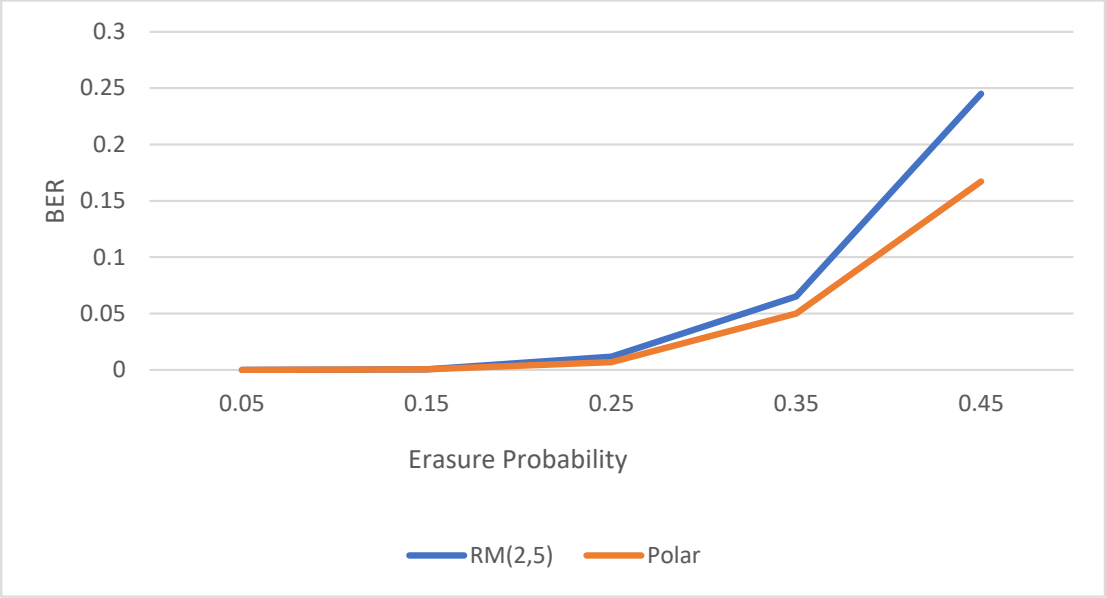


Figure 2.1: BER for polar codes and Reed-Muller(N,K) = (32,16) Codes

Table 2.2 and Figure 2.2 shows the performance of Reed-Muller codes on a BEC with a $N = 256$ [3].

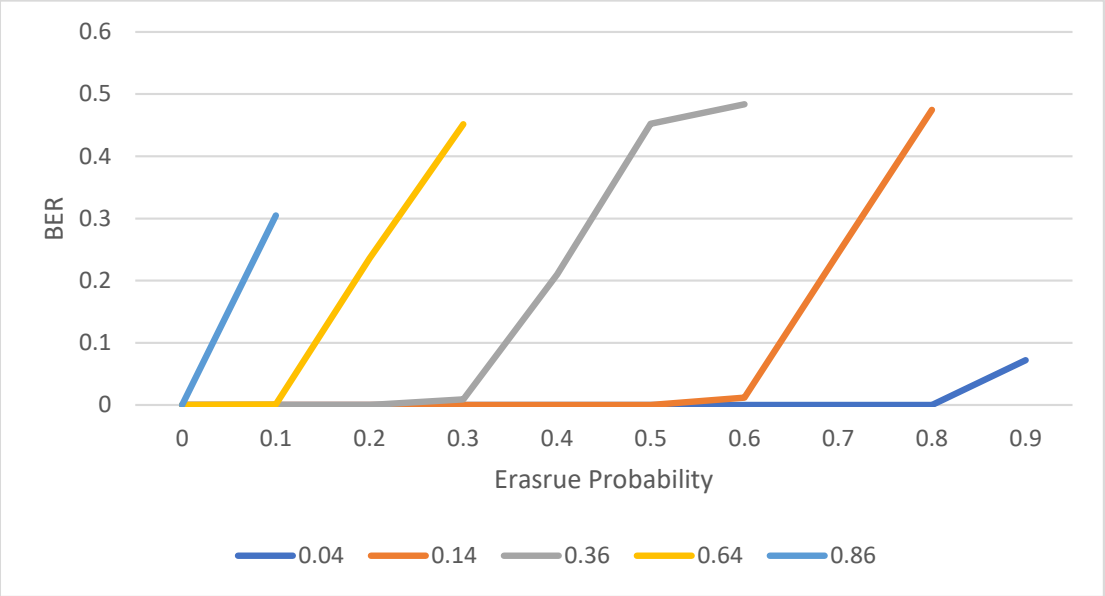


Figure 2.2: BER for Reed-Muller N=256 runs=1000 Iterations = 60 on BEC

Table 2.3 and Figure 2.3 shows the performance of polar codes on a BEC with a $N = 256$ [3].

Table 2.4 and Figure 2.4 shows the performance of Reed-Muller codes on a BSC with a $N = 256$ [3].

Table 2.5 and Figure 2.5 shows the performance of polar codes on a BSC with a $N = 256$ [3].

2.3.3 Polar encoders implementations

Because of being very simple compared to the polar decoders, the polar encoder received less amount of research while approximately all the researches was targeting to achieve a better performance polar decoder. Yet there are several ways to implement polar encoders.

Table 2.2: BER for Reed-Muller N=256 runs=1000 Iterations = 60 on BEC

	Erasure probability								
Rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.04	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.07200
0.14	0.00000	0.00000	0.00000	0.00000	0.00000	0.01195	0.24395	0.47465	-
0.36	0.00000	0.00000	0.00933	0.21003	0.45208	0.48368	-	-	-
0.64	0.00083	0.23588	0.45156	-	-	-	-	-	-
0.86	0.30484	-	-	-	-	-	-	-	-

Table 2.3: BER for polar codes N=256 runs=1000 Iterations = 60 on BEC

	Erasure probability								
Rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.04	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.01378
0.14	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00308	0.20427	-
0.36	0.00000	0.00000	0.00000	0.00011	0.02087	0.29740	-	-	-
0.64	0.00000	0.00363	0.16155	-	-	-	-	-	-
0.86	0.10419	-	-	-	-	-	-	-	-

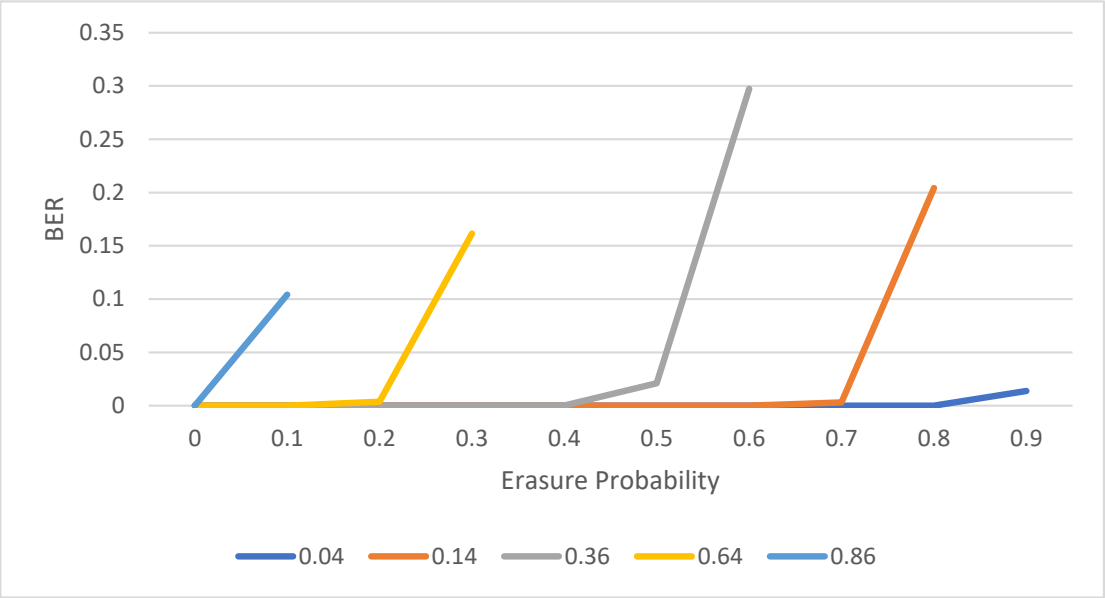


Figure 2.3: BER for polar codes N=256 runs=1000 Iterations = 60 on BEC

Table 2.4: BER for Reed-Muller N=256 runs=1000 Iterations = 60 on BSC

	Error probability										
Rate	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.36	0.37	0.38	0.39
0.04	0.00000	0.00000	0.00000	0.00000	0.00000	0.01111	0.10756	0.14389	0.21100	0.25778	0.30400
0.14	0.00000	0.00000	0.00584	0.12159	0.34368	-	-	-	-	-	-
0.36	0.00086	0.12829	0.39794	-	-	-	-	-	-	-	-

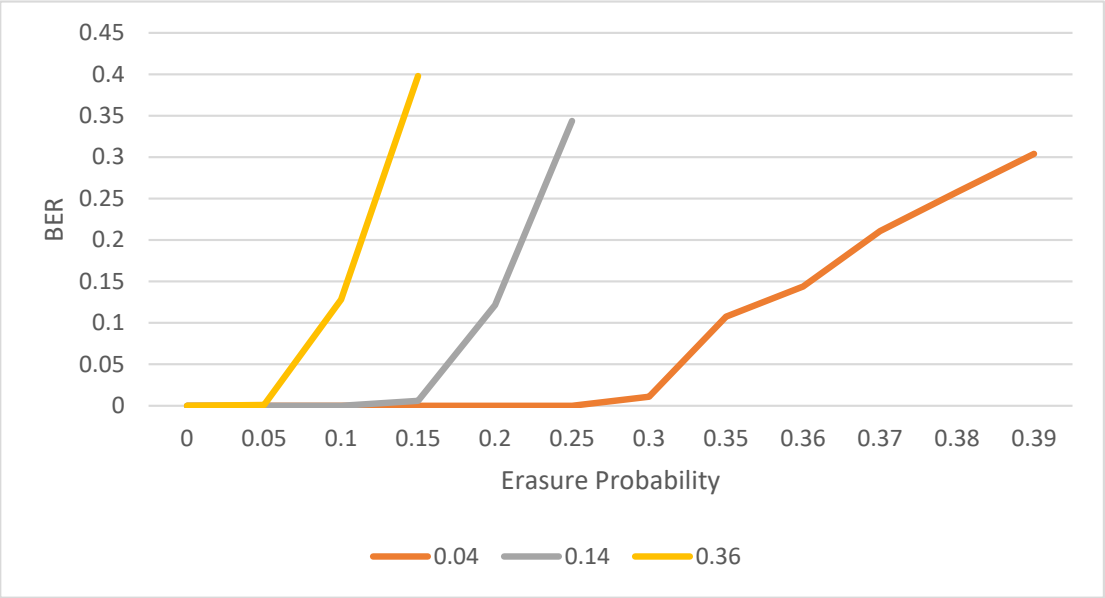


Figure 2.4: BER for Reed-Muller N=256 runs=1000 Iterations = 60 on BSC

Table 2.5: BER for polar codes N=256 runs=1000 Iterations = 60 on BSC

	Error probability										
Rate	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.36	0.37	0.38	0.39
0.04	0.00000	0.00000	0.00000	0.00000	0.00000	0.00344	0.04667	0.07844	0.11889	0.17722	0.21378
0.14	0.00000	0.00000	0.00000	0.01230	0.14297	-	-	-	-	-	-
0.36	0.00000	0.02418	0.21481	-	-	-	-	-	-	-	-

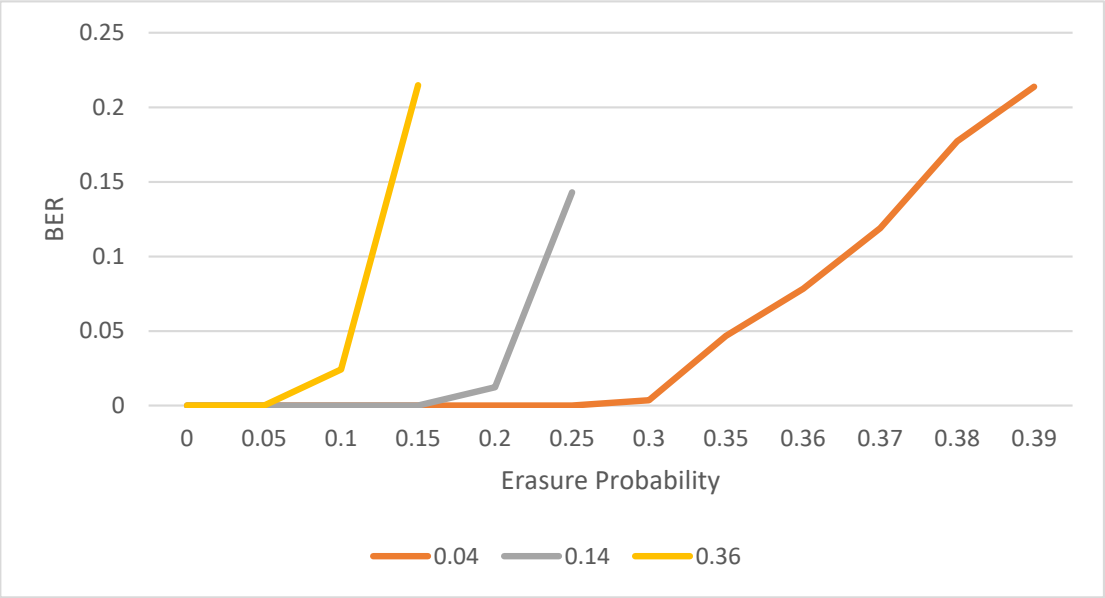


Figure 2.5: BER for polar codes N=256 runs=1000 Iterations = 60 on BSC

2.3.3.1 Combinational polar encoder

The combinational encoder can encode N bits in the same clock cycle, this fits short codewords as for longer codewords this will have a very long critical path, i.e. long clock period. This is the least complex encoder although the area may grow huge when long code words are applied.
Figure 2.6 shows the implementation of 8-bit combinational encoder with coding rate $= \frac{1}{2}$. It is important to notice that not all of the 8-bits are used because some bits are frozen (at "0" in this case) when the implementation is not programmable for different coding rates. For example, the XOR gate count is reduced from 12 gates to just 7 gates in the case of 8-bit coding with coding rate $\frac{1}{2}$.

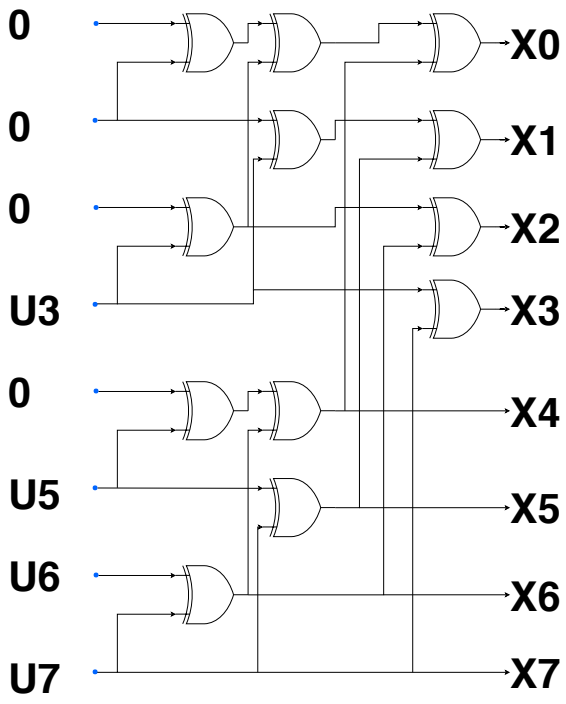


Figure 2.6: 8-bit combinational (8,4)encoder

2.3.3.2 Pipelined polar encoder

This encoder can encode N bits in N clock cycles, this fits short and long codewords, it has a relatively short critical path so it can operate on high frequencies. This is a more complex encoder but the area used by this implementation is relatively small. Figure 2.7 shows the implementation of 8-bit pipelined encoder with coding rate $= \frac{1}{2}$. This encoder can be divided into 2 main parts, the matrix generation unit which is able to generate a new row of the Kronecker product G_N, the second part (the calculation module) contains a shift register to shift the bit to decode with the corresponding G_N row, this bit is anded with the Kronecker input and then XORed with the old result of each column.

The matrix generation part area can further be reduced by the usage of Kronecker

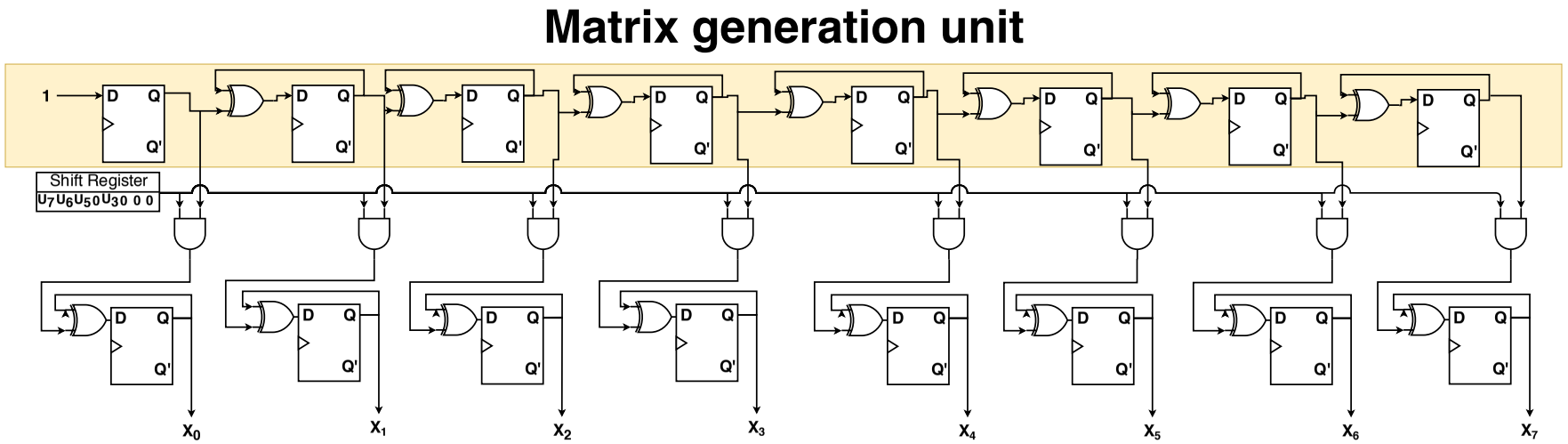


Figure 2.7: 8-bit pipelined (8,4)encoder

power property, the N generation matrix G_N can be written as

$$G_N = \begin{bmatrix} G_{\frac{N}{2}} & 0 \\ G_{\frac{N}{2}} & G_{\frac{N}{2}} \end{bmatrix}$$

$$G_N = G_{\frac{N}{2}} \oplus \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

This form of the G_N matrix can be translated in words that the G_N matrix can be calculated by calculating the $G_{\frac{N}{2}}$ matrix and then using it after being anded with another G_2 matrix [9].

Figure 2.8 shows an update to the generation matrix by the use of $G_{\frac{N}{2}}$ and another tweaked G_2 generation matrix, this G_2 is tweaked to switch only when the other $G_{\frac{N}{2}}$ finishes a complete cycle, this imitates the property of the Kronecker product of G_2.
Figure 2.9 shows an abstract of the generation matrix proposed in figure 2.8. Since this matrix generation still contains $N/2$ generation matrix, this matrix can be recursively implemented the same way as G_N, thus reaching the general form represented in figure2.10. Although the reduced area in figure 2.8,2.9 is approximately equal to the added area of the AND array, this will not hold with longer codewords and the new implementation would be smaller in area.

2.4 Polar decoders

2.4.1 Successive cancellation decoders

The channel polarization of Binary symmetric channel (BSC) performance was proved using the likelihood based successive cancellation decoders, Although the successive cancellation decoder achieved the required performance and capacity (asymptotically when N tends to infinity), it suffered a huge decoding latency $O(NlogN)$ because of its successive behavior.
The usage of successive cancellation decoder with binary erasure channel (BEC) was proven to achieve the error correction performance with both theoretical and numerical approaches in [24].
Several researches have been done to solve the decoding latency by optimizing the successive cancellation decoders or even proposing other decoders such as :

1. Belief propagation decoders.
2. Successive cancellation list decoders.

2.4.2 Belief propagation decoders

Because of the great encoding similarity, polar codes can be easily decoded with the belief-propagation decoders. The belief-propagation decoders were originally used with the Reed-Muller technique.

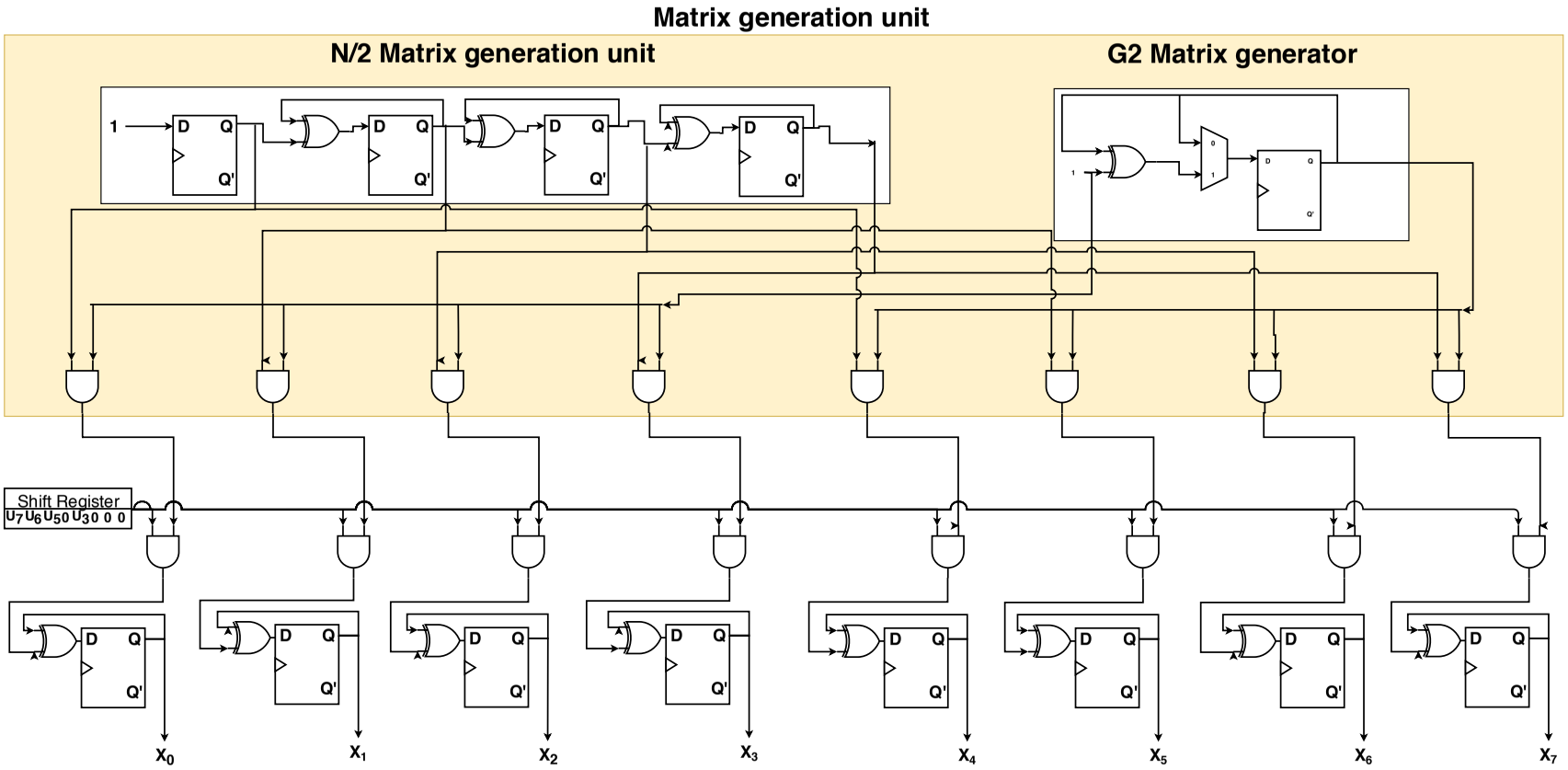
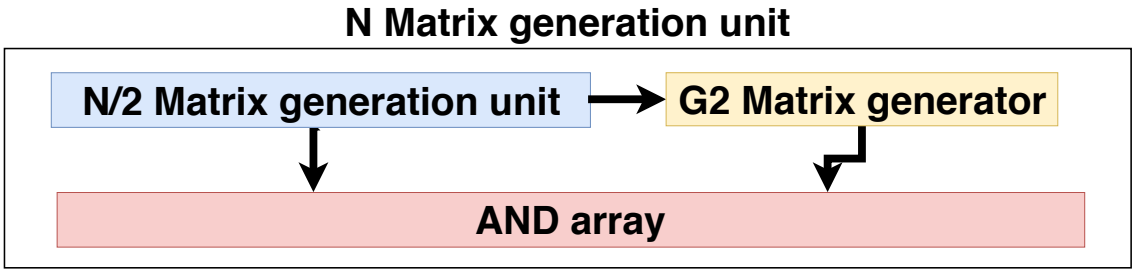


Figure 2.8: 8-bit pipelined (8,4)encoder using G_4 matrix ,G_2 matrix and an AND array

Figure 2.9: N generation matrix composed of AND array , $G_{\frac{N}{2}}$ and G_N



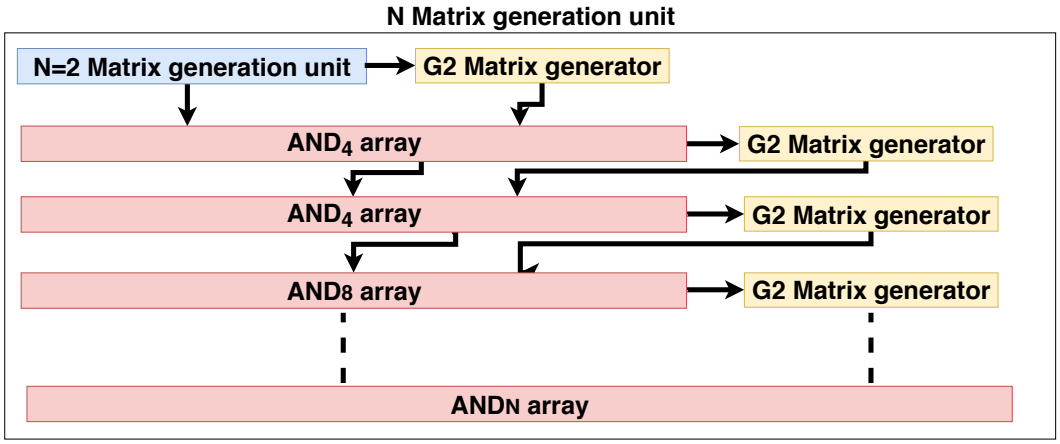


Figure 2.10: N generation matrix composed of G_2 and multiple AND arrays

To compare the polar codes with reed-muller the work in [2] used the same rate and the same decoder. The error probabilities for BSC and erasure probabilities for BEC were calculated. It was found that the polar codes have clear performance advantage over the reed-muller codes even in small codeword length $N = 32$. This codeword length $N = 32$ is the shortest codeword length were both coding techniques differ. For example at rate 0.45 using BEC channel the erasure probability of RM(2,5) was found to be 0.24507 while polar codes erasure probability was equal to 0.16722.

2.4.3 Successive cancellation list decoders (SCL)

The ordinary successive cancellation decoder was able to achieve its error correction performance only in long codeword length. Several work adopted finding a way to achieve better error correction performance. This of course comes on the price of some added complexity.
The successive cancellation decoder uses the decoded bit for decoding the next bits. If those bits are frozen, it sticks with the 0 path (assuming frozen bits are set to 0). This decision is took regardless the actual values of the likelihood ratios. Thus if an error occurs while decoding a certain bit, there is no way to fix this after. There is a risk of decoding all the subsequent bits in wrong manner since they are depending on the faulty bit.
The successive cancellation list decoders keeps several decoding paths (multiple values for the decoded bits) while decoding. It then drops the paths with the worst metric when the list count is reached.
In other words, first the starting bit Likelihood ratios are calculated and likelihood metrics are calculated for these 2 paths, adding another bit will add 2 more paths (resulting in 4 paths with 4 new metrics), the decoder keeps recalculating the required metrics every other bits till the count of paths reaches the maximum path L. Once the maximum list count is reached, for every another packet to decode 2L metrics are calculated and the L worst metric paths are dropped, this continues till reaching the codeword length N, the best metric path is decided to be the decoded codeword. It is easy to see that although the probability of error decreased (thus the performance of the decoder is increased) this came at a price of high decoding latency $O(LN^2)$ and high complexity. It can be easily noted that the successive cancellation decoder is a special case of the list decoder were the list size is set to 1.
Clearly the list decoder is higher in complexity yet gives a better error-correction performance on short code lengths as shown in [23].
A software decoder implementation of successive cancellation list decoders was presented in [20].
Figure 2.11 shows the different paths while decoding a codeword of length $N = 4$ and list size $L = 3$, ordinary successive cancellation decoder is denoted by the dashed arrows, while the list decoder paths are denoted by both the dotted and dashed arrows.
Although the first bit is known to be frozen, the successive cancellation list decoder used the paths with $\hat{u_0} = 1$ and $\hat{u_0} = 0$.

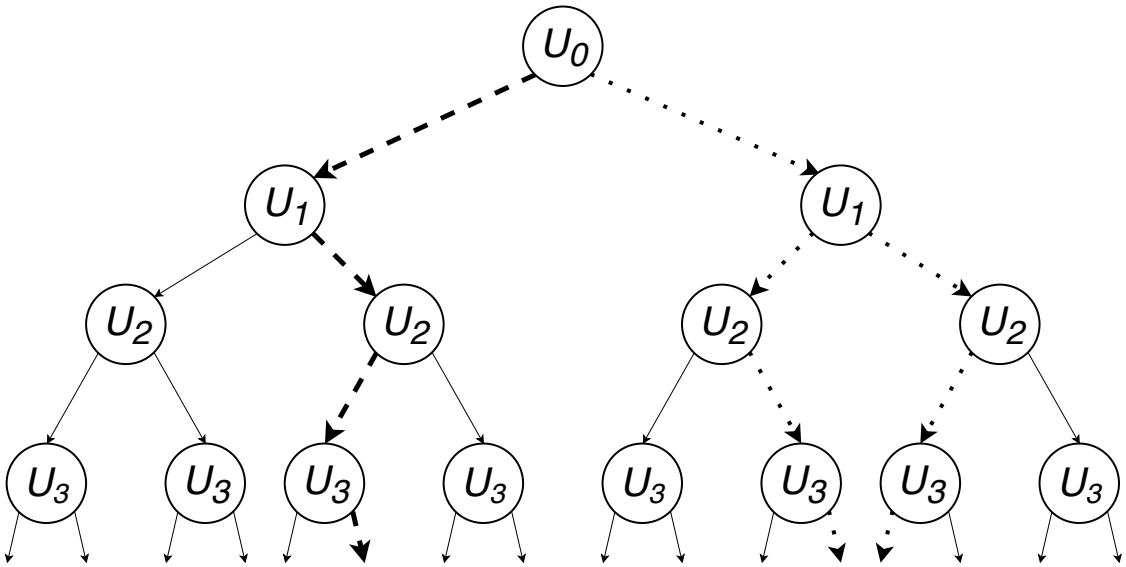


Figure 2.11: Decoding paths of N=4 codeword, List size L=3

2.4.4 Multi-bit successive cancellation decoding

Without any performance loss and by returning back to the LLR meaning at a certain point of the flow graph, [26] proposed a technique to decide the value of 2 consecutive bits together from the previous stage. Deciding the consecutive bits like this helps to skip the decoding cycles of calculating LLRs and then deciding the values of the 2-bits, this affected the overall latency tremendously by reducing the 4 cycles to decode 2 consecutive bits into just one cycle. The multi-bit decoding technique was ported to the SCL decoders. The SCL latency was reduced severely [27].

2.5 Latency reduction efforts

A lot of work has been done recently to reduce the high latency of SC decoding. This is classified into performance improvement by using short block length while maintaining the error correction performance at application acceptable levels, minimizing the decoding latency, and even simplifying the successive cancellation decoder itself. Instead of using the likelihood ratios, log likelihood ratios(LLRs) were used with minimum approximation of f function equation, therefore eliminating the multiplication and division operations [15]. LLRs were even ported to successive cancellation list decoders thus reducing its complexity. In [29] a general log-likelihood-ratio successive cancellation list (SCL) decoder is introduced, where decision of 2^k bits can be determined simultaneously for arbitrary k. A high throughput flexible implementation of a simplified successive cancellation decoder was presented in [19] and was about 8 times faster that state of the art decoders when published.
The polar decoding of long sets of used non frozen bits (rate 1) and long sets of frozen bits (rate 0) is pretty easier than other special sub-codes rates. Those sub-codes can

be decoded in a special way with less clock cycles than handling those sub-codes as other rate codes [1], this improves the decoding latency for longer codes since the rate 0,1 patterns will be longer and can be handled in shorter period.

2.6 Literature survey

A multi kernel polar decoder was presented in [8], this multi kernel decoder targets maximizing the minimum distance. A comparison between polar decoders, LDPC and Turbo decoders was done in [5], while surveying the error-correction performance and hardware efficiency of different polar decoder techniques.
Since decoding a bit using successive cancellation decoder depends on the previously decoded bits in the form of partial sums, [6] proposed a generation technique for the partial sums, it is divided into 2 parts: a matrix generation circuit and a shift-register-based partial sum calculator. [13] transformed the unrolled architecture to a multi-mode decoder along with a dedicated decoder for polar codes. A multilevel structure was used in generating the matrix as proposed in [9].
[18] introduced a scalable architecture for SC decoding using a semi-parallel encoder-based partial-sum calculation module. Both the belief propagation decoder and the successive cancellation decoder are concatenated forming a hybrid decoder in [25]. Different approaches to improve successive cancellation decoding along with the analysis of the stochastic SC decoding performance were presented in [28]. [10] introduced a low power high-throughput combinational SC decoder. [4] studied the faulty SC decoding of polar codes and proved that fully reliable data transfer is not possible at any rate.

2.7 Thesis proposal

Starting with the work in [26], the 2-bit SC decoder was generalized to a radix-4 architecture that is able to compute the LLRs in even stages only, and a minimum of 4-bit simultaneous decoding at last stage. It also decodes extra special sub-codes of higher length in the same cycle. Additionally, this work proposes a partial-sum lookahead (PSL) architecture to further reduce the latency. The proposed PSL starts the computation of subsequent nodes without waiting for decoding of current information bits, and corresponding partial sums.

2.8 conclusion

Polar codes shares a high similarity with the Reed-Muller codes, both uses the same encoder except for the frozen bit selection where hamming distance is replaced by the Bhattacharyya parameter value. The performance of polar codes is shown to supersede that of Reed-Muller codes. Polar encoders have a very simple structure, combinational implementation fits perfectly the small codeword encoders while pipelined encoder is better for longer codewords, the matrix generation circuit in the pipelined encoder can be simplified in a recursive way. The SCL decoder is

more complex yet have a better error detection and correction performance than the ordinary SC decoder.

Chapter 3: Theory

3.1 Introduction

Polar coding technique makes use of N independent B-DMC channels to generate another N channels with the same total capacity, yet this capacity is shuffled and some of the generated channels have better reliability compared to other channels. Successive cancellation decoders are the first decoders used by E.Arikan to proof the polar codes error correction performance. Although having low complexity, SC suffers from high decoding latency because of its serial(successive) structure along with the very long codewords used. It is proven that the polar codes achieves the channel capacity at long codewords reaching infinity.
In this work the latency problem is tackled by the adoption of multi-bit decoding at the same clock cycle. This is done by the usage of radix-4 processing units along with a special last stage decision unit. Another techniques were added to this proposal to further decrease the overall latency such as: partial sum lookahead and special sub-codes handling.

3.2 Channel coding theory

In [21] Shannon proved that for any channel coding technique on a limited channel bandwidth, the maximum data rate is bounded by the channel capacity C which is achievable with some random code, yet Shannon didn't exhibit any exact formable sequence that will fulfill this capacity.
E.Arikan [3] introduced channel polarization technique that achieved channel capacity for binary discrete memoryless channels **B-DMC**.
Channel polarization targets creating a set of N channels from another N independent channels. The symmetric capacity of the created channels tends to 0 or 1 as N becomes large. this process can be divided into :

1. channel combining
2. channel splitting

3.2.1 Channel combining

Recursively combine copies of a B-DMC to produce $W_N : X^N -> Y^N$ where $N = 2^n$. At the 0^{th} level of recursion $W_1 = W$ At the 1^{st} level of recursion $n = 1$ two independent copies of W_1 are combined to obtain $W_2 : X^2 -> Y^2$
$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2)$
Figure 3.1 shows the combining of 2 W channels into another 2 channels where first one is less reliable than the second one, the explanation is since the second channel is meant to carry u_1, now after combining u_1 can be deducted from the output of both W_1 and W_2.

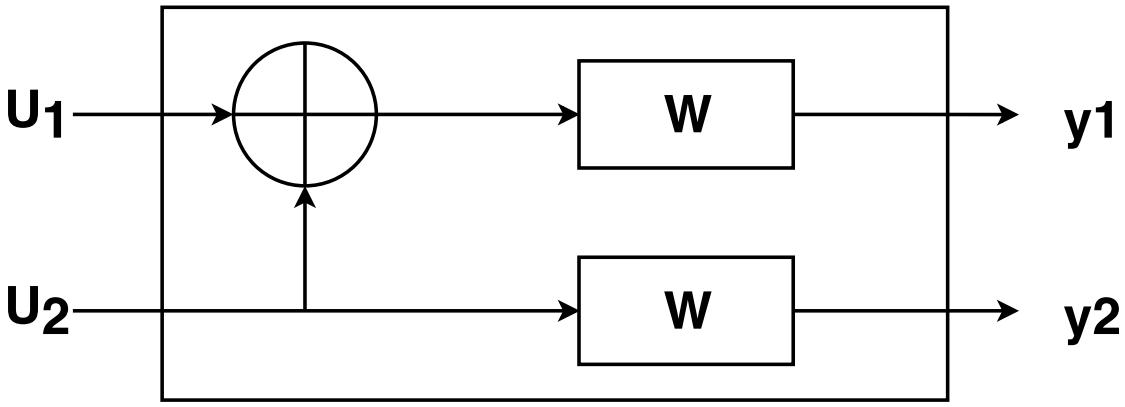


Figure 3.1: Combining 2 W_1 channels to obtain W_2

The next level of recursion combines two W_2 channel to create $W_4 : X^4 -> Y^4$
$W_4(y_1^4|u_1^4) = W_2(y_1^2|u_1 \oplus u_2, u_3 \oplus u_4)W_2(y_3^4|u_2, u_4)$

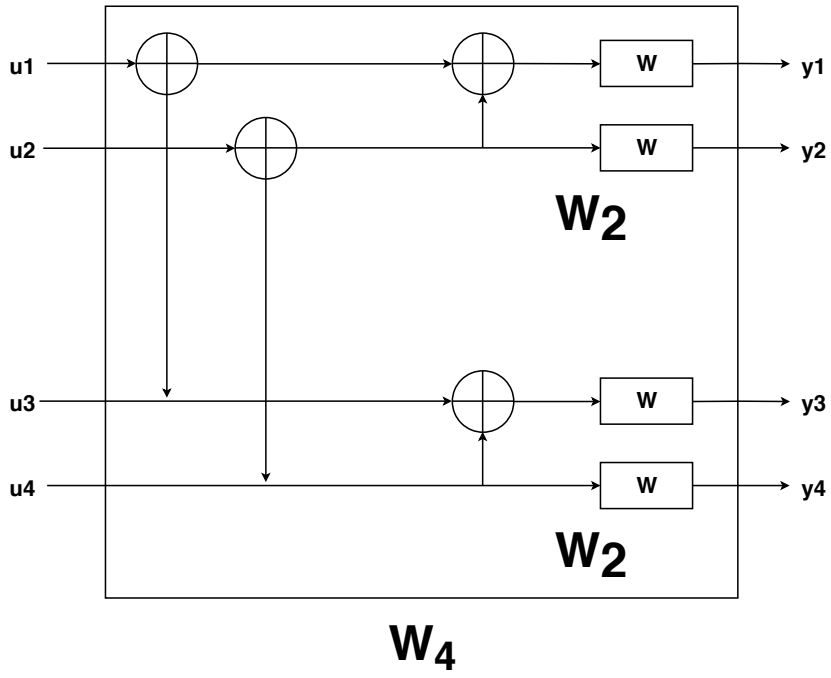


Figure 3.2: Combining 4 W_1 channels to obtain W_4 by using the generated W_2

Figure 3.2 shows the second level of channel combining , it can be seen that the first level of combining is then recombined together to generate 4 combined channels The relation between vector u_1^4 and x_1^4 can be written as

$X_1^4 = u_1^4 G_4 B_N$ with

$$G_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

G_4 can be described as the Kronecker power of the matrix as $G_4 = G^{\otimes 2}$ where

$$G = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

and B_N is a simple bit-reversal permutation matrix, for simplicity B_N is dropped from all upcoming studies.
G_N can be generalized as $G_N = G^{\otimes n}$.

3.2.2 16 bit encoder

The Generation matrix of a 16 bit channel combination G_{16} shall be represented as :

$$G_{16} = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}$$

The 16-bit channel combiner shall be implemented by and encoder as shown in figure 3.3, this encoder contains $\frac{16}{2} * log_2(16)$ XOR gates, the general form of encoder complexity is $\frac{N}{2} * log_2(N)$

3.2.3 Channel splitting

Being able to generate the polarized W_N channel set, these channels should be splitted into N channels with different transition probabilities.
The interesting property of the generated channels is that they are polarized in the sense that the reliability of a given channels is more or less than the original

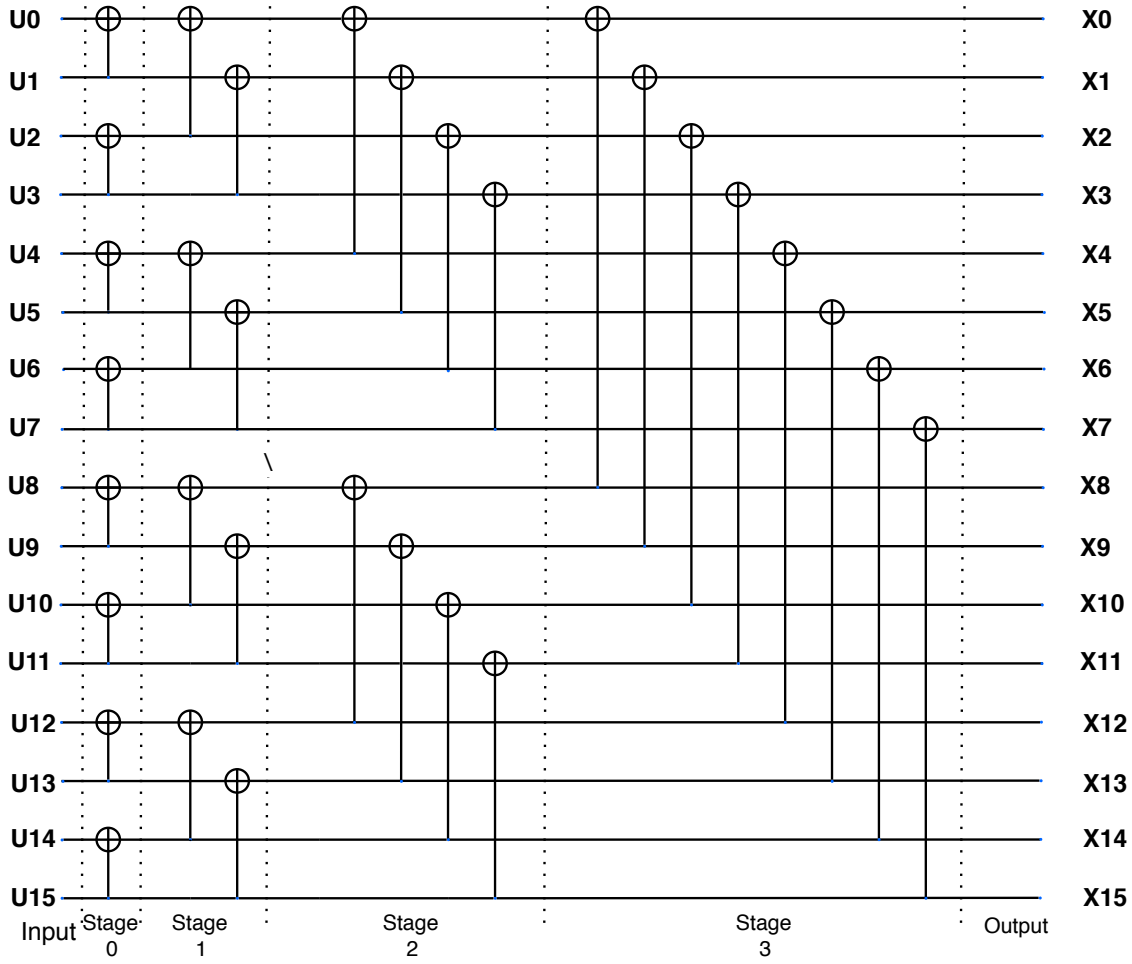


Figure 3.3: Combining 16 W_1 channels to obtain W_{16}

channel. Thus the reliable channels can be used for bit transmission and the non reliable channels shall be transmitting frozen bits on a value that is known by both encoder and decoder ,either "0" or "1" . In the upcoming work, all frozen channels are transmitting "0". The channels shall be divided to reliable and not reliable depending on the transition probabilities. As per [3] division can be carried out by calculating the Bhattacharyya parameter for the generated channels using the recursive form :

$Z(W_N^{2j-1}) = 2Z(W_{N/2}^j) - Z(W_{N/2}^j)^2$
$Z(W_N^{2j}) = Z(W_{N/2}^j)^2.$

The channels are sorted according to Bhattacharyya parameter. The count of channels that fulfill the desirable rate and have the least Bhattacharyya parameter will be used for transmitting data.
The encoder has a very simple nature. This simplicity enabled the generation of different encoders in a very non complex, and problem free implementations. Those implementations ranged from small area pipelined structure to pure combinational one cycle encoders. The design of a polar encoder has not attracted researches and optimizations like the polar decoders.
The polar decoders complexity and length depends on the codeword length which is huge. Although the successive cancellation decoder is pretty simple, it suffers from huge latency because of the large codewords used.

3.3 Successive cancellation decoders

In SC decoding, the decoded bits are estimated sequentially. The decoder is described by a flow tree [11] as shown in figure 3.4. There are 2 types of processing nodes in the flow tree which are f and g nodes, where the f function replaces the XOR gate in the encoder, and the G function resembles the check node in the encoder (where check node is the other input of the XOR gate).
The decoding of a bit $\hat{u}_i$ is performed by analyzing the Likelihood ratio $LR(y, \hat{u}_1^i - 1)$ if it is < 1 and the bit is not frozen, then the bit is decided "1" and is decided "0" otherwise. For sake of simplicity, Each likelihood ratio is going to be denoted $L(i, j)$ where i is the row count and j is the stage count. The input to decoder is at stage 0 and the LR of bit i before decision is $L(i, log_2(N))$.
As per the flow tree to decode the i_{th} bit $\hat{u}_i$, $L(i, log_2(N))$ shall be calculated,this needs LR at stage $log_2(N) - 1$ to be calculated and for this to get calculated LR at stage $log_2(N) - 2$ shall be calculated, this continues in successive pattern.

3.3.1 f,g equation derivation

As referred in [14], and by analyzing the 2-bit encoder in figure 3.5 it can be found that :

$$c = a \oplus b \tag{3.1}$$

$$d = b \tag{3.2}$$

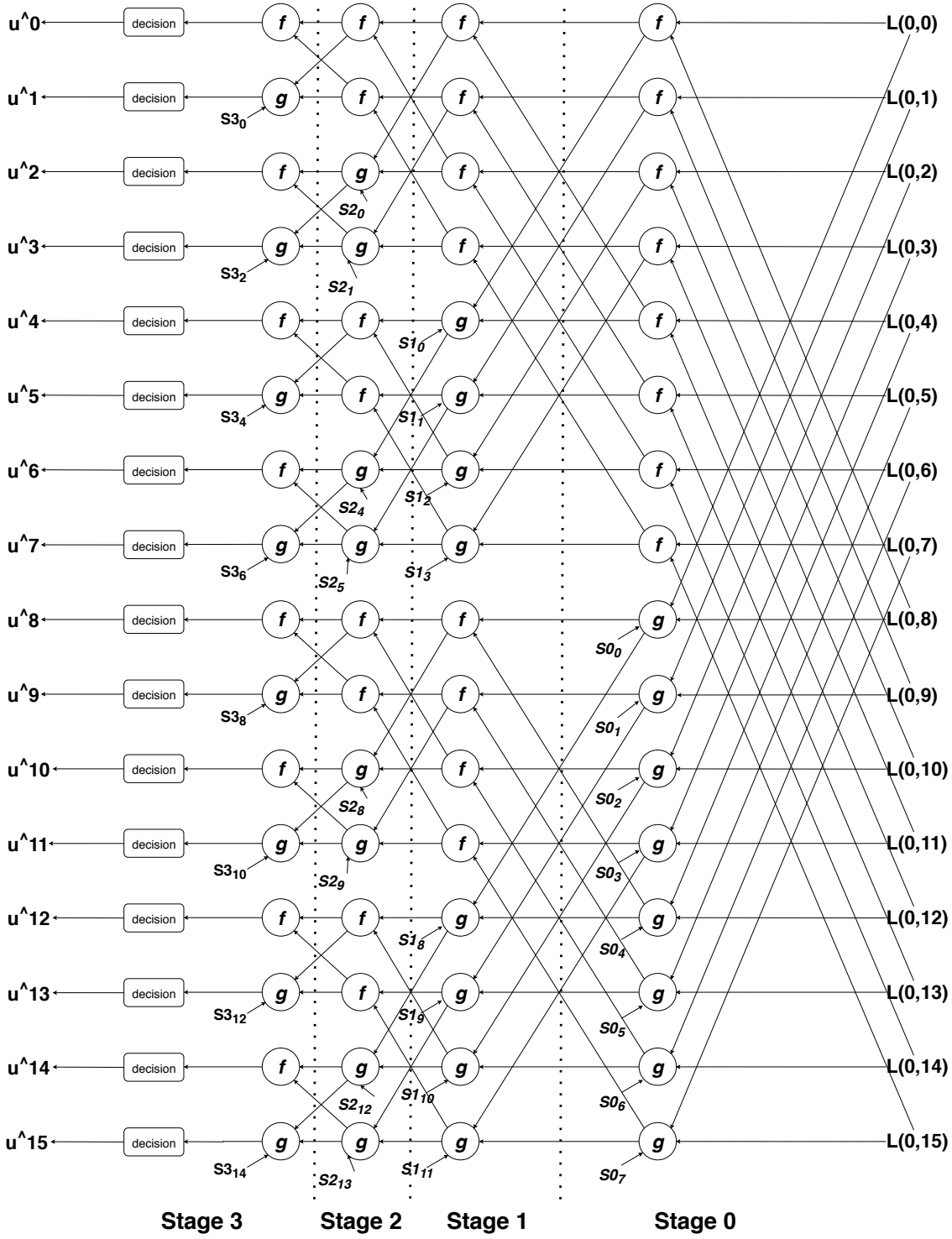


Figure 3.4: 16-bit successive cancellation decoder data flow graph

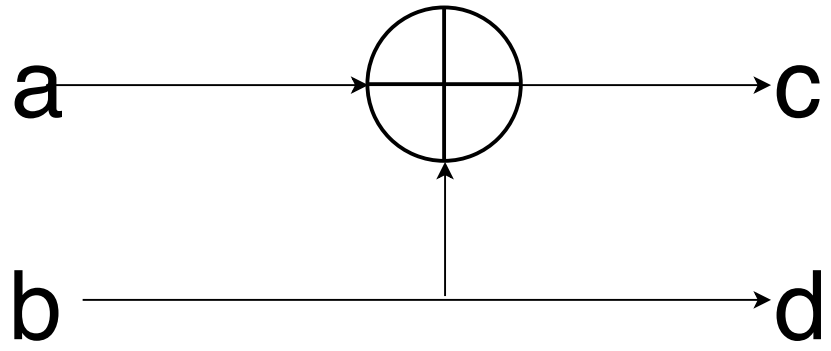


Figure 3.5: 2-bit Kronecker product

Equations 3.1,3.2 can be reformed in decoder case into:

$$\hat{a} = c \oplus d \tag{3.3}$$

$$\hat{b} = d \tag{3.4}$$

From equation 3.3,3.4 the probabilities of decoded bits can be described as:

$$P(\hat{a}=0) = P(c=0)P(d=0) + P(c=1)P(d=1) \tag{3.5}$$

$$P(\hat{a}=1) = P(c=0)P(d=1) + P(c=1)P(d=0) \tag{3.6}$$

$$P(\hat{b}=0,\hat{a}=0) = P(c=0)P(d=0) \tag{3.7}$$

$$P(\hat{b}=0,\hat{a}=1) = P(c=1)P(d=0) \tag{3.8}$$

$$P(\hat{b}=1,\hat{a}=0) = P(c=1)P(d=1) \tag{3.9}$$

$$P(\hat{b}=1,\hat{a}=1) = P(c=0)P(d=1) \tag{3.10}$$

Thus,

$$LR(\hat{a}) = \frac{P(\hat{a}=0)}{P(\hat{a}=1)} = \frac{P(c=0)P(d=0)+P(c=1)P(d=1)}{P(c=0)P(d=1)+P(c=1)P(d=0)} = \frac{LR(c)LR(d)+1}{LR(c)+LR(d)} \tag{3.11}$$

$$LR(\hat{b}_{a=0}) = \frac{P(\hat{b}=0,\hat{a}=0)}{P(\hat{b}=1,\hat{a}=0)} = \frac{P(c=0)P(d=0)}{P(c=1)P(d=1)} = LR(c)LR(d) \tag{3.12}$$

$$LR(\hat{b}_{a=1}) = \frac{P(\hat{b}=0,\hat{a}=1)}{P(\hat{b}=1,\hat{a}=1)} = \frac{P(c=1)P(d=0)}{P(c=0)P(d=1)} = \frac{LR(d)}{LR(c)} \tag{3.13}$$

Combining equations 3.12,3.13:

$$LR(\hat{b}) = LR(d)^{1-2\hat{a}}LR(c) \tag{3.14}$$

The $1-2\hat{a}$ form is to convert the $\hat{a}$ values from $0,1$ into $-1,1$ to decide whether a division or a multiplication is needed.
Equations 3.11 , 3.14 can be generalized as the f and g function as :

$$f(L_1,L_2) = \frac{L_1L_2+1}{L_1+L_2} \tag{3.15}$$

$$g(L_1, L_2, \hat{u}_s) = L_2^{1-2\hat{u}_s} L_1 \tag{3.16}$$

where $\hat{u}_s$ is the partial sum calculated from previously decoded bits.
The result of equation 3.14 can be generalized to show that i^{th} decoded bit $\hat{u}_i$ is estimated based on the previously decoded bits $\hat{u}_0, \hat{u}_1 \dots \hat{u}_{i-1}$.
The problem with equations 3.15, 3.16 is that they are complex to implement from hardware perspective because it contain multiplications and divisions. This complexity can be released by using the logarithms for both sides of the equations and calculate/use log likelihood ratios (LLR) instead of likelihood ratios. The f and g function can be calculated for likelihood ratios as [15]:

$$f(L_1, L_2) = 2tanh^{-1}(tanh(L_1/2)tanh(L_2/2)) \tag{3.17}$$

$$g(L_1, L_2, \hat{u}_s) = (-1)^{\hat{u}_s} L_1 + L_2 \tag{3.18}$$

The f equation seems very complex for implementation, yet the minimum sum approximation [15] may be used without any obvious performance decrease. The new f equation tends to :

$$|f(L_1, L_2)| \simeq min(|L_1|, |L_2|) \tag{3.19}$$

$$sgn(f(L_1, L_2)) = sgn(L_1) \oplus sgn(L_2) \tag{3.20}$$

3.4 Proposed Architecture

3.4.1 Overview

Successive cancellation decoding makes use of multiple f and g processing circuits (PUs). This Processing units implements the functions defined in (3.21) , (3.22). As shown in figure 3.6, the 4-bit decoder flow graph imitates the 4-bit encoder where f function replaces the first input of XOR gate in encoder while g function replaces the other input of the XOR gate. A 4-bit decoder contains $log_2(4)$ decoding stages.

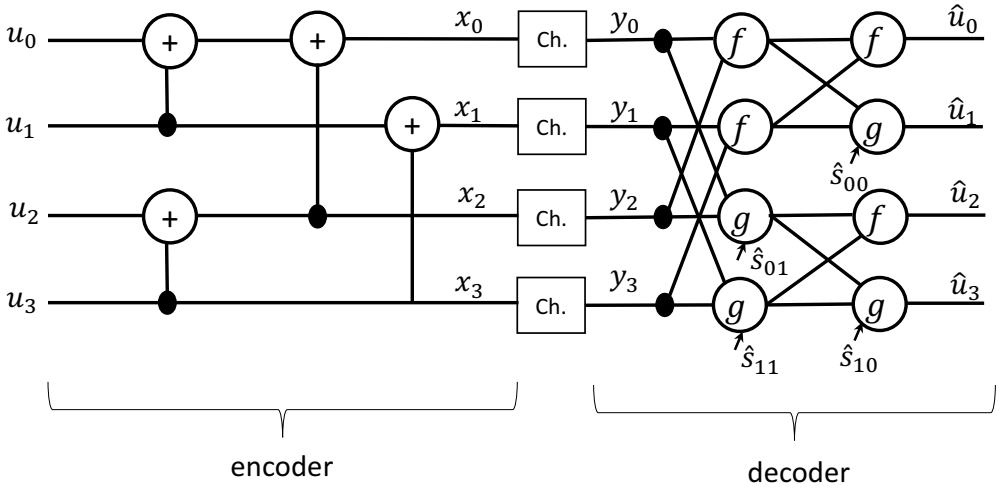


Figure 3.6: A 4-bit polar encoder / decoder

The successive cancellation decoder flow graph is divided into vertical stages. The first decoding stage is called stage 0, while the last stage (before bit decision) shall be called stage $log_2(n)-1$. A new combined processing circuit was proposed, this new circuit was able to use the LLRs at stage i to calculate the LLRs at stage $i+2$ thus skipping 2 stages at a time. Those LLRs are calculated by the use of the combined f and g functions in the form $(ff = f(f,f), fg = f(g,g), gf = g(f,f) gg = g(g,g))$. Compared to the old processing circuit, the combined PUs reduce the memory requirements since they are only storing the LLRs in even stages, and it also decreases the overall decoding latency.
A 16-bit decoder is shown in figure 3.7 using the new combined circuit. The radix-2 last stage unit will not be able to cope with the radix-4 architecture, A new hard decision unit called last stage processing unit (LSPU) is proposed where at least 4 bits can be decoded at the same clock.
For simplicity each processing unit is added to the figure 4 times once for every equation type ff, fg, gf, gg, but since those equations are never used in the same time, only 4 processing units are needed for figure 3.7 in the line architecture.
Section 3.4.3 presents the proposed unit, as depicted in Figure 3.9.

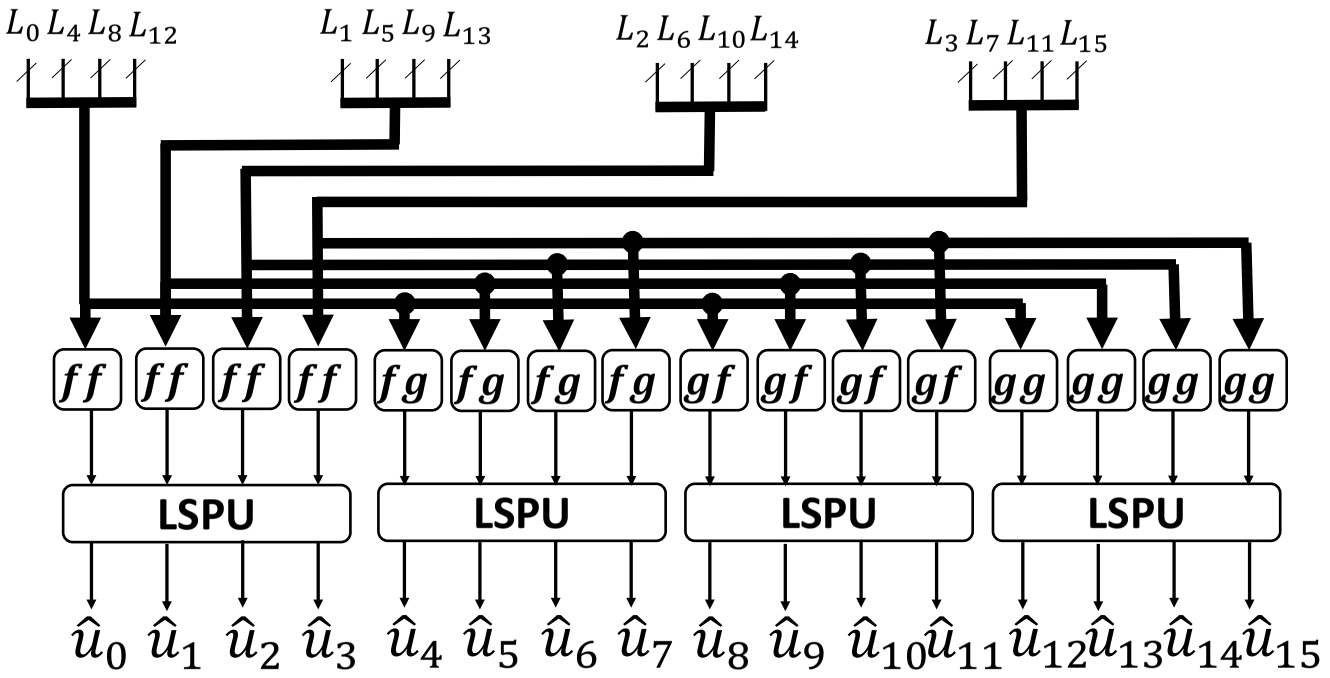


Figure 3.7: A radix-4 architecture to 16-bit SC decoder

3.4.2 Combined processing circuit

In [15], a simple form of f, g function are proposed as :

$$f(L_0, L_1) \approx sgn(L_0)sgn(L_1)min\{|L_0|, |L_1|\} \tag{3.21}$$

$$g(L_0, L_1, \hat{s}) = L_0(-1)^{\hat{s}} + L_1 \tag{3.22}$$

where L_0, L_1 are the log likelihood ratios while the effect of previously decoded bits appears in the form of the partial sum $\hat{s}$. By the use of the equations 3.21 and 3.22, the new radix-4 PU functions can be easily proven to be :

$$ff(L_0, L_1, L_2, L_3) = f(f(L_0, L_2), f(L_1, L_3)) \approx sgn(L_0)sgn(L_1)sgn(L_2)sgn(L_3). \min\{|L_0|, |L_1|, |L_2|, |L_3|\} \tag{3.23}$$

$$fg(L_0, L_1, L_2, L_3, \hat{s}_{00}) \approx sgn(L_1)sgn(L_3). \min\{|L_1|, |L_3|\} + (-1)^{\hat{s}_{00}} sgn(L_0)sgn(L_2). \min\{|L_0|, |L_2|\} \tag{3.24}$$

$$gf(L_0, L_1, L_2, L_3, \hat{s}_{01}, \hat{s}_{11}) \approx sgn(L_0(-1)^{\hat{s}_{01}} + L_2). sgn(L_1(-1)^{\hat{s}_{11}} + L_3). \min\left\{|sgn(L_0(-1)^{\hat{s}_{01}} + L_2)|, |L_1(-1)^{\hat{s}_{11}} + L_3|\right\} \tag{3.25}$$

$$gg(L_0, L_1, L_2, L_3, \hat{s}_{10}, \hat{s}_{01}, \hat{s}_{11}) = (-1)^{\hat{s}_{10}+\hat{s}_{01}} L_0 + (-1)^{\hat{s}_{11}} L_1 + (-1)^{\hat{s}_{10}} L_2 + L_3 \tag{3.26}$$

where $\hat{s}_{ij}$:i is the partial sum count and j is the decoding stage count.

The complexity of the radix-4 PU is higher than the regular radix-2 PU, since the regular PU only computes f and g, the new circuit area is a less smaller than twice the radix-2 processing unit area, yet the needed count of PUs in both the tree or line architectures is reduced to half maintaining approximately the same processing units areas. So the new architecture reduces latency in a smaller area.

3.4.3 Last stage processing unit

The Last stage processing unit(LSPU) block makes use of the 4 LLR inputs called L_0, L_1, L_2, and L_3 respectively, and the four binary inputs denoted by $frozen_0$, $frozen_1$, $frozen_2$, and $frozen_3$. The frozen pattern $frozen_i$ shows whether each bit u_i is frozen or not.
The LSPU has 16 different frozen bits patterns, so it is a must to study each of the 16 cases of the frozen bit patterns and have a closed decoding form. Yet by analyzing the Bhattacharyya parameter and the recursive equations in [3]:
$Z(W_N^{2j-1}) = 2Z(W_{N/2}^j) - Z(W_{N/2}^j)^2$
$Z(W_N^{2j}) = Z(W_{N/2}^j)^2$

Since $Z(W)$ takes a value in

$$0, 1$$

then $Z(W) > Z^2(W)$
Thus, $2Z(W_{N/2}^j) - Z(W_{N/2}^j)^2 > Z(W_{N/2}^j)^2$

This proves that for each 2 consecutive channels, the channel $2j-1$ is always less reliable than $2j$, there is no actual pattern where the bit $2j-1$ is used and $2j$ is frozen.
This reduces the available 16 different frozen bit patterns into just 6 bits that should be studied. The LSPU is able of computing the four bits $\hat{u}_0$, $\hat{u}_1$, $\hat{u}_2$, and $\hat{u}_3$ simultaneously. This processing unit makes use of equations 3.23-3.26 to calculate LLRs and perform hard decision to decode 4 bits simultaneously in the same cycle.

3.4.3.1 Case 1: All bits are frozen

All bits are frozen , the decoded values are set to "0".

3.4.3.2 case 2: All bits are frozen, u_3 is used

All bits are frozen except u_3, thus the decoded values are set to "0" except u_3.
Getting back to equation 3.26
$LLR(u_3) = (-1)^{\hat{s}_{10}+\hat{s}_{01}} L_0 + (-1)^{\hat{s}_{11}} L_1 + (-1)^{\hat{s}_{10}} L_2 + L_3$ $LLR(u_3) = L_0 + L_1 + L_2 + L_3$
$u_3 = SgnL_0.L_1.L_2.L_3 = s_{0123}$

3.4.3.3 case 3: u_0, u_1 are frozen, u_2, u_3 are not frozen

Getting back to equations 3.25,3.26
$LLR(u_2) \approx sgn(L_0(-1)^{\hat{s}_{01}} + L_2)$.
$sgn(L_1(-1)^{\hat{s}_{11}} + L_3)$.
$\min\left\{|sgn(L_0(-1)^{\hat{s}_{01}} + L_2)|, |L_1(-1)^{\hat{s}_{11}} + L_3|\right\}$ $LLR(u_2) \approx sgn(L_0+L_2).sgn(L_1+L_3)$.
$u_2 = XOR(s_{02}, s_{13})$
$u_3 = NOT(s_{13})$

3.4.3.4 case 4: u_0, u_2 are frozen, u_1, u_3 are not frozen

Getting back to equations 3.24,3.26
$u_1 = XOR(s_{01}, s_{23})$
$u_3 = NOT(s_{23})$

3.4.3.5 case 5: bit u_0 is only frozen.

Starting with equations 3.23-3.26 knowing that $u_0 = 0$:

$$LLR(u_1) = fg(L_0, L_1, L_2, L_3, \hat{s}_0 0 = 0) \tag{3.27}$$

$$LLR(u_1) \approx sgn(L_1)sgn(L_3).\min\{|L_1|, |L_3|\} + sgn(L_0)sgn(L_2).\min\{|L_0|, |L_2|\} \tag{3.28}$$

$if(sgn(L_1)sgn(L_3).\min\{|L_1|, |L_3|\} + sgn(L_0)sgn(L_2).\min\{|L_0|, |L_2|\} < 0)$
$u_1 = 1$
else
$u_1 = 0$

$$u_1 = sgn(sgn(L_1)sgn(L_3).\min\{|L_1|, |L_3|\} + sgn(L_0)sgn(L_2).\min\{|L_0|, |L_2|\}) \tag{3.29}$$

$if(\min\{|L_1|,|L_3|\} < \min\{|L_0|,|L_2|\})$
$u_1 = sgn(L_0)sgn(L_2)$
else
$u_1 = sgn(L_1)sgn(L_3)$

$$LLR(u_2) = gf(L_0, L_1, L_2, L_3, \hat{s}_0 1 = u_1, \hat{s}_1 1 = u_1) \tag{3.30}$$

$LLR(u_2) \approx sgn(L_0(-1)^{u_1} + L_2)$.
$sgn(L_1(-1)^{u_1} + L_3)$.
$\min\left\{|sgn(L_0(-1)^{\hat{s}_{01}} + L_2)|, |L_1(-1)^{\hat{s}_{11}} + L_3|\right\}$

$$u_2 = sgn(L_0(-1)^{u_1} + L_2).sgn(L_1(-1)^{u_1} + L_3) \tag{3.31}$$

$if(\min\{|L_1|,|L_3|\} < \min\{|L_0|,|L_2|\})$
$u_2 = sgn(L_0 sgn(L_0)sgn(L_2) + L_2).sgn(L_1 sgn(L_0)sgn(L_2) + L_3)$
$u_2 = sgn(|L_0|sgn(L_2) + L_2).sgn(L_1 sgn(L_0)sgn(L_2) + L_3)$
$u_2 = sgn(L_2).sgn(L_1 sgn(L_0)sgn(L_2) + L_3)$
$if(|L_1| < |L_3|)$
$u_2 = s_2.s_3$
else
$u_2 = s_0.s_1$
else
$u_2 = sgn(L_0 sgn(L_1)sgn(L_3) + L_2).sgn(L_1 sgn(L_1)sgn(L_3) + L_3)$
$u_2 = sgn(L_0 sgn(L_1)sgn(L_3) + L_2).sgn(|L_1|sgn(L_3) + L_3)$
$u_2 = sgn(L_0 sgn(L_1)sgn(L_3) + L_2).s_3$
$if(|L_0| < |L_2|)$
$u_2 = s_2.s_3$
else
$u_2 = s_0.s_1$
$LLR(u_3) = gg(L_0, L_1, L_2, L_3, \hat{s}_{10} = u_2, \hat{s}_{01} = u_1, \hat{s}_{11} = u_1)$
$= (-1)^{u_2+u_1} L_0 + (-1)^{u_1} L_1 + (-1)^{u_2} L_2 + L_3$
$if(\min\{|L_1|,|L_3|\} < \min\{|L_0|,|L_2|\})$
$u_1 = sgn(L_0)sgn(L_2)$
$if(|L_1| < |L_3|)$
$u_3 = sgn(L_3)$

else
$u_3 = s_0.s_1.s_2$
else $if(\min\{|L_1|,|L_3|\} > \min\{|L_0|,|L_2|\})$
$u_3 = s_3$

3.4.3.6 case 6: No frozen bits

the decoder is just encoding s_0, s_1, s_2, s_3
$u_0 = XOR(s_0, s_1, s_2, s_3)$

$u_1 = XOR(s_1, s_3)$
$u_2 = XOR(s_2, s_3)$
$u_3 = s_3$

3.4.4 Hard decision unit implementation

Figure 3.8 and algorithm 1 illustrates the Hard decision LSPU function. All the 6 frozen and non frozen pattern scenarios are handled with cases 1 though 6 while the other scenarios are omitted for being impossible. The 2^4 available patterns are not all possible because of the Bhattacharyya parameter properties, so only the valid patterns of frozen bits are implemented. The notations s_i, s_{ij}, and s_{ijkl} represent the sign bits of $L_i, \{L_i + L_j\}$, and $\{L_i + L_j + L_k + L_l\}$ respectively. The LLRs are implemented using the 2's complement to have easy separability of the sign bit while easing the operation on these arguments. The corresponding hardware architecture of LSPU is depicted in Figure 3.9.

3.4.5 Special sub-codes decoding

The work in [1] simplified the successive cancellation decoding by adding special decoding techniques for rate zero and rate one nodes, this work proposes extension to these special sub-codes to allow decoding more sequences instantly. The key idea is that having frozen bits duplicates some encoded bits;i.e $x_m = x_n = u_i + u_j$. This consequently adds redundancy in LLRs, and assists in decoding by averaging both LLR_m and LLR_n at the channel output. Therefore, $LLR = LLR_i + LLR_m$ is used in decoding u_i and u_j.
To be able to generate a decoding form, special sub-code length is assumed to be 8 and then will be generalized to general length k.
Yet in this work, Special sub-codes of maximum length of 16-bit are only handled as the targeted codeword length is 1024 and longer length sub-codes will be rare in this limited codeword. This length is optimum as it can be decoded in the same cycle while still being able to improve decoding latency.
Figure 3.8 can be extended for any node where all the k child bits fit in one of the following criteria:

3.4.5.1 Criterion 1: All bits are frozen

All the k child bits are frozen, these child bits are decoded simultaneously to value of zero.

3.4.5.2 Criterion 2: All bits are frozen except last bit

All the k child bits are frozen except for k.
Assuming 8-bit encoder with this frozen pattern:
$x_0 = u_7$
$x_1 = u_7$

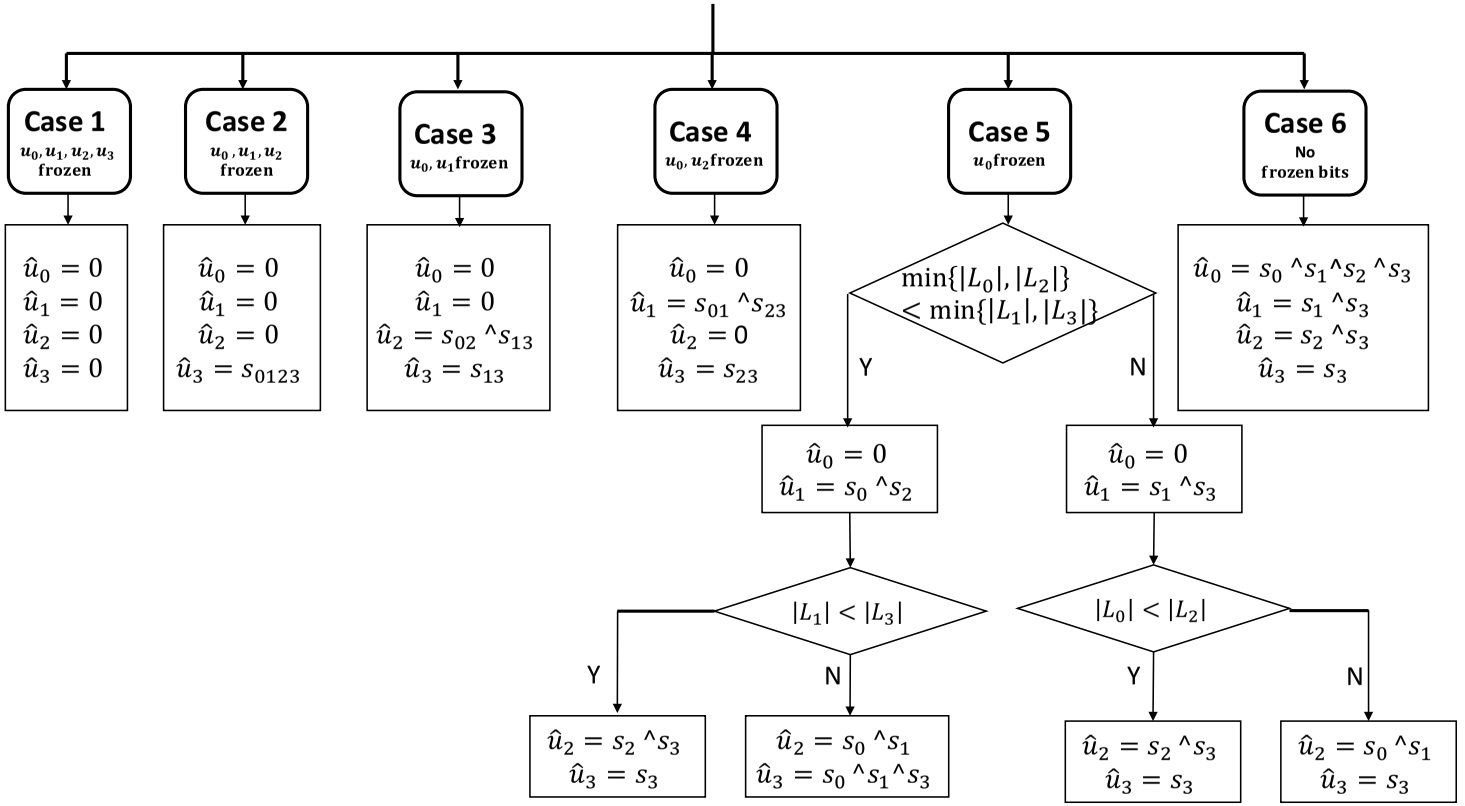


Figure 3.8: Flowchart explaining the radix-4 last stage algorithm

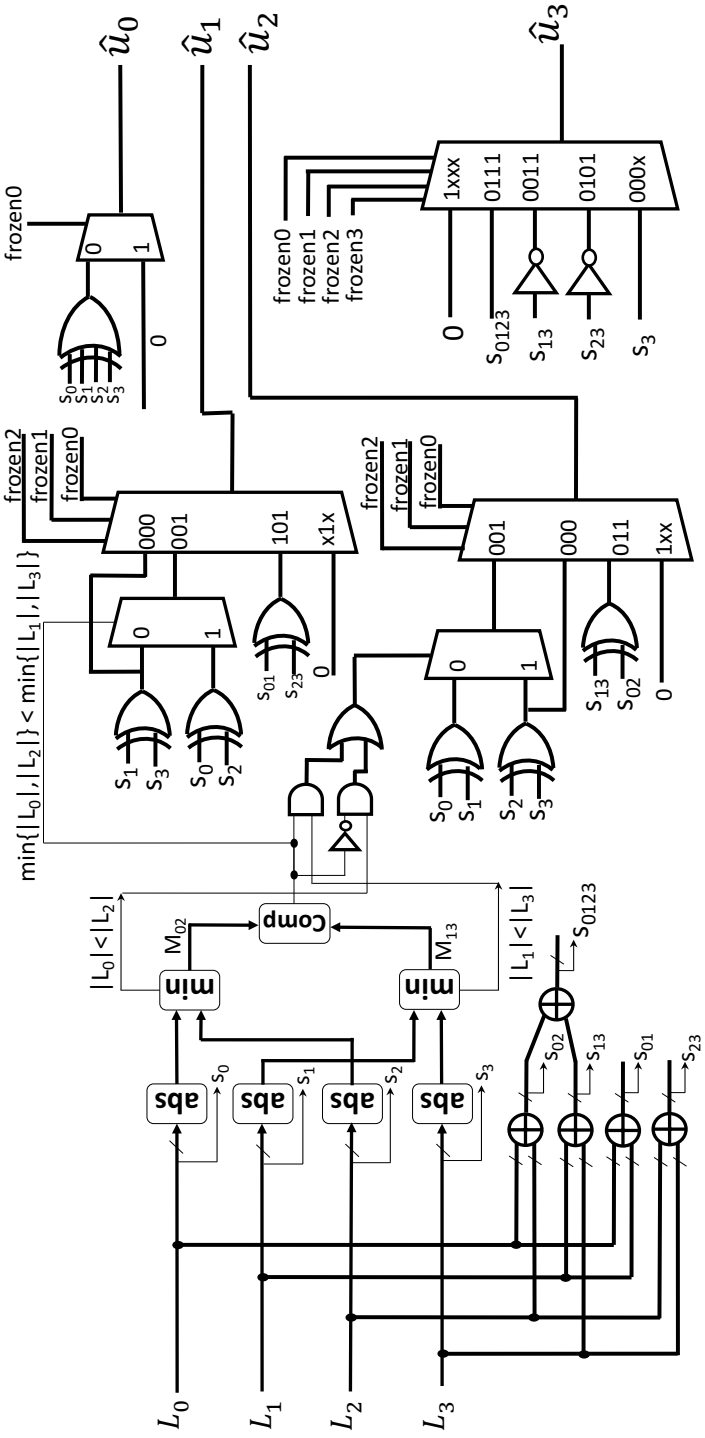


Figure 3.9: Last stage radix-4 processing unit

Algorithm 1: Radix-4 last stage processing unit

Case 1: All bits are frozen (rate zero)
$\hat{u}_0 = 0, \hat{u}_1 = 0, \hat{u}_2 = 0, \hat{u}_3 = 0$
Case 2: u_0, u_1, u_2 are frozen
$x_0 = u_3, x_1 = u_3, x_2 = u_3, x_3 = u_3$
$\hat{u}_0 = 0, \hat{u}_1 = 0, \hat{u}_2 = 0, \hat{u}_3 = s_{0123}$
Case 3: u_0, u_1 are frozen
$\hat{u}_0 = 0, \hat{u}_1 = 0, \hat{u}_2 = \text{XOR}\ (s_{02}, s_{13})$,
$\hat{u}_3 = (s_{13})$
Case 4: u_0, u_2 are frozen
$\hat{u}_0 = 0, \hat{u}_2 = 0, \hat{u}_1 = \text{XOR}\ (s_{01}, s_{23})$,
$\hat{u}_3 = (s_{23})$
Case 5: u_0 is frozen
$\hat{u}_0 = 0$,
if $\min\{|L_0|, |L_2|\} < \min\{|L_1|, |L_3|\}$ **then**
  $\hat{u}_1 = \text{XOR}\ (s_0, s_2)$
  if $|L_1| < |L_3|$ **then**
    $\hat{u}_2 = \text{XOR}\ (s_2, s_3), \hat{u}_3 = s_3$
  else
    $\hat{u}_2 = \text{XOR}\ (s_0, s_1), \hat{u}_3 = \text{XOR}\ (s_0, s_1, s_2)$
  end if
else
  $\hat{u}_1 = \text{XOR}\ (s_1, s_3), \hat{u}_3 = s_3$
  if $|L_0| < |L_2|$ **then**
    $\hat{u}_2 = \text{XOR}\ (s_2, s_3)$
  else
    $\hat{u}_2 = \text{XOR}\ (s_0, s_1)$
  end if
end if
Case 6: No frozen bits (rate one)
$\hat{u}_0 = \text{XOR}\ (s_0, s_1, s_2, s_3)$
$\hat{u}_1 = \text{XOR}\ (s_1, s_3), \hat{u}_2 = \text{XOR}\ (s_2, s_3), \hat{u}_3 = s_3$

$x_2 = u_7$
$x_3 = u_7$
$x_4 = u_7$
$x_5 = u_7$
$x_6 = u_7$
$x_7 = u_7$
It is obvious that all the LLRs at the receiver will be carrying the same information, thus the decoding will be:

$$LLR_{u_{k-1}} = \frac{1}{k} * L_{0:k-1} \tag{3.32}$$

where $L_{0:k-1} = \sum_{j=0}^{k-1} L_j$
it can be deducted that:

$$\hat{u}_{k-1} = S_{0:k-1} \tag{3.33}$$

where $S_{0:k-1} = sign(L_{0:k-1})$ and the frozen bits are decoded simultaneously to value of zero.

3.4.5.3 Criterion 3: All bits are frozen except middle and last bits

k bits are frozen except for $k-1$ and $k/2-1$.
Assuming 8-bit encoder with this frozen pattern:
$x_0 = u_3 \oplus u_7$
$x_1 = u_3 \oplus u_7$
$x_2 = u_3 \oplus u_7$
$x_3 = u_3 \oplus u_7$
$x_4 = u_7$
$x_5 = u_7$
$x_6 = u_7$
$x_7 = u_7$
The last equations shows that $x_{0:3}$ are the same and $x_{4:7}$ are also the same, Thus
$\hat{u}_{k-1} = S_{k/2:k}$
$\hat{u}_{k/2-1} = S_{0:k/2-1} + \hat{u}_{k-1}$
and the frozen bits are decoded simultaneously to value of zero.

3.4.5.4 Criterion 4: All bits are frozen except the quarters of the sub code

k bits are frozen except for $k/4-1$, $k/2-1$, $3k/2-1$ and , $k-1$.
Assuming 8-bit encoder with this frozen pattern:
$x_0 = u_1 \oplus u_3 \oplus u_5 \oplus u_7$
$x_1 = u_1 \oplus u_3 \oplus u_5 \oplus u_7$
$x_2 = u_3 \oplus u_7$
$x_3 = u_3 \oplus u_7$
$x_4 = u_5 \oplus u_7$
$x_5 = u_5 \oplus u_7$
$x_6 = u_7$
$x_7 = u_7$
the frozen bits are decoded simultaneously to value of zero while $\hat{u}_{k-1} = S_{3k/4-1:k-1}$

$\hat{u}_{3k/4-1} = S_{k/2-1:3k/4-1} + \hat{u}_{k-1}$
$\hat{u}_{k/2-1} = S_{k/4-1:k/2-1} + \hat{u}_{k-1} + \hat{u}_{3k/4-1}$
$\hat{u}_{k/4-1} = S_{0:k/4-1} + \hat{u}_{k-1} + \hat{u}_{3k/4-1} + \hat{u}_{k/2-1}$

3.4.5.5 Criterion 5: All bits are frozen except the last 2 bits

k bits are frozen except for $k-1$ and $k-2$.
Assuming 8-bit encoder with this frozen pattern:
$x_0 = u_6 \oplus u_7$
$x_1 = u_7$
$x_2 = u_6 \oplus u_7$
$x_3 = u_7$
$x_4 = u_6 \oplus u_7$
$x_5 = u_7$
$x_6 = u_6 \oplus u_7$
$x_7 = u_7$
the frozen bits are decoded simultaneously to value of zero while $\hat{u}_{k-1} = S_{0:k-1_{odd}}$
$\hat{u}_{k-2} = S_{0:k-1_{even}} + \hat{u}_{k-1}$.

3.4.5.6 Criterion 6: All bits are frozen except the last 4 bits

k bits are frozen except for $k-1$, $k-2$, $k-3$ and $k-4$.
$x_0 = u_4 \oplus u_5 \oplus u_6 \oplus u_7$
$x_1 = u_5 \oplus u_7$
$x_2 = u_6 \oplus u_7$
$x_3 = u_7$
$x_4 = u_4 \oplus u_5 \oplus u_6 \oplus u_7$
$x_5 = u_5 \oplus u_7$
$x_6 = u_6 \oplus u_7$
$x_7 = u_7$
the frozen bits are decoded simultaneously to value of zero while
$\hat{u}_{k-1} = S_{\sum L_{4*j+3}}$
$\hat{u}_{k-2} = S_{\sum L_{4*j+2}} + \hat{u}_{k-1}$
$\hat{u}_{k-3} = S_{\sum L_{4*j+1}} + \hat{u}_{k-1} + \hat{u}_{k-2}$
$\hat{u}_{k-4} = S_{\sum L_{4*j}} + \hat{u}_{k-1} + \hat{u}_{k-2} + \hat{u}_{k-3}$
where j ranges from 0 to $k/4-1$

3.4.5.7 Criterion 7: No frozen bits

no bits are frozen then the sign of each LLR is assumed to be input bit to a polar encoder of size k , the encoder output $\hat{x}_{0:k}$ are the needed $\hat{u}_{0:k}$. $\hat{u}_0 = s_0 \oplus s_1 \oplus s_2 \oplus s_3 \oplus s_4 \oplus s_5 \oplus s_6 \oplus s_7$
$\hat{u}_1 = s_1 \oplus s_3 \oplus s_5 \oplus s_7$
$\hat{u}_2 = s_2 \oplus s_3 \oplus s_6 \oplus s_7$
$\hat{u}_3 = s_3 \oplus s_7$
$\hat{u}_4 = s_4 \oplus s_5 \oplus s_6 \oplus s_7$
$\hat{u}_5 = s_5 \oplus s_7$

$\hat{u}_6 = s_6 \oplus s_7$
$\hat{u}_7 = s_7$

3.4.6 Partial sum calculation

By regarding the 4-bit encoder and decoder flow graph 3.6, the f function in decoder replaces the XOR symbol in encoder while the g function replaces the dot in encoder, so it is obvious that the g function depends on the bit value on the input of the XOR (thus partially re-encoding the decoded bits) The partial sum generation matrix introduced in [9] calculates the updated partial sums after decoding one new bit$\hat{u}_i$ in one clock cycle. Figure 3.10 shows a partial sum calculation matrix based on a generation matrix as the one used for encoding along with the shift register and XOR matrix, this XOR matrix differs from that of the encoder matrix as shown. The shift register is meant to feed the recently decoded bit u_i every cycle. This Partial sum calculation can further be reduced by using the generation matrix simplification [9] as shown in figure 3.11 and the general form can be as in figure 3.12.

The main problem is that the radix-4 proposal decodes 4 bits simultaneously,while this partial sum network is limited to calculate partial sums after only on decoded bit in one cycle

The partial sum generation matrix is tweaked to update 4 bits at a time depending on the fact that:

$G^{\otimes N} = G^{\otimes N-4} \otimes G^{\otimes 4}$

where

$G = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $G^{\otimes 2} = G \otimes G$

Thus every 1 is replaced with $\begin{bmatrix} \hat{u}_i + \hat{u}_{i+1} + \hat{u}_{i+2} + \hat{u}_{i+3} & \hat{u}_{i+1} + \hat{u}_{i+3} & \hat{u}_{i+2} + \hat{u}_{i+3} & \hat{u}_{i+3} \end{bmatrix}$ where those bits are the results of the 4-bit encoding of the newly decoded bits

And every 0 is replaced with $\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$

The resulting word is XORed with the past partial sum register creating the current partial sum register. This implementation is described in figure 3.13. When compared to figure 3.12 the difference is only the removal of the ordinary shift register and replace it with a circuit capable of feeding the combined versions of the last decoded 4-bits $u_i, u_{i+1}, u_{i+2}, u_{i+3}$ This approach can be further tweaked to be able to update partial sum after decoding 16 bits at a time in case of special sub-code handling.

3.4.7 Partial sum lookahead

To decrease the overall decoder area ,the line architecture [30] is used over the tree architecture, Only the PUs needed to calculate the LLRs in the most crowded clock cycles (the stage 2 ff LLRs for example) are implemented, thus only the PUs that can be utilized in at least one cycle are instantiated. For n-bit codeword decoder, only $n/4$ combined processing units are instantiated.

The dependency of LLRs calculations on the previously decoded bits through the partial sum usage is what gives the successive cancellation decoder its successive

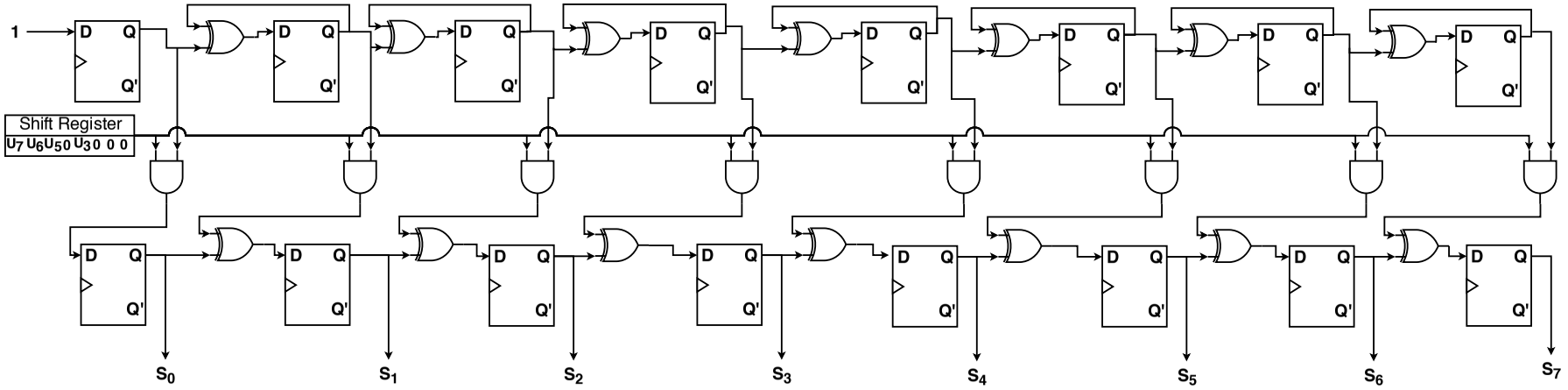


Figure 3.10: Pipelined partial sum calculation circuit for 8-bit successive cancellation decoder

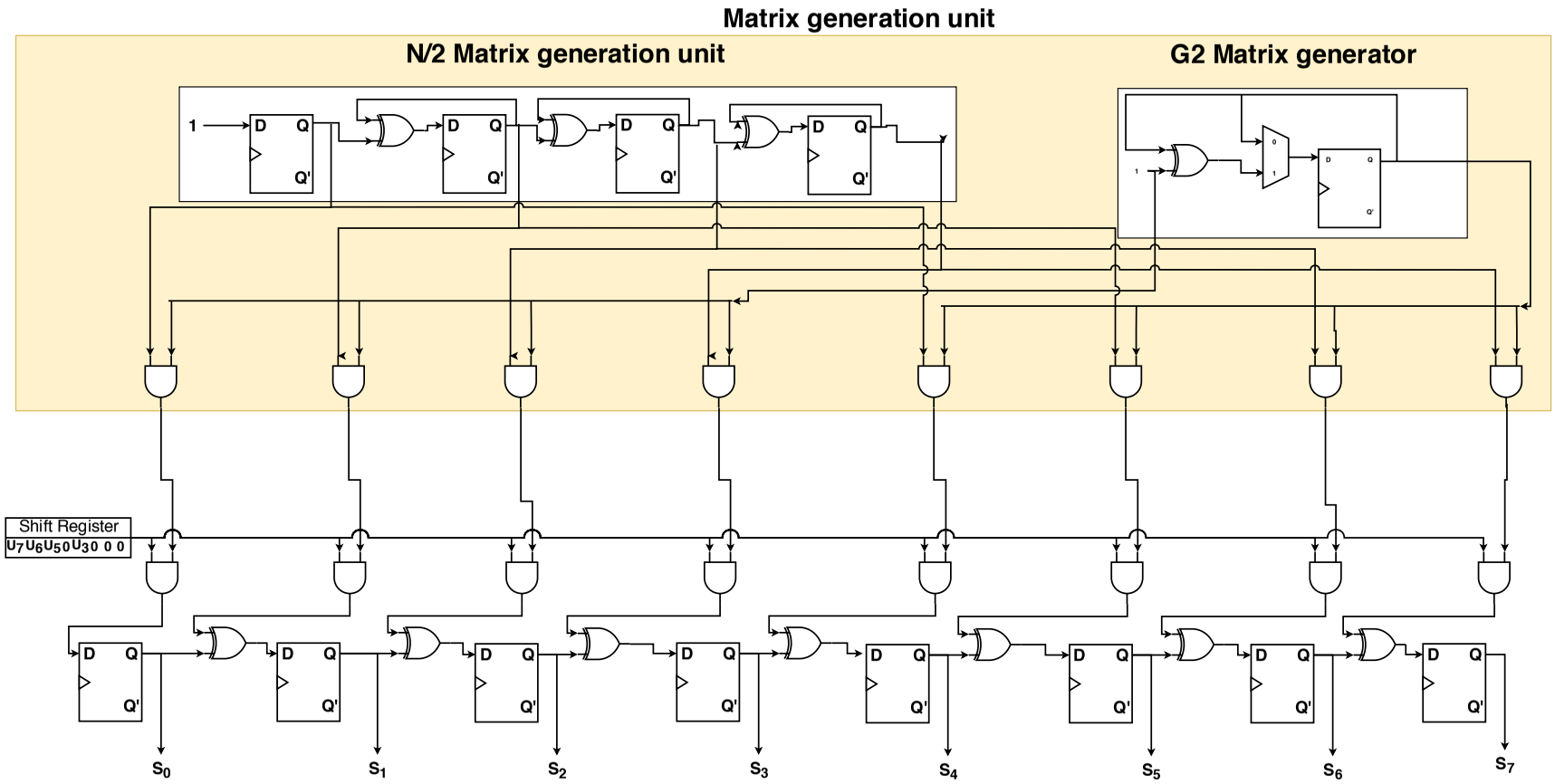


Figure 3.11: Pipelined partial sum calculation circuit for 8-bit successive cancellation decoder using G_4 generator

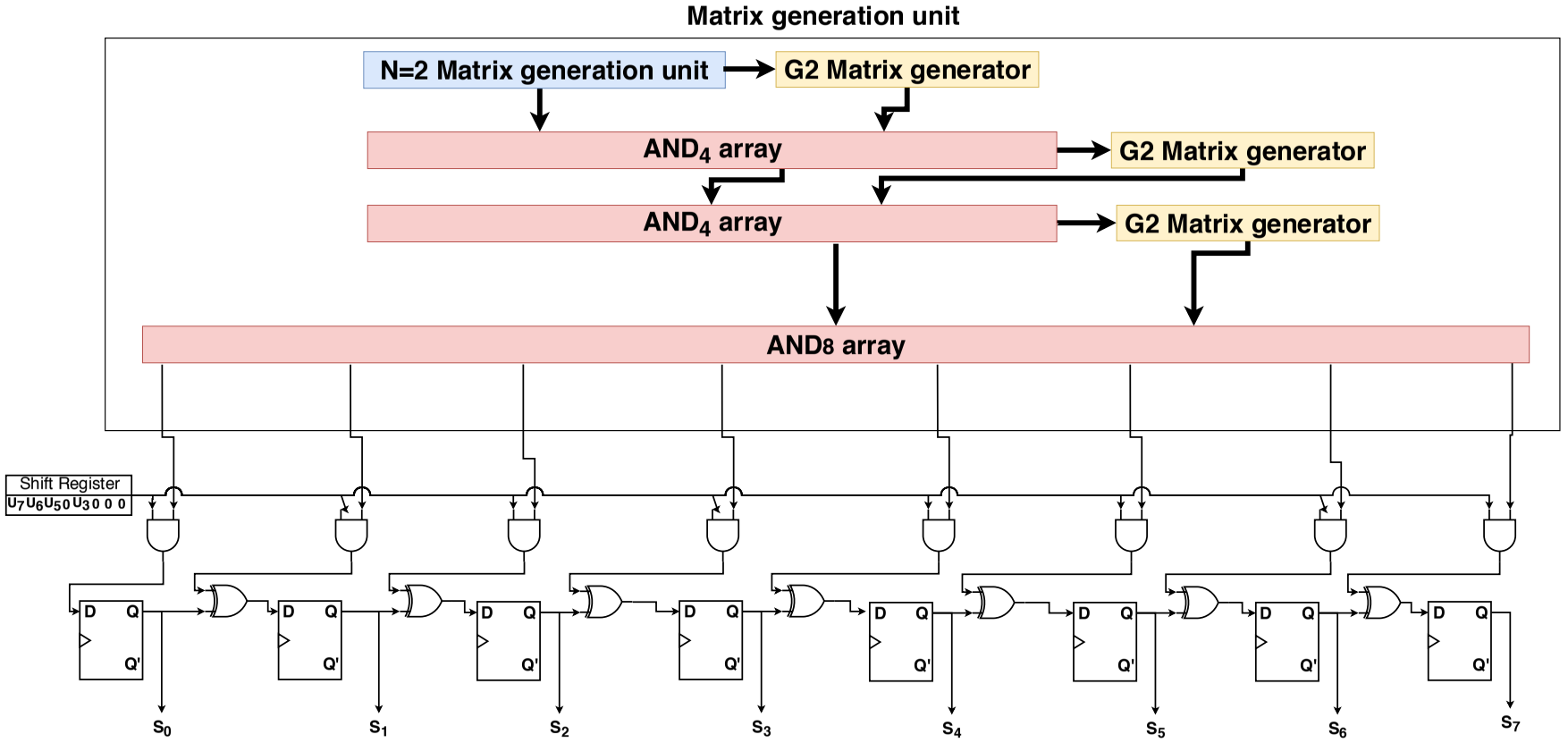


Figure 3.12: Pipelined partial sum calculation circuit for 8-bit successive cancellation decoder using the general form generation matrix

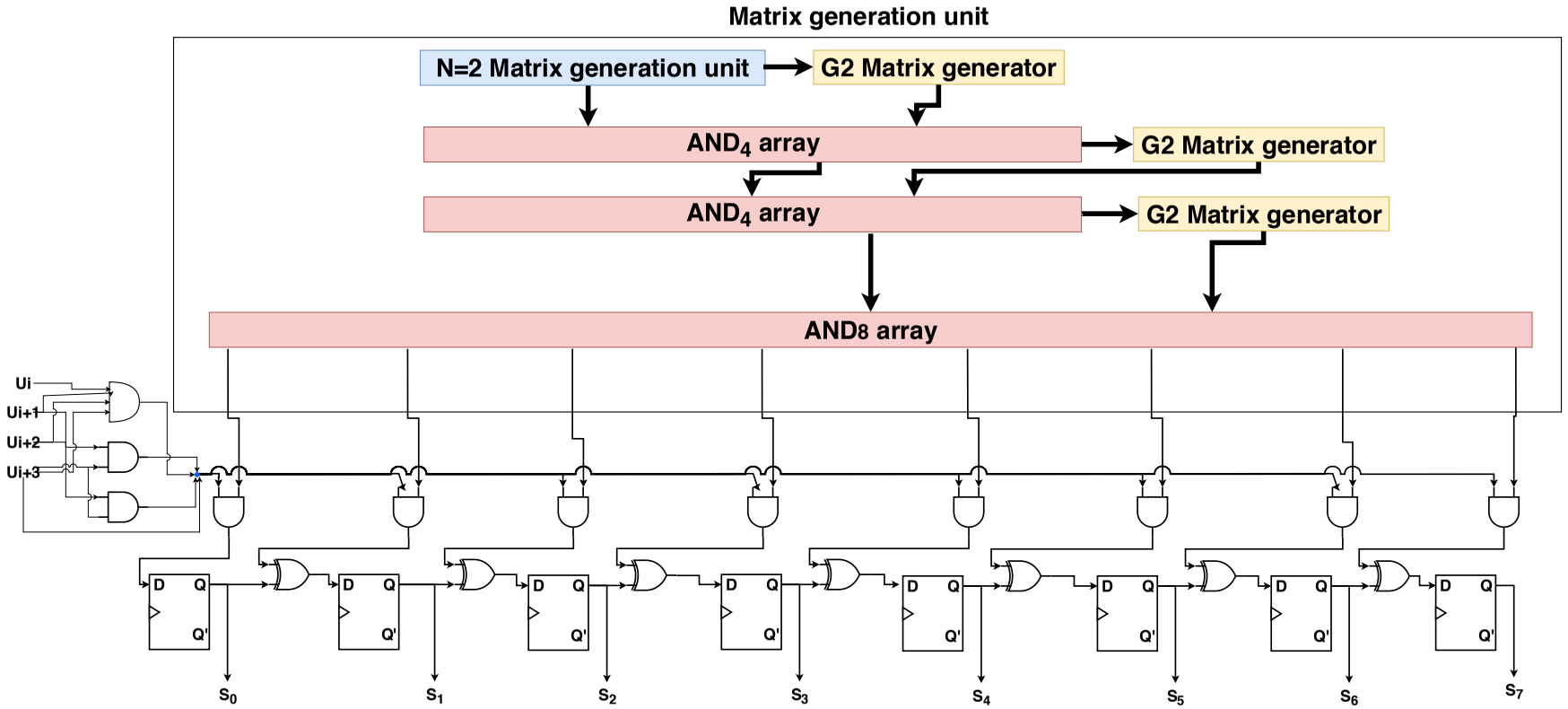


Figure 3.13: 4-bit at a cycle pipelined partial sum calculation circuit for 8-bit successive cancellation decoder using the general form generation matrix

structure, thus the instantiated processing units may remain idle for most of the time due to this successive structure. A bottle neck in the decoding process appears because of this wait on the newly calculated partial sums. And since the partial sums are updated every cycle by the newly decoded 4-bits only, this gives 2^4 possible values for the decoded bits to be. To break the bottleneck, the partial sum lookahead technique is proposed. Thus reducing the decoding latency of ordinary successive cancellation decoder. The partial sum lookahead idea imitates the carry lookahead in digital subtracter circuits that may be used in the decoding process. Instead of waiting for the hard decision of the bits $u_i, u_{i+1}, u_{i+2}, u_{i+3}$ to update the partial sum values and then calculate the LLRs, which will be leading to the decoding of the next 4 bits $u_{i+4}, u_{i+5}, u_{i+6}, u_{i+7}$, the 16 possible values of the bits $u_i, u_{i+1}, u_{i+2}, u_{i+3}$ are used to calculate partial sums and are fed into 16 different idle processing units, when the bits are decoded the LLRs in stage $log_2(n) - 2$ will be ready through a multiplexer. Thus the bits $u_{i+4}, u_{i+5}, u_{i+6}, u_{i+7}$ can be hard decided in the next cycle.

It can be seen that the use of partial sum lookahead in the radix-4 architecture needs 16 free processing units, the same idea can be used for previous stages such as stage $log_2(n) - 4$ but in this case higher number of idle processing units are needed (namely 256 processing unit) along with a huge multiplexer. In the 1024 decoder, only 256 processing units are instantiated so the partial sum lookahead can not be used for stage $log_2(n) - 4$ since the maximum number of free processing units will be 255. Figure 3.14 shows the clock cycle scheduling of 2 (64-bit SC decoders) both uses the radix-4 architecture while only the second one combines it with partial sum lookahead technique.

The partial sum lookahead was being used for 3/4 of the total LLRs calculated (those calculated using only $log_2(n) - 2$ calculations with out getting back to older stages). Thus the partial sum lookahead decreases 75% of the total cycles of calculating $log_2(n) - 2$. The total cycles will be $75\% * n/4 = \frac{3*n}{16}$. That is in the case of (1024,512) decoder, the decoding latency was reduced by 192 cycles to reach 404 cycles compared to 596 cycles for the radix-4 only architecture. The savings in latency are more significant with higher code lengths. In figure 3.14, the notation L_i^k refers to the i^{th} LLR at the k^{th} decoding level. The notation $L_{i:j}^k$ refers to the set of LLRs from L_i^k to L_j^k.

3.4.8 Memory architecture

To reduce the overall latency, the memory architecture is implemented in a way to achieve concurrent access for all LLR input values for a specific PU. The proposed architecture includes 64 single-port memory banks. The 0th (first) decoding level processing the input LLRs (received from the channel) has the the i^{th} PU. This PU has four inputs $L_i^0, L_{i+n/4}^0, L_{i+2n/4}^0, L_{i+3n/4}^0$, where n is the size of the codeword. In order to avoid conflicts in memory accesses, the LLRs are stored in different memory banks to guarantee simultaneous access. Similarly, the i^{th} PU at next decoding level has 4 inputs; $L_i^2, L_{i+n/16}^2, L_{i+2n/16}^2, L_{i+3n/16}^2$ which are stored in different banks too. This avoids any possible memory access bottleneck which affects the decoding latency significantly.

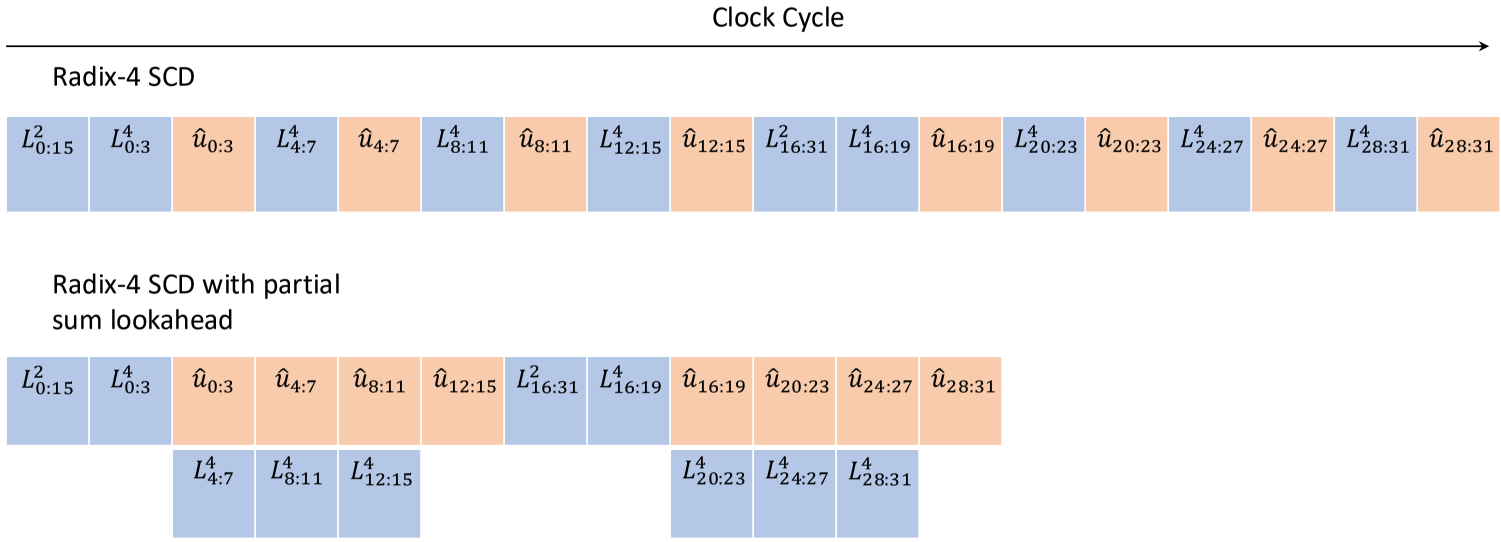


Figure 3.14: A comparison between Radix-4 SC decoding and Radix-4 SC with partial sum lookahead for a 64-bit decoder

3.5 Conclusion

The channel coding is divided into 2 parts, the channel combining and channel splitting. These processes make use of N different B-DMC channels to generate another set of N channels that are polarized in the way that some of them are not reliable for data transmission while other channels are highly reliable.
Successive cancellation decoder is based on processing units that are capable of calculating 2 types of equations f, g. The usage of log likelihood ratios along with the minimum sum approximation decreased the mathematical complexity of this processing unit.
A radix-4 processing unit was presented that is capable of calculating the LLRs by the use of ff, fg, gf, gg functions. A special last stage processing unit that is capable of handling the only 6 legal cases of frozen bits is presented.
The last stage processing unit is extended to some of the easy scenarios of subsequent bits so that it is capable of decoding more than 4-bit at the same clock cycle if any of this frozen bit scenarios are valid.
A new partial sum calculation circuit is adopted and updated to fit the multi-bit decoding in the last stage unit. Since lots of processing units are instantiated and not used all time because of the serial behavior of SCD, these processing units are used to handle every available combination of decoded bits and then multiplex the correct choice once those bits are decided. The memories are divided into banks to be able to provide the bandwidth required by the radix-4 processing units.

Chapter 4: BER simulation results

4.1 BER simulations results on different channel models

Matlab simulations have been done to prove the proposed architecture bit error performance on a binary discrete memoryless channel on both binary erasure channel and AWGN.
The binary erasure channel is modeled by XORing random bits in the data flow with a probability 1-W.
While the AWGN is modeled by adding some white noise (on all frequencies with Gaussian probability) on the bit stream after being transmitted, there is no easy way to map the AWGN channel versus the probability of error as it is not quantized error insertion, so it makes more sense to model the AWGN BER versus the Signal to noise ratio (SNR).

4.2 BER simulations results on a binary erasure channel

Different codeword length where used ranging from 16 bit to 1024 bit. Shorter codewords where not used because the polar codes performance will not appear so the error correction performance will be worse. And because the differences between the reed-muller codes and polar codes vanish at those short codewords.
The Matlab simulations where done on a count of simulation iterations large enough to get the expected BER at a high accuracy (i.e the total bit count simulated is 100 times the BER expected or more). Frozen bits patterns are calculated using Bhattachrya parameter, coding rate was set to $\frac{1}{2}$, 8 different values of transition probabilities, Matlab BSC function was used with 1-W as probability of error input, the results were plotted on the Y axis with a log scale while the X axis plotting erasure probability on a log graph.

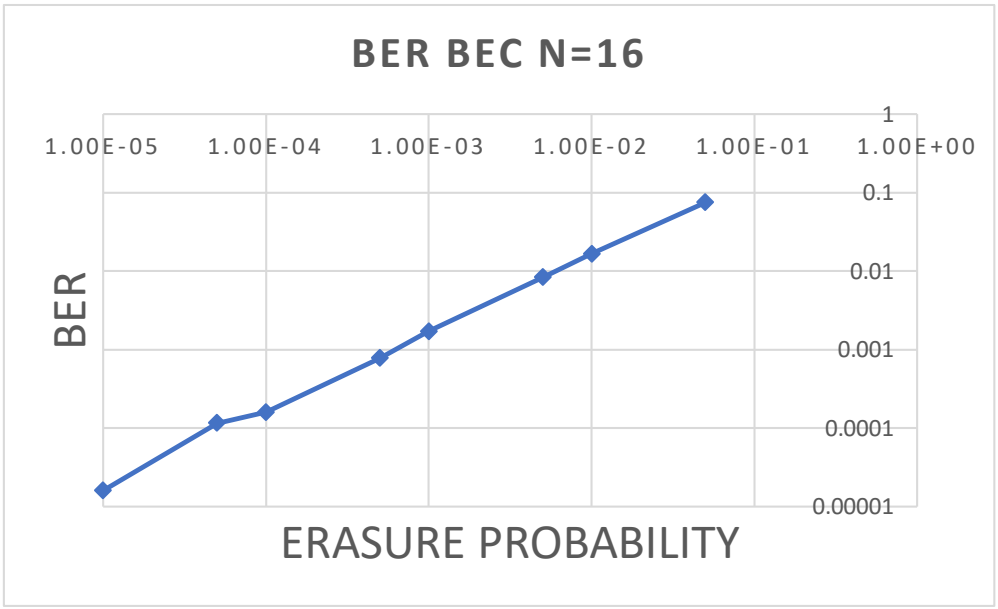


Figure 4.1: BER with a polar codes length N 16 on a binary erasure channel

Figure 4.1 shows the BER with a polar codes length $N = 16$ and code rate= $\frac{1}{2}$ on a binary erasure channel with erasure probability P_e where the X axis represents P_e and the Y axis represents BER. It can be seen that at higher erasure probability the bit error rate is approximately equal to the erasure probability, this can be explained that the error correction performance of polar codes at codeword length 16 is not good enough because the channels are not really polarized, in other words the selected bits are not very reliable compared to the frozen bits, this problem is fixed with every longer codeword length applied.

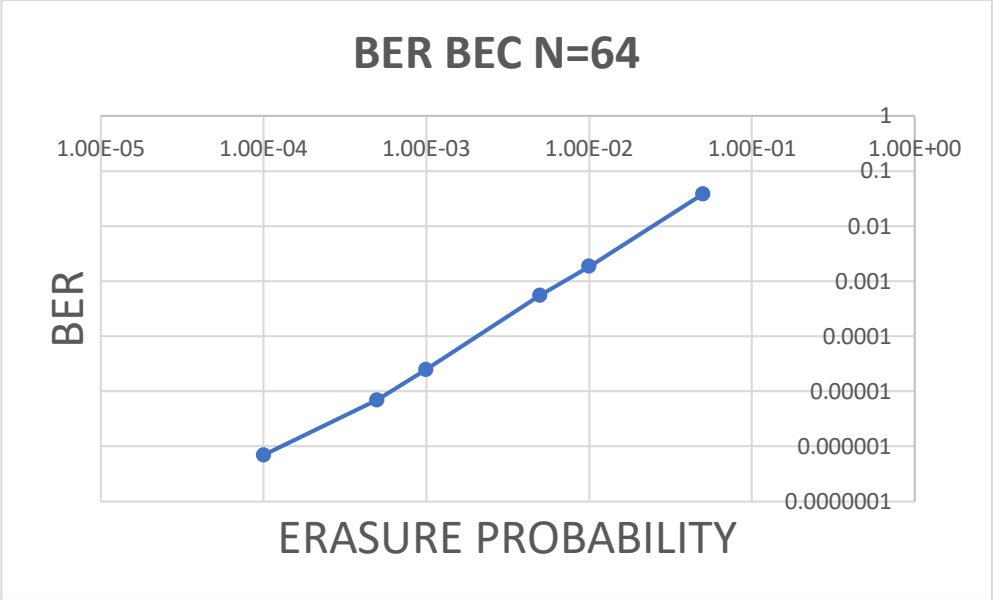


Figure 4.2: BER with a polar codes length N 64 on a binary erasure channel

Figure 4.2 shows the BER with a polar codes length $N = 64$ and code rate$\frac{1}{2}$ on a binary erasure channel with erasure probability P_e where the x axis represents P_e and the Y axis represents BER. It is easily noted that the BER rate dropped to approximately $\frac{1}{2}$ for the slightly longer codeword length. Error probability didn't reach zero in the smaller erasure probability points, but because of the limited simulation power, for a relatively convenient simulation time the lowest error probability reached was $7E(-7)$.

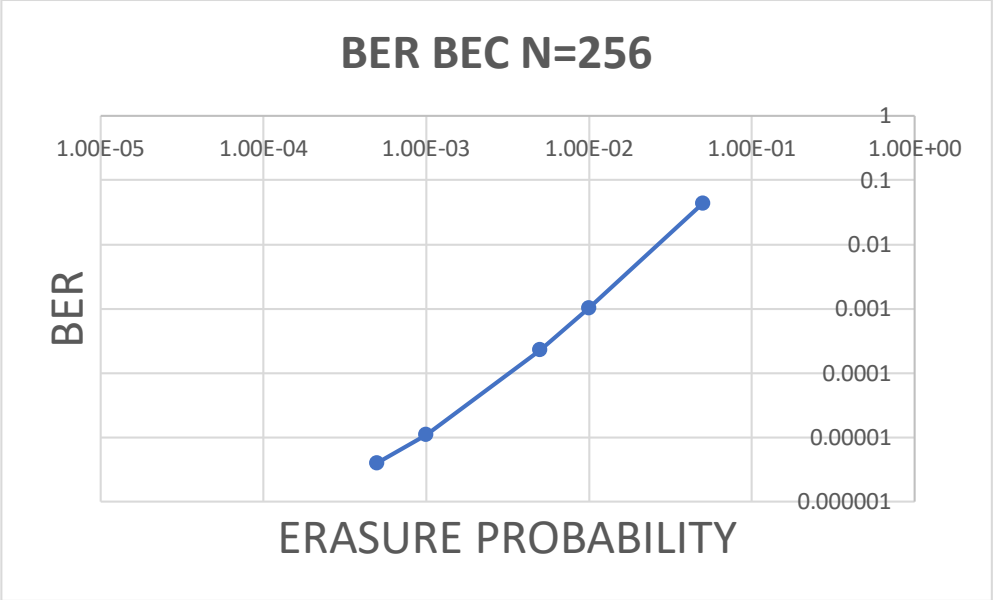


Figure 4.3: BER with a polar codes length N 256 on a binary erasure channel

Figure 4.3 shows the BER with a polar codes length $N = 256$ and code rate$\frac{1}{2}$ on a binary erasure channel with erasure probability P_e where the x axis represents P_e and the Y axis represents BER. Here the BER curve slope is very high, the error probability drops fast when the erasure probability is dropped from $1.00E-01$ to $1.00E-02$, There is a simulation limitation starting to appear with this curve with the stated simulation iterations, the Error probability after decoding is getting very rare that it needs very long Matlab simulations to cover. Thus the lowest error probability to get in a convenient simulation time is $4E(-6)$.

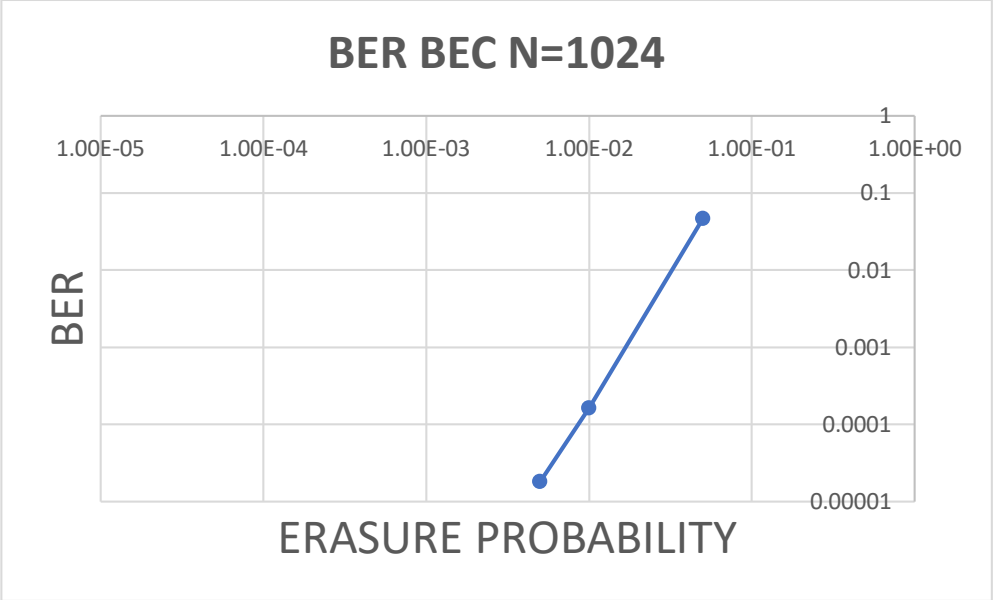


Figure 4.4: BER with a polar codes length N 1024 on a binary erasure channel

Figure 4.4 shows the BER with a polar codes length $N = 1024$ and code rate= $\frac{1}{2}$ on a binary erasure channel with erasure probability P_e where the X axis represents P_e and the Y axis represents BER. It can be seen that still for high erasure probability $5.00E-02$, the BER is approximately the same as the erasure probability thus there is no real gain for the coding at this high erasure rates, yet at lower erasure probability $1.00E-05$ the BER obviously dropped to near zero BER, because of the simulation length limitation, it can be seen that the BER is clipped at a lowest value equal $1.8E(-5)$ for a convenient simulation time.

4.3 BER simulations results on AWGN channel

Different codeword length where used ranging from 16 bit to 1024 bit. Shorter codewords where not used because the polar codes performance will not appear so the error correction performance will be worse. And because the differences between the reed-muller codes and polar codes vanish at those short codewords.
The Matlab simulations where done on a count of simulation iterations large enough to get the expected BER at a high accuracy (i.e the total bit count simulated is 100 times the BER expected or more). Frozen bits patterns are calculated using Bhattachrya parameter, coding rate was set to $\frac{1}{2}$, 10 different values of Signal to noise ratios were used, Matlab AWGN function was used with SNR as input, the results were plotted on the Y axis with a log scale while the X axis plotting signal to noise ratio SNR in Db.

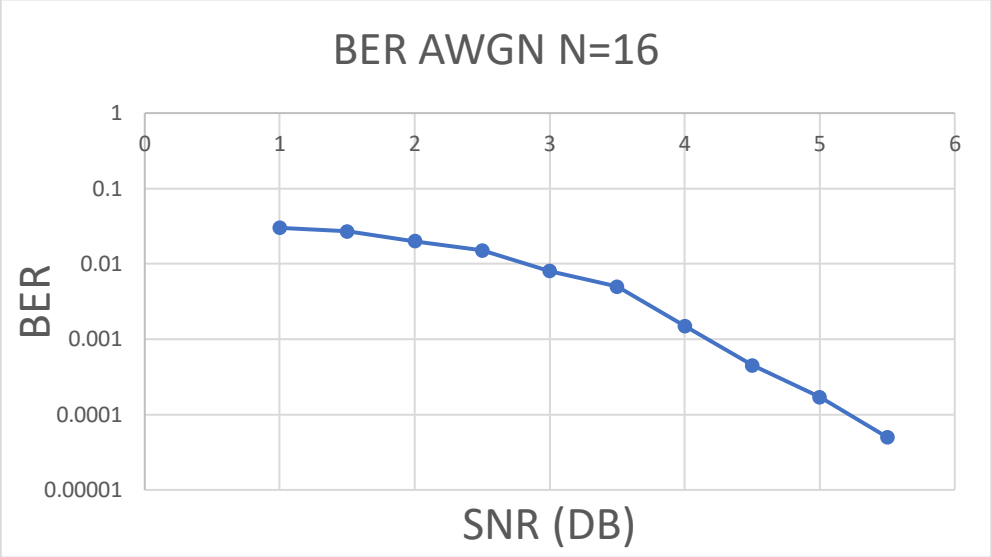


Figure 4.5: BER with a polar codes length N 16 on an AWGN channel

Figure 4.5 shows the BER with a polar codes length $N = 16$ and code rate$\frac{1}{2}$ on an AWGN channel with signal to noise ratio SNR represented on X axis and the Y axis represents BER. The AWGN is adding a white Gaussian noise to the bit stream after encoding, this is not as measurable as the Erasure because we can only manage the Signal to noise ratio SNR between the transmitted data and the noise thus there is an indirect control to the actual error probability injected to the coded data stream. As depicted from the figure when SNR gets high the probability of error drops. It can be seen that at 5 db the BER is around 0.0003

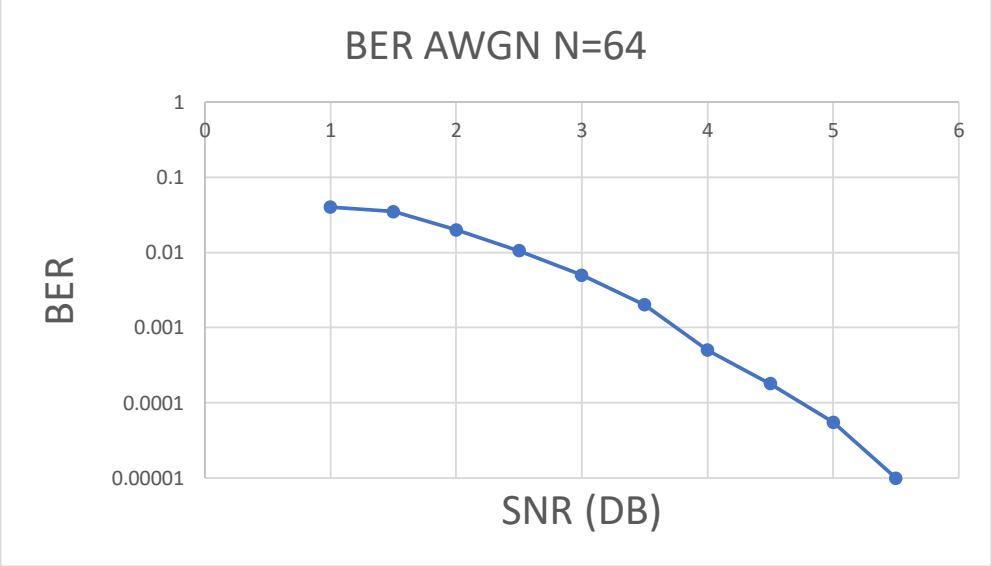


Figure 4.6: BER with a polar codes length N 64 on an AWGN channel

Figure 4.6 shows the BER with a polar codes length $N = 64$ and code rate$\frac{1}{2}$ on an AWGN channel with signal to noise ratio SNR represented on x axis in DB and the Y axis represents BER on a logarithmic scale. The BER at $N = 64$ compared to $N = 16$ is getting better as expected.

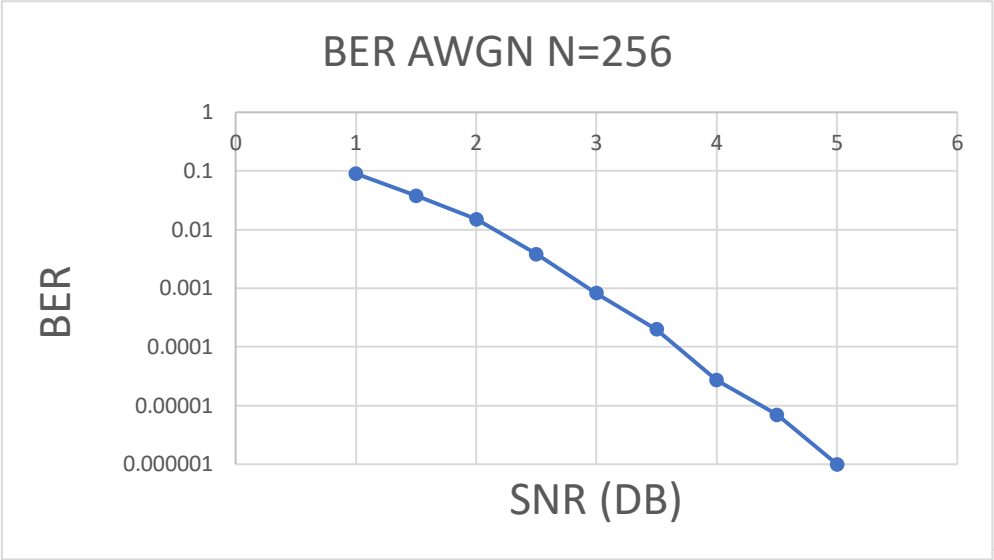


Figure 4.7: BER with a polar codes length N 256 on an AWGN channel

Figure 4.7 shows the BER with a polar codes length $N = 256$ and code rate$\frac{1}{2}$ on an AWGN channel with signal to noise ratio SNR represented on X axis and the Y axis represents BER on a logarithmic scale. The water drop shape of the curve as expected shows that as SNR increases, the BER drops with higher rate. This shows the effect of channel coding on the error correction performance. Yet at $SNR = 5.5Db$ the BER dropped to less than 0.000001, thus it was very hard to get a non zero value with the limited simulation iterations that can be possible in a convenient simulation time.

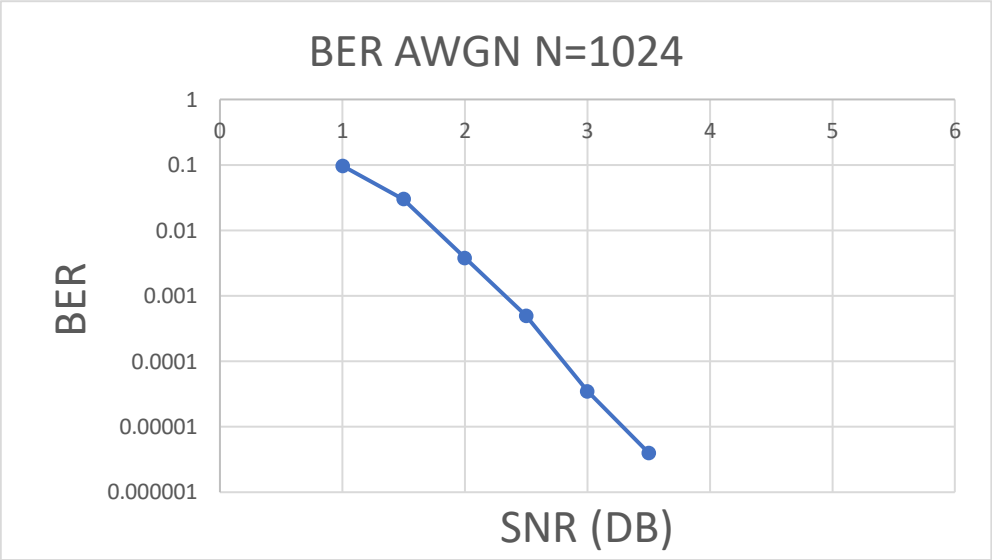


Figure 4.8: BER with a polar codes length N 1024 on an AWGN channel

Figure 4.8 shows the BER with a polar codes length $N = 1024$ and code rate$\frac{1}{2}$ on an AWGN channel with signal to noise ratio SNR represented on X axis and the Y axis represents BER. Since $N = 1024$ this gives the least error probability. With the initial minimum number of simulation iterations, the BER dropped to zero. Thus another simulations were run with more simulation points and the results were averaged until reaching the expected water drop shape of the curve. but at $SNR = 4Db$ the BER error rate dropped below the 0.000001 Threshold, yet was very hard to get a non zero value with the limited simulation iterations that can be possible in a convenient simulation time.

Chapter 5: Results

5.1 Results

5.1.1 Reducing decoding latency

The proposed polar decoder latency is computed as: $\frac{n}{4}$ cycles consumed by last stage PU, $\frac{n}{4^i}$ cycles used in calculation of the i^{th}stage LLRs where $i = 1, 2, 3, ..., \frac{\log_2(n)}{2} - 1$. The total latency then is $\frac{n}{4} + \frac{n}{4} + \frac{n}{16} + ... + \frac{n}{4^{\frac{\log_2(n)}{2}-1}} = \frac{n}{4} + \sum_{k=1}^{k=\frac{\log_2(n)}{2}-1} n.\left(\frac{1}{4}\right)^k = \frac{7}{12}n - \frac{4}{3}$ which is clearly less than the $(0.75n - 1)$ latency defined in [26] using SC decoder with precomputation.
Furthermore, the proposed partial sum lookahead replaces the $n/4$ by $n/16$, and the corresponding latency $= \frac{n}{4} + \frac{n}{16} + \frac{n}{16} + ... + \frac{n}{4^{\frac{\log_2(n)}{2}-1}} = \frac{n}{16} + \sum_{k=1}^{k=\frac{\log_2(n)}{2}-1} n.\left(\frac{1}{4}\right)^k = \frac{19}{48}n - \frac{4}{3}$.
Depending on the count of the Special codes of certain length in the codeword, the effect of special sub-code handling on latency is calculated.
Table 5.1 shows the count of length 16 special sub-codes occurrences in different codeword length of code rate $\frac{1}{2}$.
Each sub-code occurrence replaces the 5 cycles needed to decode 16-bit code with the use of partial sum look ahead with just once cycle, Thus each occurrence gives 4 clock cycle latency gain.

Table 5.1: 16-bit Special sub-code occurrences in different codeword length with code rate $\frac{1}{2}$ and the corresponding latency gain

n	**1024**	**4096**	**16384**	**65536**
occurrence count	7	37	177	803
latency gain(cycles)	28	148	708	3212

To maintain consistency in comparison, the latency calculation assumes that both control unit processing and memory accesses are done concurrently in the same clock cycle with PU processing.
Table 5.2 compares the latency improvements in the proposed architecture with those in [26].

Table 5.2: Latency Reduction in the proposed architecture (clock cycles)

codeword size (n)	**64**	**256**	**1024**	**4096**	**16384**	**65536**
Reference [26]	47	191	767	3071	12143	49151
Proposed Radix-4 SC decoder	36	148	596	2388	9444	38228
Proposed partial sum lookahead	24	100	404	1620	6484	25940
Prop. with spcl sub-codes.	-	-	376	1472	5776	22728

5.1.2 Area Improvement

According to [14] the LLR quantization is optimum in between $Q = 5$ or $Q = 6$. $Q = 5$ with decoder inputs saturated between $[-2, 2]$ is selected. The number of needed PUs is decreased to $\frac{n}{4}$ in the line architecture which is half the count in [30], the single PU area is approximately doubled. The areas of one LSPU module and one Special sub-codes decoding unit are negligible compared to the $\frac{n}{4}$ PUs areas. Therefore, the total area is almost similar to [26]. The proposed architecture no longer need to compute and store LLRs in each stage. Otherwise, it computes and store LLRs only in even stages, skipping the odd stages. As a consequence, the memory size is reduced to half. The resulting memory size for the proposed architecture is$Q * \frac{n}{2} \log n$ bits.

5.1.3 VLSI implementation

To compare the proposed architecture to [26] and the dedicated decoder of [13], the hardware implementation of a (1024, 512) radix-4 SC decoder based on tree decoding architecture was done. The design is implemented with Verilog hardware description language, while synthesis is done using Synopsys DC tool using both TSMC $65nm$ and Open Cell Library $15nm$ FreePDK Technology [16]
Table 5.3 compares [26] , [13] and the proposed architecture for different aspects.

Table 5.3: Results of (1024, 512) SC decoder implementation quantized at 5-bits

Polar decoder implementation	**[13]**	**[26]**	Proposed Radix-4 SC decoding	
Frequency(MHz)	750	500	3058	650
Decoding latency (cycle)	767	365	596	596
Total gate count	338499	1050000	377420	259270
Throughput (Mbps)	500	1360	2647	556.898
CMOS Technology	$45nm$	$65nm$	$15nm$	$65nm$
TSNT (Mbps/Kgate) (scaled to $65nm$)	1.024	1.3	1.62	2.173

Where TSNT is the Technology Scaled Normalized Throughput and defined as follows:

$$TSNT = Throughput * \frac{tech.}{65nm} * \frac{1}{gatecount} \tag{5.1}$$

Table 5.3 shows that the proposed design has a reduction in gate count of TSMC implementation by 23.4% with respect to [26]. The latency has also been reduced by a factor of 10% although it operates at a lower maximum clock frequency. If both designs operate at a similar frequency, the decoding latency reduction shall be $\frac{DecodingLatency_{old} - DecodingLatency_{proposed}}{DecodingLatency_{old}} = 22.3\%$.
The resulting gate count is different between the $15nm$ and the $65nm$ because of using two different technology kits; TSMC and freePDK. The freePDK is lacking some complex standard cells that is left to synthesizer thus degrading the gate count optimization found in TSMC kit.

The overall improvement in performance is represented by the normalized throughput increase. The improvement in TSNT is 212% compared to [26] as calculated in 5.1.
This TSNT is further improved after using line architecture thus decreasing the count of needed PUs and being able to use the partial sum lookahead and special sub-codes handling techniques.

The power consumption of the proposed radix-4 decoder (using design compiler and TSMC $65nm$) is 1.66 uW as leakage power , and 1.0187 uW as Dynamic power. This power consumption is comparable to state of the art implementations.

Chapter 6: Conclusion

6.1 Conclusion

Channel coding techniques add redundancy to the message to help in error detection and correction. The maximum rate of reliable communication of a message on a B-DMC is stated by Shannon to be the Channel capacity C. The polar codes are proven to achieve the channel capacity asymptotically when the codeword length tends to infinity.
LDPC ,Reed-muller and turbo codes are the closest contenders to polar codes. Polar codes differs from Reed-Muller codes in the frozen bit selection criteria, polar codes uses the Bhattacharyya parameter instead of the hamming distance. Polar encoders are pretty simple and several implementation were introduced.
Several polar decoders were introduced. Successive cancellation decoder is a low complexity decoder yet it suffers from high decoding latency. Because of the similarity between polar codes and reed-muller codes, belief propagation decoders can also be used for polar codes.
The error correction performance of polar codes with successive cancellation decoders can get even better with the use of the generalization of decoder called successive cancellation list decoder, although having slightly higher complexity.
Successive cancellation decoders latency were further reduced by introducing the multi-bit successive cancellation decoding. The decoding complexity itself was reduced by the usage of log likelihood ratios instead of likelihood ratios along with minimum sum approximation. This is done without affecting the error correction performance. A radix-4 processing unit was proposed that is capable of calculating the LLR values skipping 2 stages at a time. Another special hard decision unit was proposed to cope with the new processing unit, it is able to decide 4-bits at a time without performance loss.
The (1024, 512) successive cancellation decoding latency was decreased from 767 to 597 cycles. The throughput and memory efficiency have been increased. The decoding latency further decreased by the introduction of partial Sum lookahead. It was able to decrease the decoding latency further to 404 clock cycles at no additional area overhead or complexity. Furthermore, the special sub-codes handling reduced the decoder latency to 376 clock cycles, this gain is proven to grow for longer codeword length. In addition, the proposed architecture improves the TSNT by more than 212% compared to existing counterpart.

6.2 Future work

Many different implementations, adaptations and further tests have been left for future work because of the limited available time of this work.
The future work is going to be presented as points for simplicity:

1. Regarding the current algorithm, we are looking forward to implement the current RTL implementation on FPGA, this will enable giving exact figures

with calculation of the exact error correction performance while providing easier verification of the functionality.

2. Studying generation of higher radix decoders. Although adopting higher radix decoders may add extra complexity to the last stage processing unit, there shall be an optimal radix decreasing the latency to minimum without adding a deal breaking complexity.

3. Studying the current work with other decoders for polar codes such as: belief propagation and successive cancellation list decoders, studying also how to use the current proposal for SCL decoders.

4. Studying possible application of polar codes and what changes shall be addressed to current implementation to follow the application requirements

References

[1] Alamdar-Yazdi, A., and Kschischang, F. R. A simplified successive-cancellation decoder for polar codes. *IEEE Communications Letters 15*, 12 (December 2011), 1378–1380.

[2] Arikan, E. A performance comparison of polar codes and reed-muller codes. *IEEE Communications Letters 12*, 6 (June 2008), 447–449.

[3] Arikan, E. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory 55*, 7 (July 2009), 3051–3073.

[4] Balatsoukas-Stimming, A., and Burg, A. Faulty successive cancellation decoding of polar codes for the binary erasure channel. *IEEE Transactions on Communications 66*, 6 (June 2018), 2322–2332.

[5] Balatsoukas-Stimming, A., Giard, P., and Burg, A. Comparison of polar decoders with existing low-density parity-check and turbo decoders. *CoRR abs/1702.04707* (2017).

[6] Berhault, G., Leroux, C., Jego, C., and Dallet, D. Partial sums generation architecture for successive cancellation decoding of polar codes. In *SiPS 2013 Proceedings* (Oct 2013), pp. 407–412.

[7] Berrou, C., Glavieux, A., and Thitimajshima, P. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE International Conference on* (May 1993), vol. 2, pp. 1064–1070 vol.2.

[8] Bioglio, V., Gabry, F., Land, I., and Belfiore, J. Minimum-distance based construction of multi-kernel polar codes. *CoRR abs/1701.07616* (2017).

[9] Chang, Y.-N. Design of an area-efficient partial-sum architecture for polar decoders based on new matrix generator. In *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)* (June 2016), pp. 1–4.

[10] Dizdar, O., and ArÄŚkan, E. A high-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic. *IEEE Transactions on Circuits and Systems I: Regular Papers 63*, 3 (March 2016), 436–447.

[11] Fan, Y., and Tsui, C. An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation. *IEEE Transactions on Signal Processing 62*, 12 (June 2014), 3165–3179.

[12] Gallager, R. Low-density parity-check codes. *IRE Transactions on information theory 8*, 1 (1962), 21–28.

[13] Giard, P., Thibeault, C., and Gross, W. J. *High-speed decoders for polar codes*. Springer, 2017.

[14] Leroux, C., Raymond, A. J., Sarkis, G., and Gross, W. J. A semi-parallel successive-cancellation decoder for polar codes. *IEEE Transactions on Signal Processing 61*, 2 (Jan 2013), 289–299.

[15] Leroux, C., Tal, I., Vardy, A., and Gross, W. J. Hardware architectures for successive cancellation decoding of polar codes. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (May 2011), pp. 1665–1668.

[16] Martins, M., Matos, J. M., Ribas, R. P., Reis, A., Schlinker, G., Rech, L., and Michelsen, J. Open cell library in 15nm freepdk technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design* (New York, NY, USA, 2015), ISPD '15, ACM, pp. 171–178.

[17] Massey, J. Coding and modulation in digital communications. *Proc. Int. Zurich Sem.* (1974), E2(1)–E2(4).

[18] Raymond, A. J., and Gross, W. J. Scalable successive-cancellation hardware decoder for polar codes. In *2013 IEEE Global Conference on Signal and Information Processing* (Dec 2013), pp. 1282–1285.

[19] Sarkis, G., Giard, P., Vardy, A., Thibeault, C., and Gross, W. J. Fast polar decoders: Algorithm and implementation. *IEEE Journal on Selected Areas in Communications 32*, 5 (May 2014), 946–957.

[20] Sarkis, G., Giard, P., Vardy, A., Thibeault, C., and Gross, W. J. Fast list decoders for polar codes. *IEEE Journal on Selected Areas in Communications 34*, 2 (Feb 2016), 318–328.

[21] Shannon, C. E. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev. 5*, 1 (Jan. 2001), 3–55.

[22] Simon, H. Communication systems, 2017.

[23] Tal, I., and Vardy, A. List decoding of polar codes. *CoRR abs/1206.0050* (2012).

[24] Thomas, E. K., Tan, V. Y. F., Vardy, A., and Motani, M. Polar coding for the binary erasure channel with deletions. *CoRR abs/1701.01938* (2017).

[25] Yuan, B., and Parhi, K. K. Algorithm and architecture for hybrid decoding of polar codes. In *2014 48th Asilomar Conference on Signals, Systems and Computers* (Nov 2014), pp. 2050–2053.

[26] Yuan, B., and Parhi, K. K. Low-latency successive-cancellation polar decoder architectures using 2-bit decoding. *IEEE Transactions on Circuits and Systems I: Regular Papers 61*, 4 (April 2014), 1241–1254.

[27] Yuan, B., and Parhi, K. K. Low-latency successive-cancellation list decoders for polar codes with multibit decision. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 23*, 10 (Oct 2015), 2268–2280.

[28] Yuan, B., and Parhi, K. K. Successive cancellation decoding of polar codes using stochastic computing. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (May 2015), pp. 3040–3043.

[29] Yuan, B., and Parhi, K. K. Llr-based successive-cancellation list decoder for polar codes with multi-bit decision. *CoRR abs/1603.07055* (2016).

[30] Zhang, C., Yuan, B., and Parhi, K. K. Reduced-latency sc polar decoder architectures. In *2012 IEEE International Conference on Communications (ICC)* (June 2012), pp. 3471–3475.

Appendix A: Important matlab and verilog codes

A.1 Choosing the frozen bits

All this part is written using matlab.

A.1.1 calculating Bhattacharyya parameter

```
function Z=z_value(k,j,W);
if(k==2)
if(j==1)
Z=1-(W^2);
else %j=2
Z=(1-W)^2;
end%if(j=1)
else
if(j<=k/2)
Z=2*z_value(k/2,j,W)-(z_value(k/2,j,W))^2;
else
Z=(z_value(k/2,j-(k/2),W))^2;
end%if(j<=k)
end%if(k==2)
end%function

function Z=z_all(N,W);
for j=1:1:N
Z(j)=z_value(N,j,W);
end
end%function
```

A.1.2 Setting the pattern array ufixed

```
function u_fixed=ufixed(N,G_N,length_fixed,W);
[~,sorted_index]=sort(z_all(N,W));
maxIndex=sorted_index(1:length_fixed);
u_fixed=ones(1,N);
for maxindex_count=1:(N-length_fixed)
u_fixed(maxIndex(maxindex_count))=0;
end
end
```

where ufixed=1 for a frozen bit , 0 for a data bit

A.2 Processing unit verilog implementation

```
module processing_unit(a,b,c,d,S11,S12,S21,S22,function_type,result);
parameter width =5;
input wire [width-1:0] a;
input wire [width-1:0] c;
input wire [width-1:0] b;
input wire [width-1:0] d;
input wire S11;
input wire S12;
input wire S21;
input wire S22;
input wire [1:0] function_type; // 0: FF 1:FG 2:GF 3:GG
output wire [width-1:0] result;
wire [width-1:0] adder_1_a,adder_2_a,adder_3_a;
wire [width-1:0] adder_1_b,adder_2_b,adder_3_b;
wire  add_sub_1,add_sub_2,add_sub_3;
wire  overflow1,overflow2,overflow3;
wire [width-1:0] out1,out2,out3;
wire [width-1:0] input_1_1,input_1_2,input_1_3;
wire [width-1:0] input_2_1,input_2_2,input_2_3;
wire [width-1:0] minimum_1,minimum_2,minimum_3;
wire  result_A_BBAR_1,result_A_BBAR_2,result_A_BBAR_3;

adder_subtractor #width
u_adder_subtractor_1(adder_1_a,adder_1_b,add_sub_1,out1,overflow1);
adder_subtractor #width
u_adder_subtractor_2(adder_2_a,adder_2_b,add_sub_2,out2,overflow2);
adder_subtractor #width
u_adder_subtractor_3(adder_3_a,adder_3_b,add_sub_3,out3,overflow3);
min_of_absolute #width
u_min_of_absolute_1(input_1_1,input_2_1,result_A_BBAR_1,minimum_1);
min_of_absolute #width
u_min_of_absolute_2(input_1_2,input_2_2,result_A_BBAR_2, minimum_2);
min_of_absolute #width
u_min_of_absolute_3(input_1_3,input_2_3,result_A_BBAR_3,minimum_3);
//min_
assign input_1_1=a;
assign input_2_1=c;

assign input_1_2=b;
assign input_2_2=d;

assign input_1_3=(function_type==0)?minimum_1:out1;
assign input_2_3=(function_type==0)?minimum_2:out2;

assign  adder_1_a=(function_type==1)?0:c;
```

```
assign  adder_1_b=(function_type==1)?minimum_2:a;
assign  add_sub_1=(function_type==1)?b[width-1]^d[width-1]:S11;

assign  adder_2_a=(function_type==1)?out1:d;
assign  adder_2_b=(function_type==1)?minimum_1:b;
assign  add_sub_2=(function_type==1)?a[width-1]^c[width-1]^S21:S12;

assign  adder_3_a=(function_type[0]==0)?0:out2;
assign  adder_3_b=(function_type[0]==0)?minimum_3:out1;
assign  add_sub_3=(function_type==0)?a[width-1]^c[width-1]^b[width-1]^d[width-1]:
  (function_type==2)?out1[width-1]^out2[width-1]:S22;

assign  result=((function_type==0)||(function_type==3))?out3:(function_type==1)?
out2:minimum_3;
endmodule

module min_of_absolute(a,b,ABbar,minimum);
parameter width =5;
input wire [width-1:0] a;
input wire [width-1:0] b;
output wire ABbar;
output wire [width-1:0] minimum;
wire [width-1:0] temp_subtractor_result;
wire [width-1:0] result_before_absolute;
wire overflow,overflow2;
wire subtract;
adder_subtractor #width u0(a,b,subtract,temp_subtractor_result,overflow);
adder_subtractor #width absolute_module({width{1'b0}},result_before_absolute
,result_before_absolute[width-1],minimum,overflow2);
assign subtract=a[width-1]~^b[width-1];
assign ABbar= a[width-1]^temp_subtractor_result[width-1];
assign result_before_absolute=ABbar?a:b;
endmodule

module adder_subtractor(a,b,subtract,sum,overflow);
parameter width =5;
input wire [width-1:0] a;
input wire [width-1:0] b;
input wire subtract;
output wire [width-1:0] sum;
output wire overflow;

wire [width-1:0] carry_temp;
adder_subtractor_bit_block u0(a[0],b[0],subtract,subtract,sum[0],carry_temp[0]);
generate
genvar i;
```

```
for(i=1;i<width;i=i+1)
begin : generate_adder_bits
adder_subtractor_bit_block u(a[i],b[i],carry_temp[i-1],subtract,sum[i],carry_temp
end
endgenerate
assign overflow=carry_temp[width-1]^carry_temp[width-2];
endmodule
```

A.3 Last stage Processing unit verilog implementation

```
module hard_decision_unit(LLR0,LLR1,LLR2,LLR3,freeze_pattern,u0,u1,u2,u3);
parameter LLR_WIDTH=5;
input wire [LLR_WIDTH-1:0] LLR0;
input wire [LLR_WIDTH-1:0] LLR1;
input wire [LLR_WIDTH-1:0] LLR2;
input wire [LLR_WIDTH-1:0] LLR3;
input wire [3:0] freeze_pattern;
output reg u0;
output reg u1;
output reg u2;
output reg u3;
reg [LLR_WIDTH-1:0] absLLR0,absLLR1,absLLR2,absLLR3;
reg [LLR_WIDTH-1:0]sum,sum2;
always @*
begin
case (freeze_pattern)
4'b0000:
begin
u0=0;
u1=0;
u2=0;
u3=0;
end
4'b1000:
begin
u0=0;
u1=0;
u2=0;
sum=LLR0+LLR1+LLR2+LLR3;
u3=sum[LLR_WIDTH-1];
end
4'b1100:
begin
sum=LLR0+LLR2;
sum2=LLR1+LLR3;
u0=0;
u1=1;
u2=sum[LLR_WIDTH-1]^sum2[LLR_WIDTH-1];
u3=sum2[LLR_WIDTH-1];
end
4'b1010:
begin
sum=LLR0+LLR1;
```

```
sum2=LLR2+LLR3;
u0=0;
u1=sum[LLR_WIDTH-1]^sum2[LLR_WIDTH-1];
u2=0;
u3=sum2[LLR_WIDTH-1];
end
4'b1110:
begin
u0=0;
sum=LLR0+LLR2;
sum2=LLR1+LLR3;
if(sum[LLR_WIDTH-1]) sum=0-sum;
if(sum2[LLR_WIDTH-1]) sum2=0-sum2;
if(sum<sum2)
begin
u1=LLR0[LLR_WIDTH-1]^LLR2[LLR_WIDTH-1];
if(absLLR1[LLR_WIDTH-1]) absLLR1=0-absLLR1;
if(absLLR3[LLR_WIDTH-1]) absLLR3=0-absLLR3;
if(absLLR1<absLLR3)
begin
u2=LLR2[LLR_WIDTH-1]^LLR3[LLR_WIDTH-1];
u3=LLR3[LLR_WIDTH-1];
end
else
begin
u2=LLR0[LLR_WIDTH-1]^LLR1[LLR_WIDTH-1];
u3=u2^LLR2[LLR_WIDTH-1];
end
end
else
begin
u1=LLR1[LLR_WIDTH-1]^LLR3[LLR_WIDTH-1];
if(absLLR0[LLR_WIDTH-1]) absLLR0=0-absLLR0;
if(absLLR2[LLR_WIDTH-1]) absLLR2=0-absLLR2;
if(absLLR0<absLLR2)
begin
u2=LLR2[LLR_WIDTH-1]^LLR3[LLR_WIDTH-1];
end
else
begin
u2=LLR0[LLR_WIDTH-1]^LLR1[LLR_WIDTH-1];
end
u3=LLR3[LLR_WIDTH-1];
end
end
4'b1111:
```

```
begin
u0=LLR0[LLR_WIDTH-1]^LLR1[LLR_WIDTH-1]^LLR2[LLR_WIDTH-1]^LLR3[LLR_WIDTH-1];
u1=LLR1[LLR_WIDTH-1]^LLR3[LLR_WIDTH-1];
u2=LLR2[LLR_WIDTH-1]^LLR3[LLR_WIDTH-1];
u3=LLR3[LLR_WIDTH-1];
end
endcase
end
endmodule
```

الملخص

حظيت الشفرات القطبية مؤخرا باهتمام شديد من قبل الباحثين نتيجة لاثبات قابليتها من الوصول الي السعة القصوي لقناة الاتصال عند استخدام جمل طويلة في التشفير.

و لكن سرعة الاستجابة لتصاميم فك التشفير تناسبت طرديا مع طول الجمل التشفيرية مما جعل استخدام هذه التقنية مع الاستخدامات المتطلبة لسرعات استجابة عالية يعد مستحيلا.

لتخطي هذه المشكلة، اقترحت هذه الرسالة معمارية لفك التشفير مستخدمة وحدات معالجة ذات اساس رباعي مع وحدة معالجة نهائية ذو طابع خاص، حيث انها قادرة علي تحديد حتي 16 جزيئ منطقي bit في ان واحد.

علاوة علي ذلك اقترحت الرسالة فك تشفير لحظي للشفرات الخاصة الجزئية لتقليل وقت الاستجابة الكلي.

تم ايضا استخدام تقنية التوقع المستقبلي للمجموع الجزئي مما ادي الي تقليل سرعة الاستجابة و رفع الاداء الكلي للمعمارية المقترحة.

مهندس: حسين جلال حسين حسن

تاريخ الميلاد: 16 \09 \1989

الجنسية: مصري

تاريخ التسجيل: 01 \10 \2013

تاريخ المنح: 2019

القسم: هندسة الالكترونيات و الاتصالات الكهربية

الدرجة: ماجستير العلوم

المشرفون:

ا.د. حسام علي حسن فهمي

الممتحنون:

أ.د. حسام علي حسن فهمي (المشرف الرئيسي)

أ.د. محمد محمد خيري (الممتحن الداخلي)

د. كريم جمعة صديق (الممتحن الخارجي)

استاذ مساعد، قسم هندسة الإلكترونيات و الإتصالات

كلية العلوم و الهندسة، الجامعة الامريكية بالقاهرة

عنوان الرسالة:

تنفيذ دائرة فك تشفير للشفرات القطبية باستخدام الالغاء المتتالي المبسط ذو اساس رباعي

الكلمات الدالة:

الشفرات القطبية، فك التشفير باستخدام الالغاء المتتالي المبسط، الاساس الرباعي، المجموع الجزئي، الشفرات الخاصة الجزئية.

ملخص الرسالة:

في هذه الرسالة، اقترحنا تصميم سريع الاستجابة لفك الشفرات القطبية، يستند هذا التصميم علي استخدام اسلوب الالغاء المتتالي لفك التشفير.

تستند الفكرة الرئيسية علي استخدام وحدات معالجة رباعية الاساس لحساب نسب الاحتمال الوسطية مع استحداث وحدة معالجة نهائية قادرة علي فك تشفير اربع جزيئات منطقية bits او اكثر باستخدام التوقع المستقبلي للمجموع الجزئي مما يساعد في تقليل سرعة الاستجابة مع تحسين المنفعة من الوحدات المعالجة.

تنفيذ دائرة فك تشفير للشفرات القطبية باستخدام الالغاء المتتالي المبسط ذو اساس رباعي

اعداد

حسين جلال حسين حسن

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الالكترونيات و الاتصالات الكهربية

يعتمد من لجنة الممتحنين:

الاستاذ الدكتور: حسام علي حسن فهمي **المشرف الرئيسى**

الاستاذ الدكتور: محمد محمد خيري **الممتحن الداخلي**

الدكتور: كريم جمعة صديق **الممتحن الخارجي**

استاذ مساعد،

قسم هندسة الإلكترونيات و الإتصالات

كلية العلوم و الهندسة، الجامعة الامريكية بالقاهرة

كليــة الهندســة - جامعــة القاهـرة

الجيـزة - جمهوريـة مصـرالعربيــة

2019

تنفيذ دائرة فك تشفير للشفرات القطبية باستخدام الالغاء المتتالي المبسط ذو اساس رباعي

اعداد

حسين جلال حسين حسن

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الالكترونيات و الاتصالات الكهربية

تحت اشراف

ا.د. حسام علي حسن فهمي

أستاذ بقسم هندسة الالكترونيات

و الاتصالات، كلية الهندسة،

جامعة القاهرة

كليــة الهندســة - جامعــة القاهـرة

الجيـزة - جمهوريـة مصـرالعربيــة

2019

تنفيذ دائرة فك تشفير للشفرات القطبية باستخدام الالغاء المتتالي المبسط ذو اساس رباعي

اعداد

حسين جلال حسين حسن

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الالكترونيات و الاتصالات الكهربية

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة
2019