MIXED PRECISION MULTI-FRAME PARALLEL LOW-DENSITY PARITY-CHECK CODE DECODER

A Thesis

by

YUANPENG NI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Gwan Choi
Committee Members,	Jiang Hu
	Duncan Walker
Head of Department,	Miroslav Begovic

December 2016

Major Subject: Computer Engineering

ABSTRACT

As the demand for high speed and high quality connectivity is increasing exponentially, channels are getting more and more crowded. The need for a high performance and low error floor channel decoder is apparent. Low-density parity-check code (LDPC) is a linear error correction code that can reach near Shannon limit. In this work, LDPC code construction and decoding algorithms are discussed, the LDPC decoder, in fully parallel and partial parallel, was implemented, and the features and issues related to corresponding architecture are analyzed. Furthermore, a multi-frame processing approach, based on pipelining and out-of-order processing, is proposed. The implemented decoder achieves 12.6 Gbps at 3.0 dB SNR. The mixed precision scheme is explored by adding precision control and alignment units before and after check node units (CNU) to improve performance, as well as error floor. By mixing the 6-bit and 5-bit precision CNUs at 1:1 ratio, the decoder reaches $\sim$0.5 dB lower FER and BER while retaining a low error floor.

DEDICATION

To my family.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Gwan Choi, for his support and encouragement for my research. He supported me in all the difficult situations where I needed help. He suggested me to try different implementation approaches of the LDPC decoder and investigate more technical detail of the FPGA structure, which is great helpful to me for a deeper understanding of the decoding algorithm and FPGA. And therefore I am able to perform FPGA targeted optimization in this work. I would also like to thank Dr. Duncan Walker and Dr. Jiang Hu, who provided great suggestions on extending the design from FPGA to ASIC.

Several other student and people at Texas A&M also helped me in my research. Thanks to Jingwei Xu, in particular, for working on the matlab simulation model for the fully parallel decoder architecture. Also thanks to Tiben Che who helped me with the setting up of FPGA board and provided help on the configuration of the debugger core.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Digital Communication

The demand for high speed and high quality connectivity is increasing exponentially as more and more wireless and wired communication standards are being formulated, for example, gigabyte Ethernet, fiber network, satellite communication, the latest IEEE 802.11ax Wi-Fi and IEEE 802.11ad WiGig standards, as well as upcoming 5G mobile network. Moreover, the rapidly growing internet of things (IoT) and wearable devices are pushing the connectivity requirement to an even higher level.

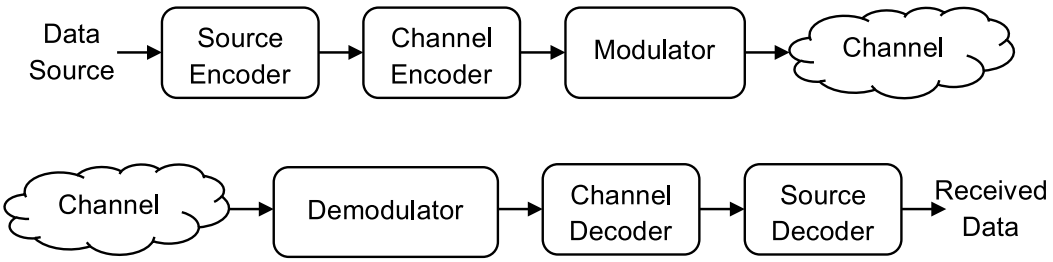

Figure 1.1: Digital communication system diagram

Normally, a digital communication system consists of source encoder, channel encoder, modulator, communication channel, demodulator, channel decoder and source decoder [1], shown in figure 1.1. Initially, the source encoder compresses the original data to minimize the size of data stream or package. Optionally, it can also encrypt the data for better security of communications. Channel encoder encodes the original information with a redundancy making it capable to be recovered after passes through the channel. After that, the encoded bits are modulated by digital modulator and loaded to the carrier, such as radio frequency (RF) waves, optical signal, or cabled electrical signal. At receiving side,

the demodulator transmits the received signal from modulated waveform, e.g. phase-shift keying (PSK) or quadrature amplitude modulation (QAM), to digital sequences. Channel decoder and source decoder then reconstruct the original information from these data. Besides these, the communication channel is the physical media that passing the signal from transmitter to receiver. In reality, the channel for wireless communication is usually the atmosphere and the space, or optical fiber and electrical cable for wired network. The general issue for these channels are noise and interference, so that the signal is often distorted after transmission. Additionally, the noise and interference can from multiple sources and in a variety of mechanisms. Generally, the noise is considered a random signal added to the signal being transmitted. A typical mathematics model for the noise in communication channel is additive Gaussian white noise (AWGN), which means the noise has a normal distribution in time domain and has uniform power across the frequency band. Channel encoder and channel decoder takes an important role in communication system for their functionality of reconstructing original information from the noisy signal.

1.2 Channel Coding

After the theoretical foundation of communications was established in Shannons landmark paper in 1948 [2], practical coding schemes of channel coding are being researched and proposed trying to approach the theoretical maximum channel capacity, known as Shannon Limit. In wireless communication, mostly used channel coding schemes and corresponding decoding technologies are: convolutional code/Viterbi decoder [3], turbo code/turbo decoder [4], and low-density parity-check code (LDPC)/LDPC decoder [5], as well as recently under development polar code/polar decoder [6]. Among these technologies, LDPC are getting more and more attention due to its outstanding performance[7].

Low-density parity-check code (LDPC) [5] is a linear error correcting code invented by Robert Gallager in 1963. It did not draw much attention at that time, an important

reason is its complexity in computation and large number of interconnection in its graph. The implementation of LDPC decoder would cost a lot of hardware resources. As integrated circuit technology has been highly developed, LDPC decoder has been practically implemented in hardware.

Previous research shows that LDPC has its theoretically noise threshold equal to the Shannon limit, and in practical implementation it can also come very close to the limit. Therefore, LDPC code is drawing an increasing amount of attention in scholars and industry [8]. Other than this feature, LDPC code also have the advantage of high parallelization, low decoding latency, and suitable for hardware implementation. Therefore, LDPC was adopted as a part of DVB-S2 satellite digital video broadcasting standard [9], 802.16 WiMax standard [10], and optional part of 802.11n/ac MIMO Wi-Fi standard.

1.3 Problem Overview

Complex interconnections between check nodes and variable node in LDPC code vastly increase the implementation difficulty of parallel decoder. Hence, congestion and long wire connections result in massive routing delay in data path. Although parallel architecture utilized the inherent parallelism of LDPC code, the throughput, power consumption and area/resource utilization is not as satisfactory as expected.

From the other aspect of view, the fixed-point implementation of LDPC decoder has degraded performance compared to the float-point implementation because of lower precision. Also the error floor of fixed-point LDPC decoder is related to the code scheme [11] and precision of LLR messages. For existing standards, the parity check matrices are already formulated. Therefore, a potential optimization for better result without modifying the matrix can be obtained by tuning the precision of LDPC decoder.

1.4 Contribution

The main contribution of this work can be concluded as follow:

1. Summarized and compared the decoding algorithms and architectures of LDPC codes.

2. Implemented fully parallel LDPC decoder in 1152-bit code length and partially parallel decoder in 2304-bit code length, and performed utilization, timing and power analysis of respective architectures.

3. Introduced multi-frame processing architecture to fully parallel decoder by pipelining and out-of-order decoding.

4. Proposed mixed precision decoding and achieved better performance as well as error floor.

All of the implemented decoder are targeted on Xilinx XC7K325T FPGA and followed IEEE 802.16e standard. The multi-frame parallel decoder reaches more than 2x throughput compared to original fully parallel decoder, and the mixed precision decoder achieved around 0.5 dB better frame error rate (FER) and bit error rate (BER) at lower SNR while maintaining a good error floor at high SNR compared to uniform precision decoders.

2. LOW DENSITY PARITY CHECK CODE

2.1 Code Scheme

Low Density Parity Check codes (LDPC) can be written as a sparse parity check matrix H consists of merely ones and zeros. This matrix can also be viewed as a matrix that represent the interconnection of a Tanner graph. Tanner graph or bipartite graph is a kind of graph whose node are divide into two disjoint sets and the edges only connect nodes in different sets. In this case, one set of the nodes are called variable nodes, represent received bits, and the other set of nodes are check nodes, represent parity check equations. As shown in figure 2.1, the columns of the matrix are mapped to the variable nodes in the graph and the rows are corresponded to check nodes.

The LDPC code can be classified to be regular or irregular. Each column of regular LDPC code matrix has equal number of ones, and so does each row. From the Tanner graph aspect of view, the degrees of variable nodes in regular LDPC code are same, and so are the check nodes. So regular LDPC code can be marked as (C, V), where C is the degree of check nodes and V represents the degree of variable nodes. In contrast, the columns or rows of irregular LDPC code has non-uniform numbers of ones, meaning that the degree of check or variable nodes are different. Irregular LDPC codes usually have better performance than regular codes [12], but the implementation of irregular LDPC decoder is more complex.

In this work, a IEEE 802.16e standard parity-check matrix is used for implementation, as shown in table 2.1. The parity check matrix of IEEE 802.16e is described by a n-by-m H matrix. The number of row *m* equals to 24 and the number of column n equals to $m(1 - R)$, where R represents the coding rate, which can be 1/2, 2/3, 3/4, or 5/6. Each element of the n-by-m matrix is a z-by-z sub-matrix, which can either be a cyclic shifted

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

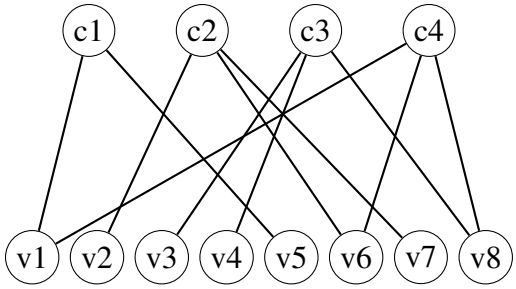

Figure 2.1: Matrix (left) and Tanner graph (right) representation of LDPC code

identity matrix or a zero matrix. The size of z-by-z matrix ranges from 24 to 96 with increment of 4, therefore the code length of WiMax LDPC varies from 576 to 2304 with increment of 96. This kind of code scheme is called quasi-cyclic low-density parity-check (QC-LDPC) code. In this standard, the code scheme is irregular, as shown in table 2.1, the degrees of check nodes are 6 or 7, and the degrees of variable nodes belong to $\{2, 3, 6\}$.

	94	73						55	83			7	0									
	27				22	79	9				12		0	0								
			24	22	81		33				0			0	0							
61		47						65	25						0	0						
		39				84			41	72						0	0					
				46	40		82				79	0					0	0				
		95	53						14	18								0	0			
	11	73				2			47										0	0		
12				83	24		43				51									0	0	
					94		59			70	72										0	0
		7	65					39	49												0	0
43					66		41				26	7										0

Table 2.1: IEEE 802.16e, 1/2 rate H matrix. The numbers are the cyclic shift lengths of identity matrix, and the blanks are zero matrices

2.2 LDPC Decoding

In general, optimal decoding of LDPC code by the maximum-likelihood algorithm is highly complicated and not feasible to implement. Instead, Gallager proposed a sub-optimal iterative soft-decoding algorithm, which is also called belief propagation [13][14] or message passing algorithm, along with the invention of LDPC code in his paper. This algorithm is able to give an excellent result and can be practically implemented. Similar and related decoding algorithms are also independently discovered by other scholars for various of applications.

2.2.1 Bit-Flipping Algorithm

Bit-flipping algorithm is a hard-decision message-passing algorithm. The principle of this algorithm is to flip or invert suspected error bits until the code word satisfy all the parity checks, by assuming that if a bit in the code word fails in most of its related check equations then it is likely to be an error bit [15]. Also to ensure that parity check equations are not concentrated on the same set of bits, the H matrix is supposed to be very sparse. The hard decision of received bits are directly generated by detector front-end and feed to the decoder. Therefore, each variable node begins with a single bit value of either zero or one. During the decoding process, the message passed along the edges of Tanner graph are binaries as well. In each iteration, a variable node broadcasts its binary value to neighboring check nodes. A check node receives the value from its neighboring variable nodes and performs parity check. If the sum of incoming variable nodes modulo by two equals zero (even number of ones) then the check node returns a *check satisfied* signal to neighboring variable nodes, otherwise returns a *check failed*. Based on all return message from its neighboring check nodes, the variable node either flips its value or keeps it unmodified. A flip is happened when the majority of the message received by the variable node is *check failed*. This process is repeated until all parity-check equations are satisfied and output

the results in variable nodes, or until reaching the maximum number of iterations set by decoder designer then the decoder gives up and returns an error code.

2.2.2 Sum-Product Algorithm

In contrast with hard-decision message-passing algorithms like the bit-flipping algorithm, the sum-product algorithm is a soft-decision message-passing algorithm. The single bit message representing each decision is now replaced with a probability value, and the input of variable node becomes the probability of a received bit from channel. This probabilistic decoding algorithm has been described in detail and proved in Gallager's paper [5]. While the probabilistic decoding involves a large number of multiplication of probability values which causes great computation complexity especially in hardware implementation, calculating the probability message in log domain makes it much easier. So in realistic implementations, the log-likelihood ratio (LLR) are used for calculation [16]. With this algorithm, the decoding procedure can be described in following steps:

Step 1. Initialize the log-likelihood ratio of all variable nodes from corresponding received signal:

$$L_n = ln\left[\frac{P(x_n = 0|y_n)}{P(x_n = 1|y_n)}\right] \tag{2.1}$$

Where x_n represents the *n*th bit of original signal $X(x_1, x_2, ..., x_n)$, and y_n represents the *n*th bit of received signal $Y(y_1, y_2, ..., y_n)$.

For a channel with additive white Gaussian noise (AWGN) which variance is σ^2, equation 2.1 can be written as:

$$L_n = \frac{2y_n}{\sigma^2} \tag{2.2}$$

Step 2. Propagate the LLR messages from variable nodes to their corresponding neighboring check nodes along the edges of the graph.

Step 3. Update the LLR message from check node side by equation:

$$R_{mn} = \Psi^{-1}\left[\sum_{n' \in N(m)\backslash n} \Psi\left(Q_{n'm}\right)\right] \cdot \sigma_{mn} \tag{2.3}$$

Where

$$\Psi(x) = -ln\left|\tanh\left(\frac{x}{2}\right)\right| \tag{2.4}$$

$$\sigma_{mn} = \prod_{n' \in N(m)\backslash n} sgn\left[Q_{n'm}\right] \tag{2.5}$$

And $N(m)$ is the set of variable nodes connected to check node *m*.

Step 4. Check nodes return the updated LLR message to variable nodes along the edges of graph.

Step 5. Update the LLR message at variable nodes by the summation of log-likelihood of received bit and returned LLR message from neighboring check nodes $M(n)$.

$$Q_{nm} = L_n + \sum_{m' \in M(n)\backslash m} R_{m'n} \tag{2.6}$$

And also update the soft decision

$$P_n = L_n + \sum_{m \in M(n)} R_{mn} \tag{2.7}$$

From step 2 to step 5 is one complete decoding iteration. At the end of each iteration, the hard-decision is generated by the sign of soft decision.

$$\hat{x}_n = \begin{cases} 1, & P_n < 0 \\ 0, & otherwise \end{cases} \tag{2.8}$$

The iteration will repeat until the parity check $\hat{x}H^T = 0$ is satisfied, or reached the maximum iteration limit set by decoder designer.

2.2.3 Min-Sum Algorithm

By looking into equation 2.4, it can be found that the value of $\Psi(x)$ decreases rapidly as $|x|$ increases, as shown in figure 2.2. Therefore, equation 2.3 is dominated by the minimum value of $|Q_{n'm}|$. So equation 2.3 can be simplified by approximate the sum of Ψ function to the minimum $|Q_{n'm}|$ value:

$$R_{mn} = \left(\min_{n' \in N(m) \backslash n} |Q_{n'm}| \right) \cdot \sigma_{mn} \tag{2.9}$$

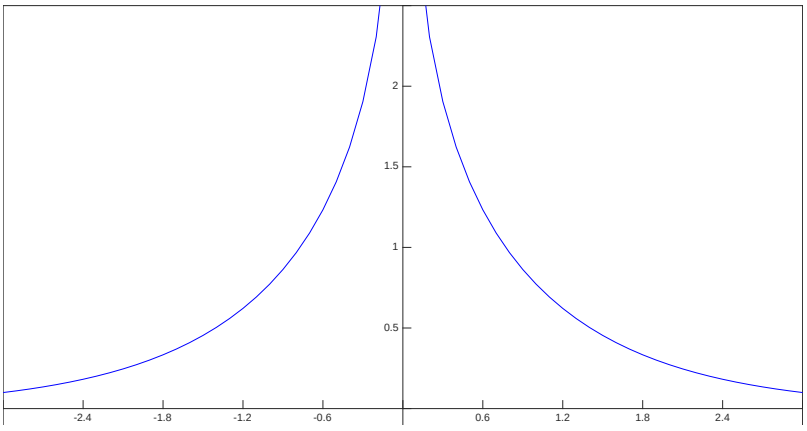

Figure 2.2: Function $\Psi(x) = -ln \left| \tanh \left(\frac{x}{2} \right) \right|$

The min-sum algorithm simplified the check node processing of the sum-product algorithm, while other steps remain the same. Lower computation complexity is helpful in the respect of implementation. However, because of the approximation as described above there is a performance degradation compared to the sum-product algorithm. It is known that scaling down or offset the LLR value is helpful in improving the performance of min-sum algorithm [17], the corresponding algorithm are called normalized min-sum (NMS) and offset min-sum (OMS) algorithm [18].

In OMS algorithm, $|Q_{n'm}|$ in equation 2.9 is offset by a positive constant β depending on the code parameters [19][20]. By carefully select the offset constant, the OMS algorithm can have a performance close to the sum-product algorithm.

$$R_{mn} = \sigma_{mn} \cdot \max\left(\kappa_{mn} - \beta, 0\right) \tag{2.10}$$

$$\kappa_{mn} = \min_{n' \in N(m) \backslash n} |Q_{n'm}|$$

On the other hand, a scaling factor α is applied after variable node update in the NMS algorithm. Related research indicated that the optimal scaling factor can be found by density evolution techniques [21]. In hardware implementation a scaling factor of 0.75 is usually selected, providing that it can be easily implemented by adding and shifting circuit while still provide a performance close to optimal.

$$Q_{nm} = \alpha \cdot \left(L_n + \sum_{m' \in M(n) \backslash m} R_{m'n} \right) \tag{2.11}$$

3. FULLY PARALLEL LDPC DECODER DESIGN

3.1 Introduction

Various kind of implementation of LDPC decoder have been proposed in research by many scholars. Based on the parallelism and the mapping from the Tanner graph to hardware architectures, the implementation of LDPC decoder can be classified as fully parallel architecture, partially parallel architecture, and fully serial architecture.

Table 3.1 compared the three kinds of implementations. Fully parallel architecture is the most straightforward implementation of LDPC decoding algorithms. Each check node in Tanner graph is mapped to a check node unit (CNU), each variable node is implemented as a variable node unit (VNU) and edges in the graph are physically connected by wires. So each decoding iteration can be completed in a single clock cycle, and LLR messages are stored in flip-flops distributed in each VNU or CNU. Fully parallel architecture usually has high throughput but also large resource utilization and great difficulty in floor planning and routing[22].

Fully serial architecture has the fewest resource utilization because it has only one variable node unit (VNU) and one check node unit (CNU) [23]. In each clock cycle, a VNU or CNU processes the LLR update of a single node in the Tanner graph. Therefore, the speed of fully serial decoding is low. Because of this feature, the fully serial approach is suitable for relative shorter code length and applications that have a limited logic resource.

Partially parallel architecture is an implementation between fully parallel and fully serial [24]. It trades-off between area and time[25] according to a specific design requirement by reuses CNU and VNU units throughout decoding iterations and process LLR updates in several clock cycles.

In this chapter, technical details and the analysis of implementing fully parallel decoder

architecture will be explained.

Spec.	Fully Parallel	Partially Parallel	Fully Serial
Code Length	N	N	N
Code Rate	R	R	R
Fold factor	1	F_c, F_v	N
No. of VNUs	N	N/F_v	1
No. of CNUs	$N(1-R)$	$N(1-R)/F_c$	1
Clocks Per Iteration	1	$F_c + F_v$	$N(2-R)$
Memory Type	Distributed flip-flops	Distributed Memory Blocks	Single Memory Block

Table 3.1: Comparison of LDPC decoder architectures

3.2 Architecture

As figure 3.1 shows, the fully parallel decoder consists of a VNU array and CNU array, corresponding to variable nodes and check nodes. The implementation of VNUs and CNUs are in combinational logic. And an array of memory is used to store intermediate LLR messages during iterations. The memory is implemented by flip-flops distributed in nodes. Specifically in this design the flip-flops are distributed at the output of CNUs.

3.3 Check Node Unit (CNU)

Since in fully parallel architecture each check node is directly mapped to a CNU, the main function of CNU is to calculate and update LLR message base on equations men-

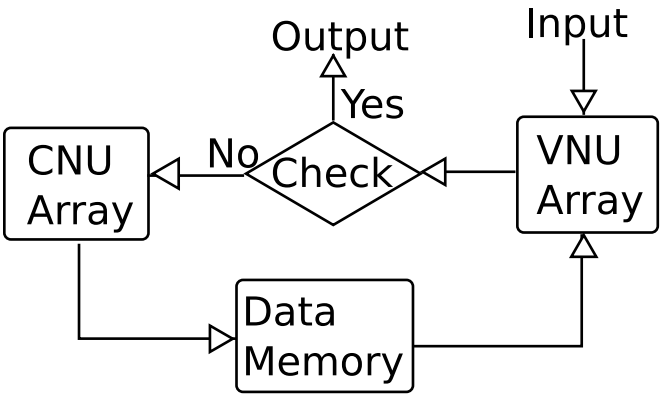

Figure 3.1: Fully parallel architecture diagram

tioned in Chapter 2, specifically equation 2.3 for the sum-product algorithm and equation 2.9 for the min-sum algorithm. Considering that compared to the sum-product algorithm, the normalized min-sum algorithm has much lower computation complexity while still have a similar performance, the normalized min-sum algorithm is implemented in this work.

In the normalized min-sum algorithm, each CNU needs to complete the following tasks:

1. Find the minimum absolute value of input LLRs excluding the LLR from destination variable node

2. Calculate the sign of each output

3. Update output data by combining the result from task 1 and 2.

For task 1, a comparator network is required if implement (2.9) directly in hardware, which cost large amount of logic resources. The equation can be simplified for hardware implementation by defining Min and $SubMin$ representing the minimum and the second minimum value in all input of this CNU, and equation (2.9) is now equivalent to:

$$Min_m = \min_{n' \in N(m)} |Q_{n'm}|$$

$$SubMin_m = \quad 2nd\min_{n' \in N(m)} |Q_{n'm}|$$

$$\kappa_{mn} = \begin{cases} SubMin_m, & n = IndexOfMin \\ Min_m, & otherwise \end{cases} \tag{3.1}$$

$$R_{mn} = \kappa_{mn} \cdot \sigma_{mn} \tag{3.2}$$

The micro architecture of parallel CNU can now be described as figure 3.2. The inputs are LLR messages Q_{nm} from neighboring variable nodes. Firstly, all the inputs are converted from 2's complement form to sign-magnitude form. The magnitude part goes through comparators to find the minimum and second minimum value as well as the index of the minimum value. Then base on the index of the minimum value, magnitudes of results are selected by a group of multiplexer, as equation 3.1. At the same time, the signs of inputs are XOR-ed together to get the sign of results. And lastly, the magnitude of result and the sign are converted back to 2's complement form to be the updated LLR.

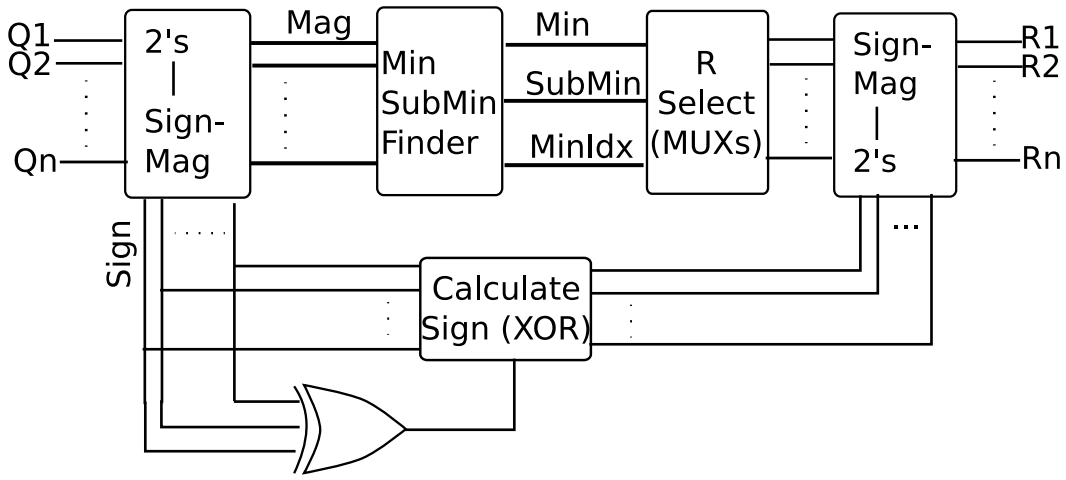

Figure 3.2: Parallel CNU micro architecture

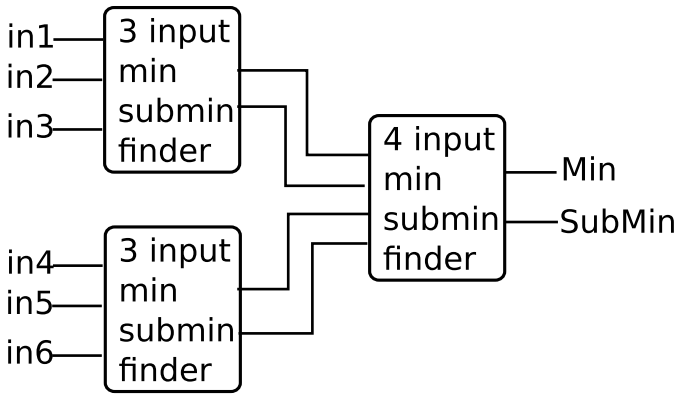

Figure 3.3: Minimum and second minimum value finder

The finding of minimum and second minimum value is the crucial part of CNU design, a well-designed finding circuit will result in better performance and resource utilization. In this work, an idea that similar to the merge-sort algorithm is implemented. For example, to find the minimum and second minimum value from 6 inputs, the diagram is shown as figure 3.3. The 3-input finder uses 3 comparators to find minimum and sub-minimum number in its input. The 4-input finder's inputs are connected to the outputs of two 3-input finders, so its $input_1 < input_2$ and $input_3 < input_4$ is known. Therefore, only 3 comparisons (compare $input_1$ with $input_3$, $input_1$ with $input_4$, and $input_2$ with $input_3$) are needed to find the final minimum and sub-minimum value. So totally 9 comparators are used in this example.

3.4 Variable Node Unit (VNU)

In the implementation of the normalized min-sum algorithm, VNU handles the LLR message and decision update at variable nodes base on equation (2.11), (2.7), and (2.8). More specifically, in hardware the summation of input LLR value and the message received from neighboring CNUs is calculated by using an adder tree to produce the soft-decision. Then subtract each input respectively for LLR message updating. And also get

the new hard-decision by taking the most significant bit (MSB) of soft-decision. Targeted on Xilinx XC7K325T FPGA, the adder tree is implemented by ternary (3-input) adders for better resource utilization, as shown in figure 3.4.

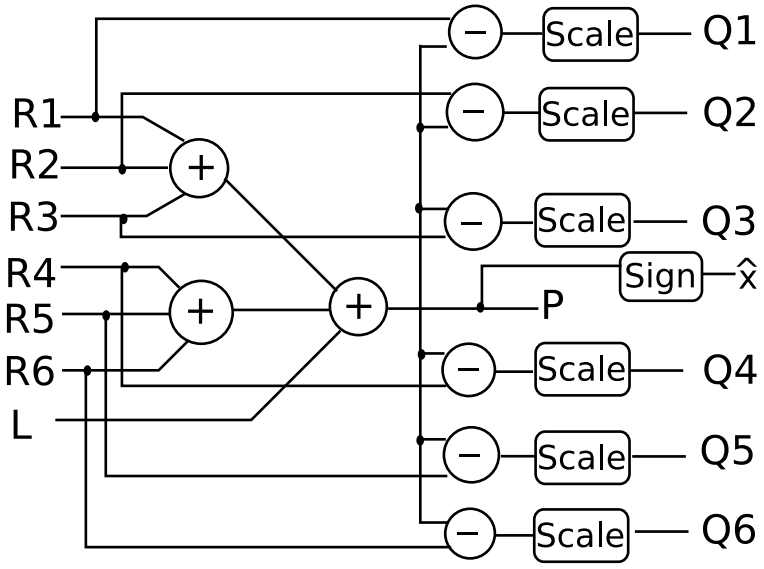

Figure 3.4: Parallel VNU micro architecture

The scaling factor α in equation 2.11 is set to 0.75 as explained in Chapter 2. The scaling can be done by two right shifts and one addition in hardware. Moreover, because of the scaling, two more fractional bits are added to the result which requires an additional rounding operation for precision adjustment. By further logic simplification, the rounding operation can be achieved by simply feed the OR of last two input bits to carry-in of the adder, as shown in figure 3.5.

On the other hand, each updated LLR value in VNU is the sum of inputs. So there are also extra bits at most significant side. In order to keep the bit width of LLR message, saturating units are connected after scaling. Therefore, if the scaled sum still exceeds designed bit width, it will be saturate to the maximum allowed value.

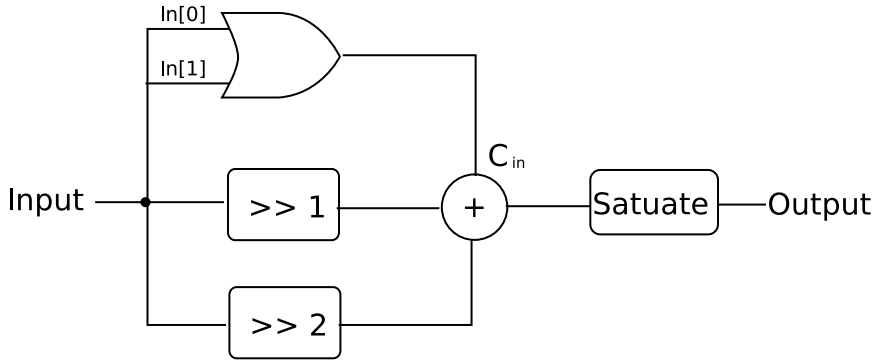

Figure 3.5: scaling unit with rounding and saturate

3.5 FPGA Result

Based on the architecture described as above, a IEEE802.16e standard, 1152-bit, 1/2 rate fully parallel decoder is implemented on Xilinx XC7K325T FPGA. The utilization, timing and power are analyzed as follow.

3.5.1 Utilization Analysis

As shown in table 3.2, look-up table (LUT) utilization ratio is very high compared to that of flip-flops (FF). This is because of the architecture of fully parallel decoder. The most part of this architecture are combinational logic circuits for implementing CNU and VNU array. Only a small part of memory (flip-flops) is used for holding the data after each iteration.

3.5.2 Timing Analysis

Large combinational logic blocks in the circuit combining with complex routing of interconnection network results in congestion and massive delay. Table 3.3 is the timing of a critical path starting from 97th CNU going through 22 levels of logic consisting of VNUs and interconnections and end up at the 61th CNU. Noticed that the total data path delay is 13.586ns and logic delay only takes 14.4% while the 85.6% part is caused by routing.

Resource	Utilization	Available	Utilization%
FF	54903	407600	13.47
LUT	145662	203800	71.47
Memory LUT	1122	64000	1.75
I/O	2	500	0.40
BRAM	2	445	0.45
DSP48	144	840	17.14
BUFG	4	32	12.50
MMCM	1	10	10.00

Table 3.2: Utilization summary of IEEE802.16e standard 1152-bit, 1/2 rate fully parallel LDPC decoder on Xilinx XC7K325T FPGA

So it can be concluded that the data path delay of fully parallel architecture is dominated by the interconnection network that establishes the Tanner graph. Although fully parallel decoder maximized the use of intrinsic parallelism of LDPC code, huge data path delay brings down the frequency and reduces throughput. Therefore, the main challenge of fully parallel decoder design is to perform meticulous floor planning to avoid congestion and to develop efficient wiring strategy to minimize routing delay.

In summary, this implementation of fully parallel LDPC decoder works at 70MHz clock frequency. Measured at 3.0 dB SNR, the average clock cycle to decode one frame is 13 cycles. So the average throughput is around 6Gpbs.

Source	DUT/LDPC/CNU97
Destination	DUT/LDPC/CNU61
Path Type	Setup
Requirement	14.000ns
Data Path Delay	13.586ns (logic 1.961ns (14.434%) route 11.625ns (85.566%))
Logic Levels	22
Clock Path Skew	-0.465ns
Clock Uncertainty	0.132ns

Table 3.3: Critical path delay of implemented fully parallel decoder

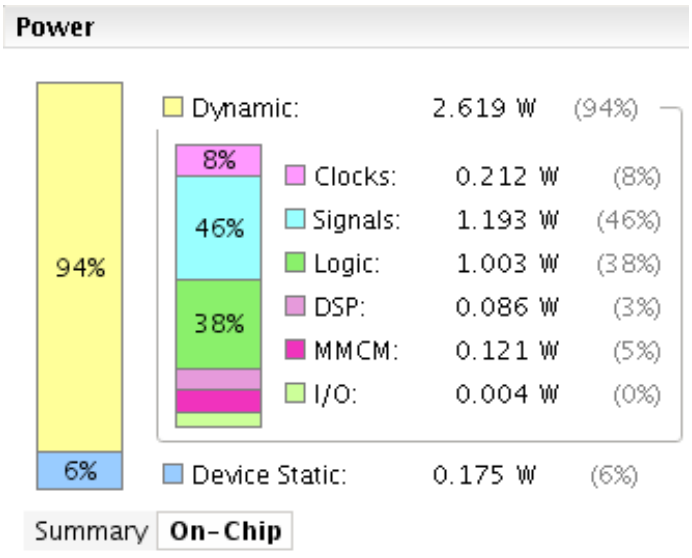

Figure 3.6: On-chip power of IEEE802.16e standard 1152-bit, 1/2 rate fully parallel LDPC decoder on Xilinx XC7K325T FPGA

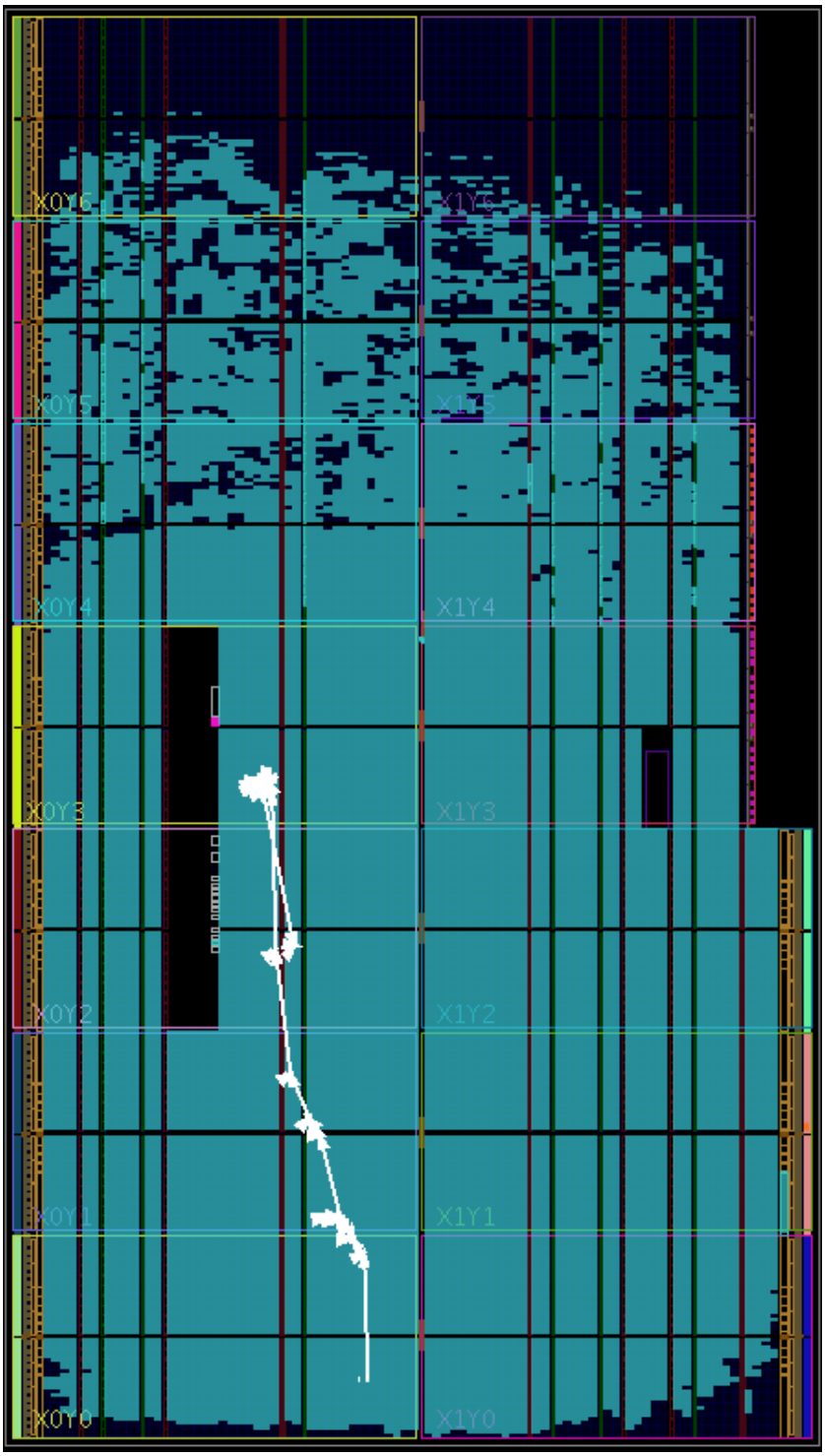

Figure 3.7: FPGA layout of IEEE802.16e standard 1152-bit, 1/2 rate fully parallel LDPC decoder on Xilinx XC7K325T FPGA with critical path marked

4. PARTIALLY PARALLEL LDPC DECODER DESIGN

4.1 Introduction

Although fully parallel architecture maximized the inherent parallelism of LDPC code, it is difficult to implement fully parallel decoder for longer code length (2K or more) even with good floor planning and routing due to the complexity of interconnection and large resource utilization, as discussed in Chapter 3. In order to achieve longer code length, architectures such as bit-serial decoder[26], layered decoder[27], and partially parallel decoder are being researched by scholars. In this chapter the design of partially parallel decoding architecture is discussed in detail.

4.2 Architectural Design

In partially parallel architecture, each CNU or VNU handles multiple check nodes or variable nodes in Tanner graph in different clock cycles. This is done by dividing data memory in groups, one group is selected for processing in each clock cycle, as shown in figure 4.1.

In this work, a IEEE 802.16e standard partially parallel decoder of code length 2304-bits is implemented. The data memory of VNU and CNU are both divided to two groups based on even and odd address (fold factor $F_c = 2$ and $F_v = 2$). Therefore, theoretically this implementation should have a resource utilization close to 1152-bit fully parallel decoder in Chapter 3. So the number of CNUs and VNUs are equal to the number of check nodes and variable nodes divided by fold factor (1152 VNUs and 576 CNUs in this design). Since CNUs and VNUs access the memory at different clock cycles, a memory access control unit is implemented at the form of finite state machine. In each iteration, the processing procedure can be described as following steps:

Step 1. Memory access control unit set R memory status as reading, Q memory status as writing, and switch data select MUX/DEMUX at VNU array to group A.

Step 2. VNU array process data group A from R memory and write back to group A of Q memory. Data select switch to group B and perform same operation as group A

Step 3. Memory access control unit set Q memory status as reading, R memory status as writing, and switch data select MUX/DEMUX at CNU array to group A.

Step 4. CNU array process data group A from Q memory and write back to group A of R memory. Data select switch to group B and perform same operation as group A

Step 5. Increase iteration count, and repeat step 1 to step 5 until parity check satisfied or reach maximum allowed iteration which is set to 32 in this design.

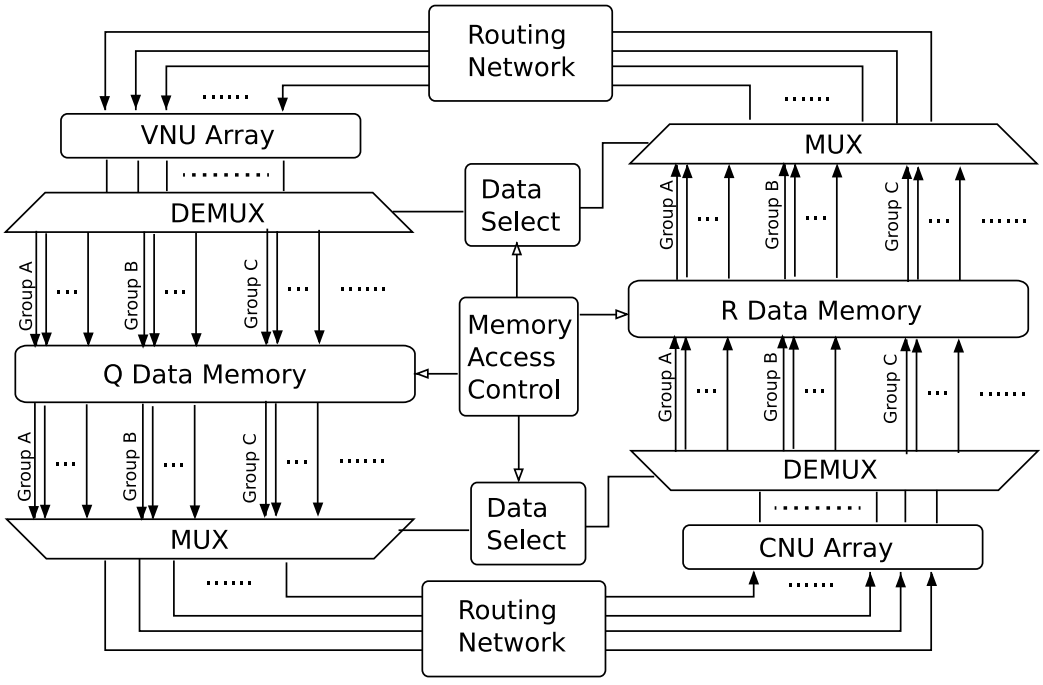

Figure 4.1: Partially parallel decoder micro architecture

4.3 FPGA Result

As shown in table 4.1, as expected, the LUT utilization of 2304-bit partially parallel decoder with folding factor of 2 at both CNU and VNU side is close to that of 1152-bit fully parallel decoder in Chapter 3.

Resource	Utilization	Available	Utilization%
FF	129712	407600	31.82
LUT	153600	203800	75.37
Memory LUT	640	64000	1.00
I/O	2	500	0.40
BRAM	2	445	0.45
DSP48	72	840	8.57
BUFG	6	32	18.75
MMCM	1	10	10.00

Table 4.1: Utilization summary of IEEE802.16e standard 2304-bit, 1/2 rate partially parallel LDPC decoder on Xilinx XC7K325T FPGA

The number of interconnecting wires between CNU array and VNU array is divided by the folding factor as well. So, as table 4.2 shows, the data path delay is also close to the fully parallel decoder in Chapter 3.

In summary, the resource utilization of partially parallel decoder is related to the folding factor of CNU array and VNU array. When folding factor increases, the usage of logic resource and the number of interconnection decreases. Less logic and wiring usually re-

sults in lower delay. On the other hand, higher folding factor means more clock cycles are needed per iteration. So by trade-off between resource, data path delay, and clock cycles per iteration, optimal folding factor can be found for better overall throughput or to meet specific specification requirement. Specifically, in this work by choosing folding factor of 2 for CNU and VNU, a frequency of 71Mhz is achieved. Measured at 3.0dB SNR, an average of 10 iterations or 40 clock cycles are need for decoding of a frame. Therefore, the throughput of this design is around 4GBps.

Source	DUT/LDPC/FSM
Destination	DUT/LDPC/CNU39
Path Type	Setup
Requirement	14.000ns
Data Path Delay	13.697ns (logic 1.363ns (9.951%) route 12.334ns (90.049%))
Logic Levels	17
Clock Path Skew	-0.141ns
Clock Uncertainty	0.132ns

Table 4.2: Critical path delay of partially parallel decoder

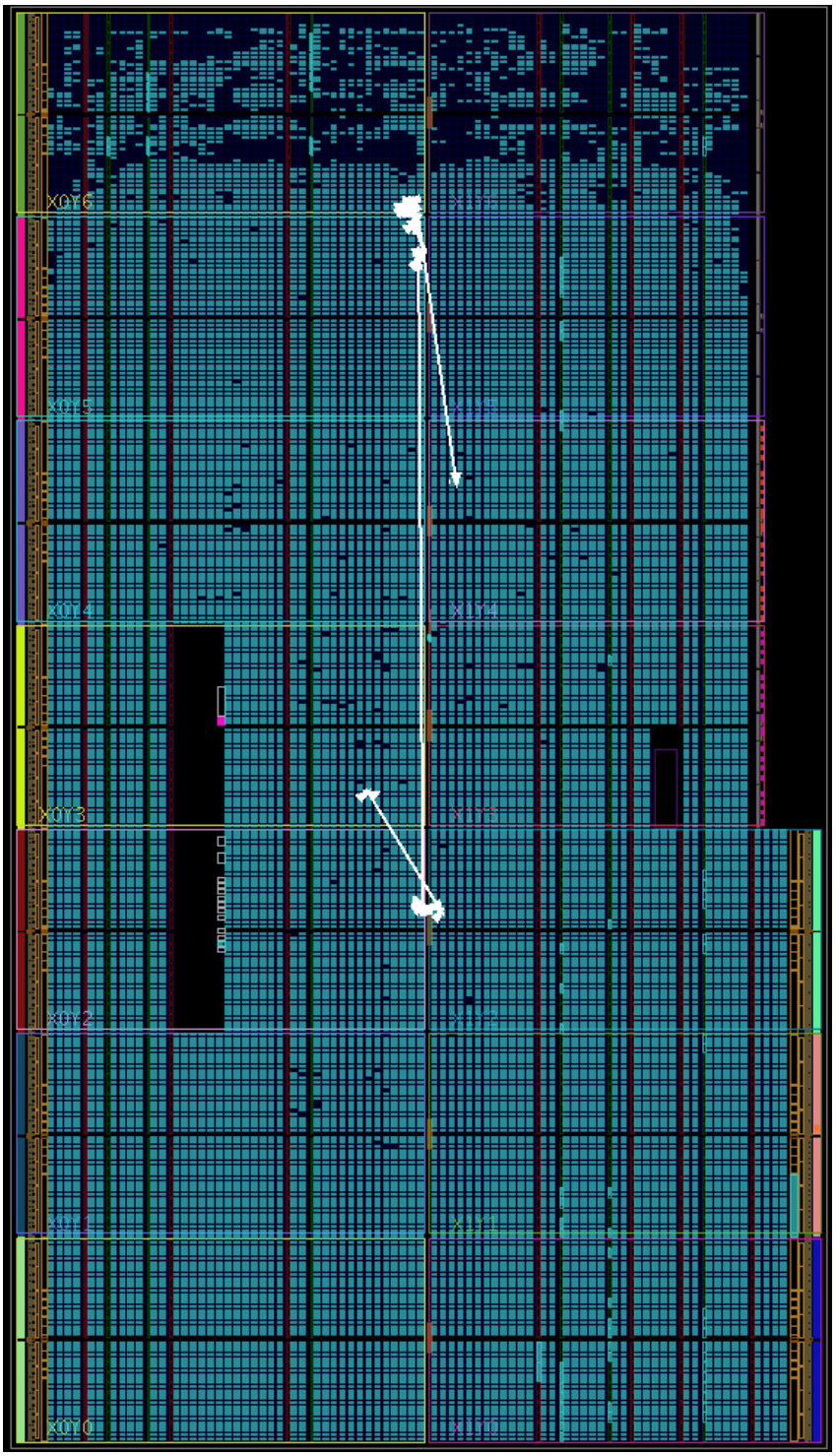

Figure 4.2: FPGA layout of IEEE802.16e standard 2304-bit, 1/2 rate partially parallel LDPC decoder on Xilinx XC7K325T FPGA with critical path marked

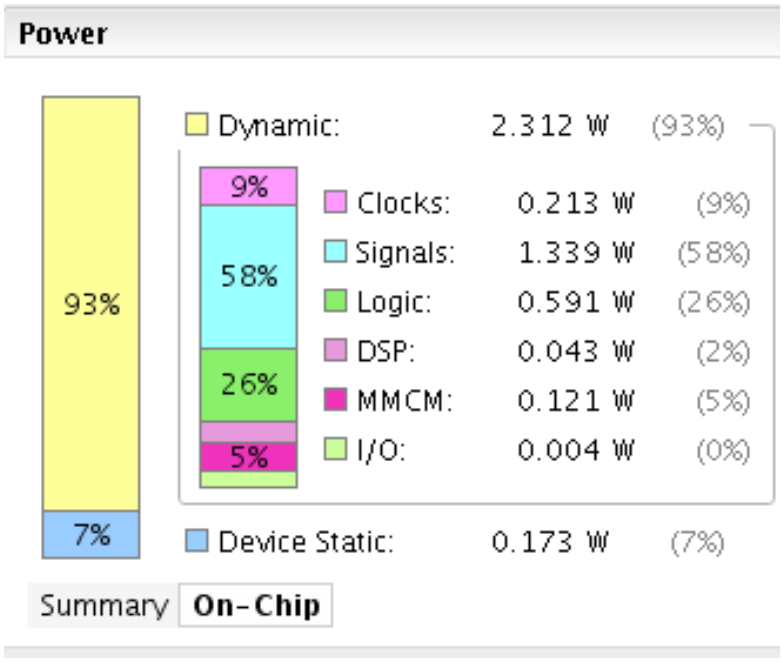

Figure 4.3: On-chip power of IEEE802.16e standard 2304-bit, 1/2 rate partially parallel LDPC decoder on Xilinx XC7K325T FPGA

5. MULTI-FRAME PROCESSING

5.1 Introduction

According the analysis in previous chapters, the main issue of fully parallel decoder is the large interconnection network for it to build the Tanner graph. Therefore, despite fully parallel architecture maximized the inherent parallelism of LDPC code, large data path delay results in low clock frequency which in turn limits the overall throughput.

Pipelining is one of the most common solution for achieving higher clock frequency and throughput. However, because of the iterative feature and data dependency, pipeline stages in LDPC decoder has to stall until all depended data and LLR messages get updated. Noticed that generally two different frames do not have any dependency in between. Therefore, instead of stall, more frames can be loaded into decoder and decoding simultaneously. Detailed architectural design of multi-frame processing decoder is explained in following sections.

5.2 2-Stage Design

According to the two phases of message passing algorithm, an obvious way to divide the pipeline stages is one for CNU update and another stage for VNU update. This can be implemented as the architecture described in the following subsection.

5.2.1 Architecture

Figure 5.1 demonstrates a 2-stage pipelined fully parallel decoder architecture diagram. In this design, the main architecture of the decoder is similar to fully parallel decoder except that instead of using a single array of memory, this design has memory in both CNU and VNU, and therefore form a two stage pipeline. Besides VNUs and CNUs, there are also another two memory arrays for holding two input frames. So at each clock

cycle one of the input frame, e.g. frame A, involves the calculation at VNUs, while at the same time check node update of frame B is completed by CNUs. Then at second clock cycle frame selection switch to frame B, and VNUs update the variable nodes of frame B, also at the same time, the check nodes of frame A are updated by CNU. By switching frame selection back and forth between the two frames at same pace of decoder pipeline, these two frames can be decoded simultaneously at different pipeline stages and therefore avoided stall.

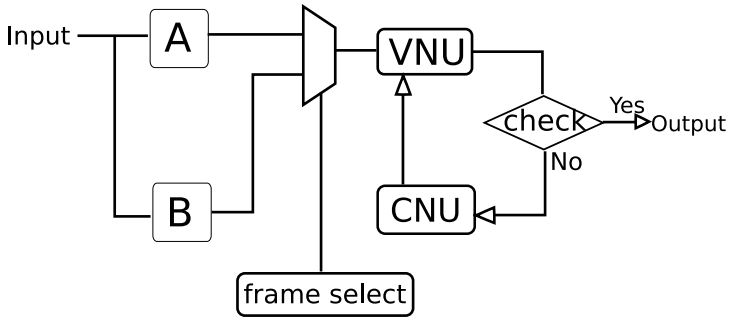

Figure 5.1: 2-stage decoder diagram

5.2.2 FPGA Result

Table 5.1 is the critical path delay of the 2-stage design. Noticed that the data path delay is just slightly better than the original fully parallel decoder. By examine the FPGA layout and routing in figure 5.2, it can be found that the wire length of critical path is still very long. It turns out the 2-stage pipeline dividing solution may not be the optimal, seeing that this critical path from VNU to CNU has only a few logic cells at the begin and followed by long wire, then it ends up at CNU logic. Therefore, adding one more stage before CNUs is tend to be a better solution.

Besides the decoder pipeline, there are also exist potential issues at data memory side.

Both input frames memories A and B are required to close to VNUs, or either of them will become the bottle neck. Also, the frame selection switches at every clock cycle. So for a long code length, the selecting unit would have large fan-out which makes it difficult to achieve short delay time to meet the requirement.

Source	DUT/LDPC/VNU582
Destination	DUT/LDPC/CNU3
Path Type	Setup
Requirement	11.000ns
Data Path Delay	10.140ns (logic 1.252ns (12.347%) route 8.888ns (87.653%))
Logic Levels	12
Clock Path Skew	-0.719ns
Clock Uncertainty	0.093ns

Table 5.1: Critical path delay of IEEE802.16e standard 1152-bit, 1/2 rate, 2-stage multi-frame decoder

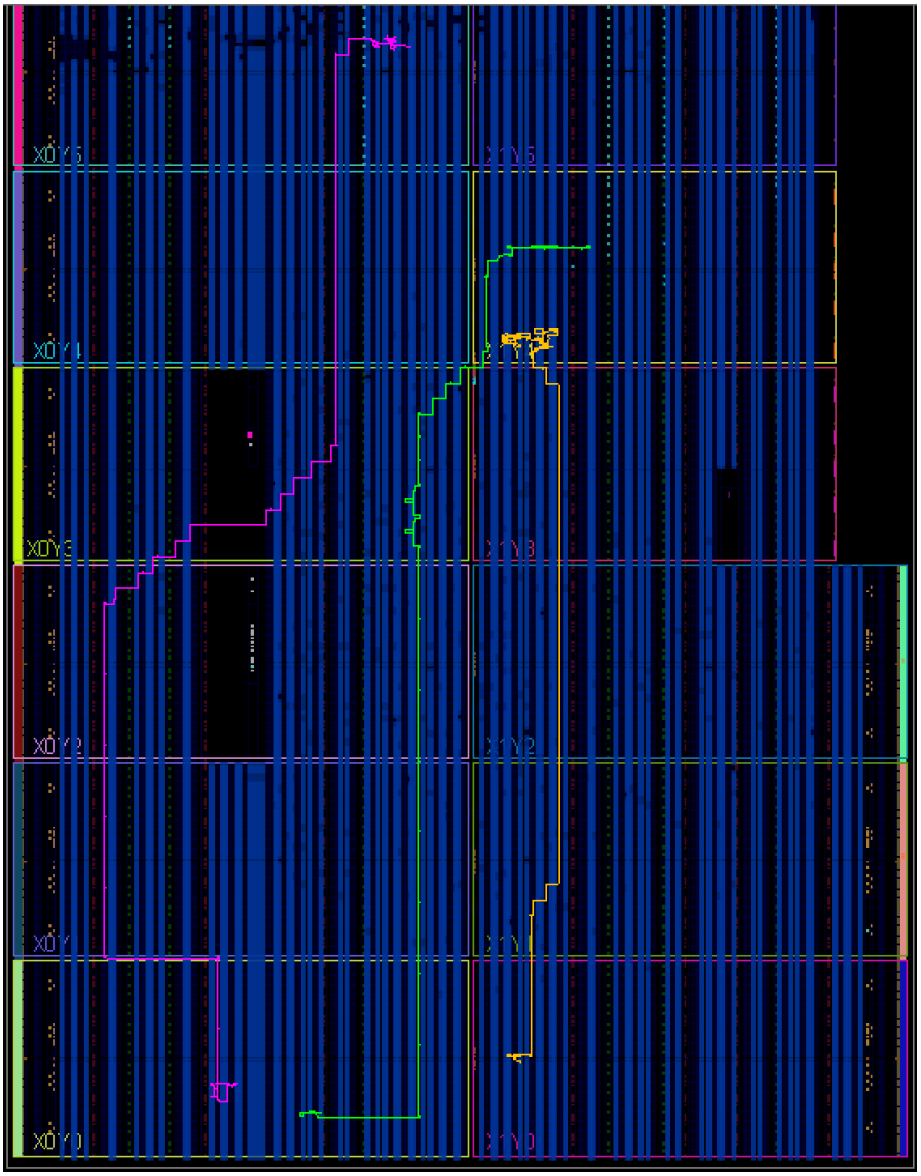

Figure 5.2: Critical paths in 2-stage design, marked in cyan, green and yellow

5.3 3-Stage Design

Based on the analysis in previous section, a 3-stage pipeline structure is implemented. Also data memory part is redesigned to a better architecture.

5.3.1 Architecture

As demonstrated in figure 5.3, the pipeline stages are: 1) VNU arrays. 2) CNU stage 1, formed by scaling and saturating unit, 2's complement to sign-magnitude conversion, and wire from VNU to CNU. 3) CNU stage 2, contains minimum/sub-minimum value finder, MUXs, and sign-magnitude to 2's complement conversion. At data memory side, memory of the three frames forms a circle so that the data moves at the same pace as decoder pipeline. By doing this, it ensures the data dependency between VNU and data memory will always satisfied. Moreover, compared to 2-stage design, it has only one data path from memory A to VNU. So it is easier to get a lower net delay. Additionally, in this design control signal paths are separated from data path. At each clock cycle, from memory A to B only 6-bit iteration counter updates, from CNUs to memory C only 1-bit "empty frame" mark updates. Therefore, it avoids the high fan-out situation in 2-stage design.

5.3.2 FPGA Result

As shown in figure 5.4, instead of crossing multiple clock regions as designs in previous sections, the critical path in 3-stage design is in one clock region. Therefore, a much shorter data path delay can be achieved, as shown in 5.2. Moreover, the logic resource utilization ratio of this design does not increase too much, shown in table 5.3.2. Because although the design has one more array of memory for an additional frame, the data memory part can be implement purely by cyclic shift registers which require no additional LUTs. In conclusion, the implemented 3-stage multi-frame decoder architecture can reach

145MHz frequency, and around 12.6Gbps throughput.

Source	DUT/LDPC/lrd_reg
Destination	DUT/LDPC/la_reg
Path Type	Setup
Requirement	7.143ns
Data Path Delay	6.706ns (logic 0.302ns (4.504%) route 6.404ns (95.496%))
Clock Path Skew	-0.295ns
Clock Uncertainty	0.119ns

Table 5.2: Critical path delay of 3-stage multi-frame decoder architecture

5.4 Result Comparison

Table 5.4 summarized the 3 designs: original fully parallel decoder, 2-stage design, and 3-stage design. From the table we can conclude that by pipelining the parallel decoder, the clock frequency can be significantly increased. And combining with multi-frame processing architecture, the 3-stage design more than doubled the throughput compared to the original fully parallel decoder. While on the other hand, because of frequency increase, power consumption grows rapidly. Therefore, this design is more suitable for applications emphasis more on performance.

Resource	Util.	Available	Util.%
FF	116337	407600	28.54
LUT	152897	203800	75.02
Mem LUT	1123	64000	1.75
I/O	3	500	0.60
BRAM	2	445	0.45
DSP48	144	840	17.14
BUFG	4	32	12.50
MMCM	1	10	10.00

Table 5.3: Utilization of IEEE802.16e standard 1152-bit, 1/2 rate, 3-stage fully parallel LDPC decoder with multi-frame processing on XC7K325T FPGA

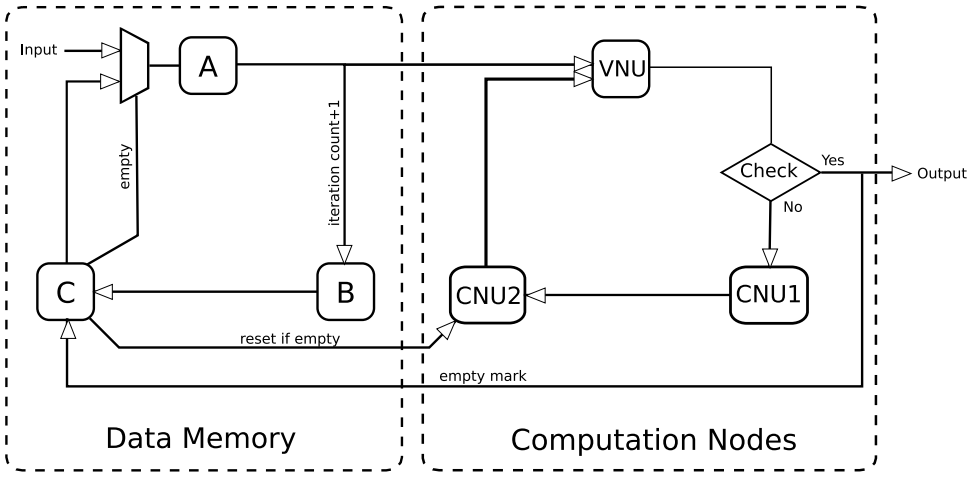

Figure 5.3: 3-stage decoder diagram

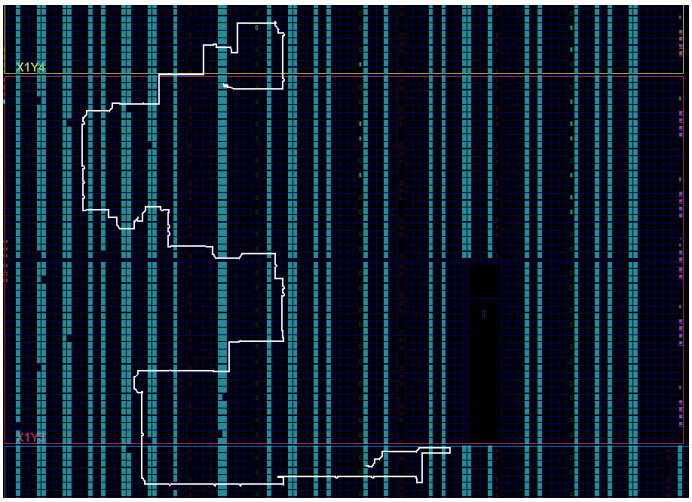

Figure 5.4: Critical path of 3-stage multi-frame decoding architecture

	Full Parallel	2-Stage	3-Stage
Frequency (MHz)	70	100	145
Throughput (Gbps)	6.0	8.7	12.6
LUT%	71.47	73.33	75.02
FF%	13.47	21.82	28.54
Power(W)	2.62	2.16	6.03

Table 5.4: FPGA result summary

6. PRECISION MIXING

6.1 Introduction

Previous related research by J. Xu et al [28] indicated that more significant bits in LLR calculation result in good performance at lower SNR, but relatively higher error floor at high SNR. On the other hand, more less significant bits in LLR calculation leads to lower error floor, but obscuring the confidence of LLR degrades low SNR performance. To achieve better performance without degrading the noise floor, a design that mixes different precision in check node or variable nodes is proposed.

6.2 Precision Control and Alignment

In order to achieve precision mixing, a set of precision control units and precision alignment units are added before and after CNUs, as shown in figure 6.1. In this design the precision of VNU is uniform, which is set to the higher precision in the mix. So the LLR messages passed from VNU to CNU also has the data width of higher precision. The precision control units are rounding up circuits set to cut the fractional bits of messages from VNUs to a given precision. So before entering CNUs with lower data width, the LLR messages are rounded up to fit the data width of CNUs. For instance, the data width in the VNU of a decoder with a mixed precision of 6-bit and 5-bit is chosen to be 6 bit. After summation and scaling, the 6-bit output LLR message goes from VNU to precision control units. For 6-bit CNUs, precision control units leave the LLR message unchanged, while for 5-bit precision CNUs, precision control units round up the last bit of LLR message and produces a 5-bit input to these CNUs.

For fixed point numbers with different precision, the decimal points should be aligned before adding. This is achieved by precision alignment units which adds zeros at the end

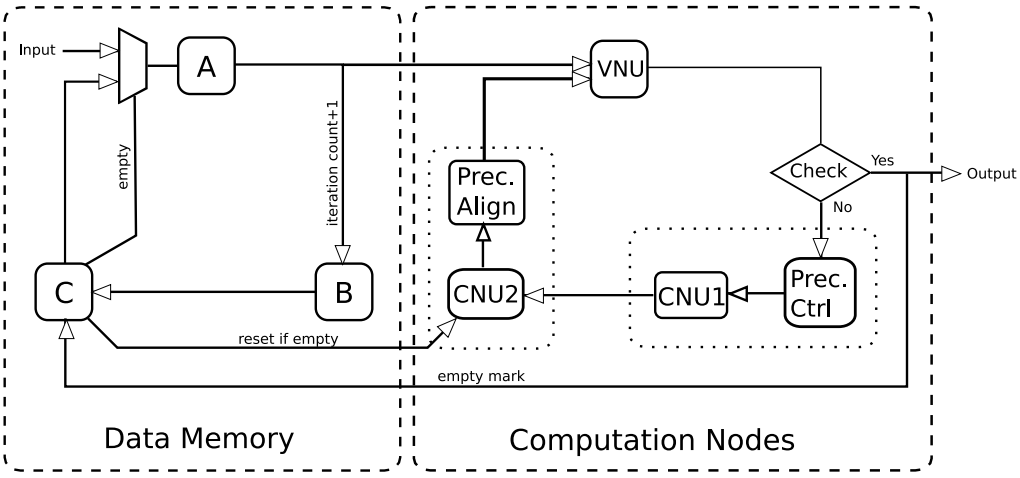

Figure 6.1: Diagram of 3-stage parallel LDPC decoder with precision control and alignment units

of CNU output if the CNU has lower precision so that all the LLR message entering VNUs are aligned to the higher precision LLR messages.

6.3 FPGA Result

For the implementation in this work, one of the precision is 6-bit which has 5 integer bits and 1 fractional bit, notation as (5,1). Another precision is 5-bit, all of which are integer bits, notation as (5,0). Two precision are mixed at 1:1 ratio at CNU side, and (5,0) CNUs are chosen randomly. All VNUs output at (5,1) precision, then precision control units round up (5,1) to (5,0) for 5-bit CNUs.

Figure 6.2 shows the comparison of results from FPGA. At lower SNR, the mixed precision frame error rate (FER) and bit error rate (BER) are significantly better than decoders with uniformly (5,0) and (5,1) precision. At high SNR, the FER and BER of mixed precision decoder is getting close to uniformly (5,1) precision decoder which is the better side of the curve. Therefore, we can conclude that by precision mixing at check nodes, the performance at lower SNR can be improved without degrade error floor at high SNR.

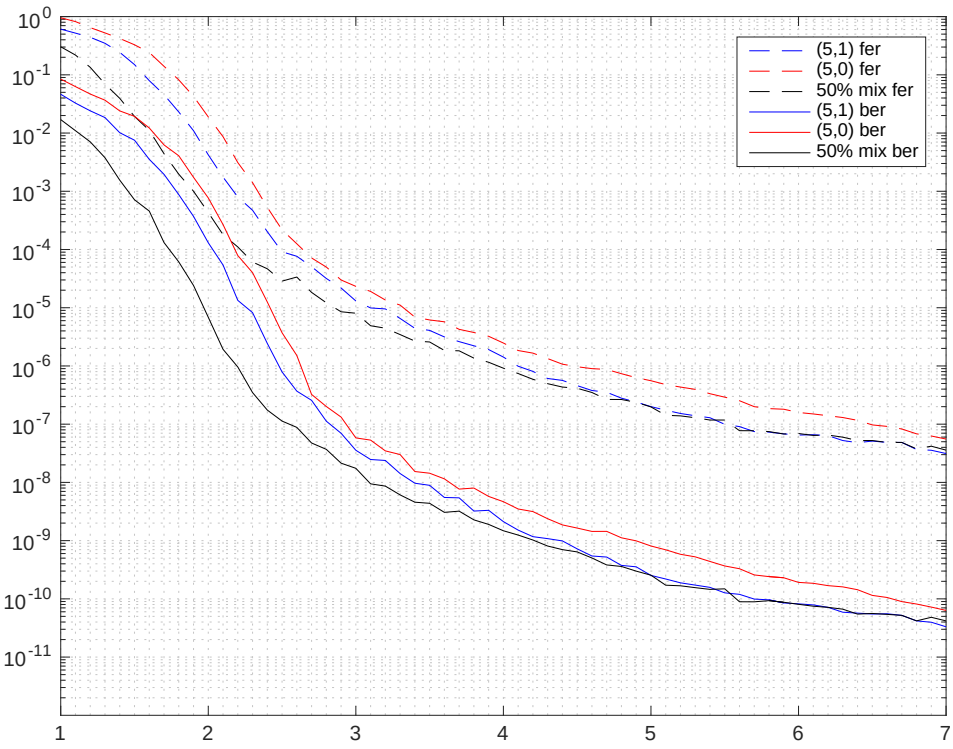

Figure 6.2: FER and BER curve of different precision and mixed precision decoder with respect to SNR

7. SUMMARY

7.1 Summary

This thesis presented a systematic design of parallel LDPC decoder architecture. Multi-frame processing and precision mixing design approaches are also proposed to increase throughput as well as improve performance and error floor.

Chapter 2 concluded and compared different decoding algorithms. Started with the bit-flipping algorithm, the procedure of message passing is introduced. Then the mechanism of soft-decision decoding based on the sum-product algorithm is summarized. Simplified from sum-product equations, the min-sum algorithm is introduced at last.

Chapter 3 firstly compared different implementations of LDPC decoder base on the parallelism. Then the fully parallel architecture is introduced in detail, as well as the micro architecture of CNU and VNU. The implemented fully parallel decoder achieved 1152-bit code length, works at 70MHz and has a throughput of 6 Gbps at 3.0 dB SNR. Based on the fully parallel architecture, Chapter 4 introduced the partially parallel architecture for longer code length by dividing data into groups and process in multiple clock cycles. This implementation accomplished 2304-bit code length within similar resource utilization as 1152-bit fully parallel decoder by dividing VNU and CNU memory into two groups. And it works at 71MHz, having a throughput of 4 Gbps at 3.0 dB SNR.

In Chapter 5, the multi-frame processing architecture based on fully parallel decoder is proposed. By pipelining the VNU and CNU arrays and out-of-order processing of frames, the 3-stage design is able to reach 145MHz frequency and 12.6 Gbps throughput at 3.0 dB SNR.

Additionally, in Chapter 6 the mixed precision scheme is proposed and implemented by precision control and alignment units before and after CNU stages. This scheme improved

lower SNR performance of the decoder and also maintained a low error floor at high SNR.

7.2 Future Work

Although the parallel decoder can have high throughput, the area utilization ratio is low, because more than 50% of the area is used for wiring. Therefore, a better wiring and floor plan algorithm should be developed for better area and resource utilization. On the other aspect, the power consumption also increases very fast when more components are added and higher frequency is reached by 3-stage design. By further optimizing the architecture, lower power consumption is expected to be achieved.

The partially parallel decoder in Chapter 4 also has a similar pipeline structure as the multi-frame processing designs. Therefore, by extending the multi-frame processing to partially parallel decoder, it is expected to reach a better throughput. However, further trade-off between throughput and area should be considered as well.

Since the mixed precision scheme in Chapter 6 effectively enhanced the performance of the decoder while not degrading error floor, but for now the selection varied precision nodes are selected randomly and only at ratio 1:1. Further systematic investigation and also more combinations of mixing is going to performed in order to find the optimal mixing scheme.

Also this work is prepared to be published to ISCAS 2017 after further research.

REFERENCES

[1] J. G. Proakis, *Digital Communications, 5th edition*. McGraw-Hill Education, November 2007.

[2] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[3] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[4] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Transactions on communications*, vol. 44, no. 10, pp. 1261–1271, 1996.

[5] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[6] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[7] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[8] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[9] M. Eroz, F.-W. Sun, and L.-N. Lee, "Dvb-s2 low density parity check codes with near shannon limit performance," *International Journal of Satellite Communications and Networking*, vol. 22, no. 3, pp. 269–279, 2004.

[10] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area networks. part 16: Air interface for fixed broadband wireless access systems," *IEEE Std*, vol. 802, pp. 16–2004, 2004.

[11] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Gen03-6: Investigation of error floors of structured low-density parity-check codes by hardware emulation," in *IEEE Globecom 2006*, pp. 1–6, IEEE, 2006.

[12] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE transactions on information theory*, vol. 47, no. 2, pp. 619–637, 2001.

[13] M. P. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on communications*, vol. 47, no. 5, pp. 673–680, 1999.

[14] A. Shokrollahi, "Ldpc codes: An introduction," *Digital Fountain, Inc., Tech. Rep*, p. 2, 2003.

[15] S. J. Johnson, "Introducing low-density parity-check codes," *University of Newcastle, Australia*, 2006.

[16] M. Karkooti and J. R. Cavallaro, "Semi-parallel reconfigurable architectures for real-time ldpc decoding," in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, vol. 1, pp. 579–585, IEEE, 2004.

[17] J. Chen and M. P. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on communications*, vol. 50, no. 3, pp. 406–414, 2002.

[18] V. Savin, "Self-corrected min-sum decoding of ldpc codes," in *2008 IEEE International Symposium on Information Theory*, pp. 146–150, IEEE, 2008.

[19] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of ldpc codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[20] K. K. Gunnam, *Area and Energy Efficient VLSI Architectures for Low-Density Parity-Check Decoders Using an On-The-Fly Computation*. PhD thesis, Texas A&M University, 2006.

[21] J. Heo, "Analysis of scaling soft information on low density parity check code," *Electronics Letters*, vol. 39, no. 2, pp. 219–221, 2003.

[22] A. J. Blanksby and C. J. Howland, "A 690-mw 1-gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, 2002.

[23] A. Prabhakar and K. Narayanan, "A memory efficient serial ldpc decoder architecture," in *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 5, pp. v–41, IEEE, 2005.

[24] A. Shevchenko, R. Maslennikov, and A. Maltsev, "Comparative analysis of different hardware decoder architectures for ieee 802.11 ad ldpc code," in *MELECON 2014-2014 17th IEEE Mediterranean Electrotechnical Conference*, pp. 415–420, IEEE, 2014.

[25] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for quasi-cyclic ldpc codes," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 15, no. 4, pp. 483–488, 2007.

[26] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A 3.3-gbps bit-serial block-interlaced min-sum ldpc decoder in 0.13-μm cmos," in *2007 IEEE Custom Integrated Circuits Conference*, pp. 459–462, IEEE, 2007.

[27] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of ldpc codes," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, pp. 107–112, IEEE, 2004.

[28] J. Xu, T. Che, E. Rohani, and G. Choi, "Asynchronous design for precision-scaleable energy-efficient ldpc decoder," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 136–140, IEEE, 2014.