VLSI Architectures of Turbo Decoder

Thesis submitted in the partial fulfillment of requirement for the award of degree of

Master of Technology

in

VLSI Design & CAD

Submitted By:

Navnish Kumar

Roll No: 601061015

Under the guidance of:

Dr. Sanjay Sharma

Associate Professor

Department of Electronics and Communication Engineering

Thapar University

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

June 2012

DECLARATION

I, **Navnish Kumar**, hereby certify that the work which is being presented in this thesis entitled **"VLSI Architectures of Turbo Decoder"** by me in partial fulfilment of the requirements for the award of degree of Master of Technology in VLSI Design from Thapar University (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Sanjay Sharma**, Associate Professor, ECED, Thapar University, Patiala.

The matter presented in this thesis has not been submitted in any other University / Institute for the award of any other degree.

Date: 29-06-2012

Navnish Kumar
Roll No. 601061015

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date: 29.6.2012.

Dr. Sanjay Sharma
Associate Professor
ECED

Countersigned by:

Dr. Rajesh Khanna
Professor and Head ECED
Thapar University, Patiala

Dr. S. K. Mohapatra
Dean of Academic Affairs
Thapar University, Patiala

i

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an un-trodden path towards and unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to **Dr. Sanjay Sharma, Associate Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Head of the Department, Dr. Rajesh Khanna** as well as **PG Coordinator, Dr. Kulbir Singh, Assistant Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

ABSTRACT

Turbo codes are one of the most successful error correction codes. Due to the use of Pseudo random interleaves, encoded data appears to be random in channel, yet turbo decoders are intelligent enough to decode the original data. And also due to its inherited recursive nature to use soft inputs and outputs, it provides better BER compared to convolution codes. But there is a price to pay for better communication performance is in terms of Storage requirement, decoding delay and computation complexity.

The objective of this thesis work is to analyse different Soft Input Soft Output (SISO) a posterior probability (APP) turbo decoders at architecture level which gives different performance metrics in terms hardware requirement and throughput. I used a graphical approach known as Tile-graph to analyse different implementations of SISO APP Turbo decoder, which provides an estimate of decoding delay and hardware requirement. Two different optimized approaches, Sliding Window and Parallel Window for SISO MAP algorithm are discussed and further the idea of composition of Sliding Window and Parallel window is also taken in account using tile graph approach known as Hybrid Tiling. Further to increase the throughput of turbo decoders, pipelining and retiming has been also applied at block level.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In a digital transmission system, Channel coding is a process of signal transformation to improve communication performance by encoding signal in such a way that it can better withstand the effect of channel noise and fading. Channel coding is used to accomplish trade-offs between error performance and bandwidth or between power transmitted and error performance. Channel coding can be divided into two categories:

- Waveform Coding
- Structured Sequence

Here, in this report we will discuss only structured sequence.

Structured Sequence: These structure deals with transforming the data sequences into better sequences by introducing redundancy bits. These redundancy bits are used for error detection and correction. Encoding procedures provide the coded signals with better distance properties compared to original data.

There are two ways by which we can use these redundancies to control errors:

Error detection and retransmission: These techniques use redundancies to detect an error; If an error is detected, receive doesn't try to correct it rather send a request to transmitter to retransmit the data. In this technique, a two way link is required between transmitter and receiver.

Forward Error Control: In these techniques, only one way link is required for data transmission, since redundancies are used for both error detection and correction.

Structured Sequences can further be classified into three subcategories, names as:

- Block Code

- Convolution Code
- Turbo Code

Block Codes: In case of block codes, source data is segmented into blocks of k data bits per block, where each block can represent any one of 2^k different messages. Encoder transforms each k bit data into the blocks of n-bit called code bits, where k<n. The new (n-k) bits which are added by encoder carry no new information are called redundant bits. This code is referred as an (n, k) code. Ratio of redundant bits to the data bits, denoted (n-k)/k, is called redundancy of the code. And the ratio of data bits to total number of bits, k/n, in a single block is called the code rate.

Following are the some well-known block codes:

- Hamming Codes
- Extended Golay Codes
- BCH Codes

Convolution Codes: In case of block codes, coding is described by two integer, k and n. where k is the number of input data bits and n represents total number of bits on output of encoder.

A convolution code is described by three integers, n, k, and K, where k/n has the same code rate significance that was for block codes. The integer K is a parameter known as Constraint Length, it represents number of stages in Convolution encoder. While in case of block codes, there is a unique n bit code for every k bit code, in case of convolution codes, n bit output code is a function of both k bit current input as well as K-1 previous k input patterns.

Convolution encoder can use one out of two approaches from the following:

- Systematic Convolution Codes
- Nonsystematic Convolution Codes

Systematic convolution codes are the one in which the input k-bit pattern appears as part of output n bit code pattern.

Following are the wall known Convolution decoder:

- Hard Decision Viterbi Decoding
- Soft Decision Viterbi Decoding
- Sequential Decoding
- Feedback Decoding

Turbo Codes: Recently, a new class of error correcting codes called Turbo codes was introduced by Berrou, Glavieux, and Thitimajshima, in 1993 [1]. Concatenated coding scheme were initially proposed as a method to achieve large coding gain by combining two or more relatively simple component codes. Resulting codes has the error correction capability of much longer codes, yet possess enough structure to be decoded on receiver side.

Turbo codes are constructed by using two or more component codes on different interleaved version of same information. For convolution codes, final step at decoder yield hard decision decoded bits but for concatenated codes like turbo decoder, decoding algorithms must have to pass soft decoded information across decoders.

Following are the two main algorithms for Turbo decoder:

- Soft Output Viterbi Algorithm
- SISO-MAP Algorithm

Turbo codes will be explained in much detail in chapter 2.

1.2 Motivation

In 1948 Claude E. Shannon established the fundamental limits in transmission speed in the systems of digital communication and led to the search for coding techniques for approaching the limits of channel capacity. Turbo Codes are one of the most important developments in recent years within channel coding, which are reaching the limits of this capacity.

The near Shannon limit error correction capability has lead turbo codes to become the coding technique of choice in many communication systems and storage systems since their introduction. Some of these applications include, among others, the Third Generation Partnership Project (3GPP), Consultative Committee for Space Applications (CCSDS) telemetry channel coding and Wideband CDMA, which require throughputs in the range of 2 Mbps to several 100 Mbps. As these applications continue to evolve, the trend is shifting toward more and more strict requirements on power consumption and processing speeds as part of standard design practice, which has prompted for more efficient turbo decoder implementations.

1.3 Objective

For Turbo codes, speed is the most critical issue; but for handheld devices, area and power are also the issue of importance.

In this study, we will start with the basic architecture of Turbo decoder based on SISO-MAP Algorithm [2]. We will use a graphical approach known as Tile-graph which is used for the performance analysis of architectures. Further, Optimized architecture with increased computational complexity but performing better in terms of decoding delay and storage requirement will be discussed.

Sliding Window (SW) SISO architecture perform better than standard SISO-MAP architecture, but still for long sequences, due to long decoding delay, we have to look for better architectures.

Further, Parallel Window (PW) SISO architecture are discussed, which perform better than SW SISO architecture, but resource requirement is too high. At last, a highly optimized technique has been discussed known as Hybrid Tiling, which uses both SW and PW methodology in same architecture. And in the end, performance metrics are given for these architectures.

1.4 Literature Survey

C.E.Shennon [3] explored a direct relation between channel capacity, bandwidth and power transmitted. He proved that, to use channel capacity more and more efficiently, either we have to increase more bandwidth or have to transmit more power. But the main problem is that we want to minimize the usage of both bandwidth and transmitted power.

G.D.Forney [4] proposed concatenated codes in 1966. In his paper, he proved that in spite of using a single long coding, if we divide the encoding in two parts, decoder's complexity can be reduced to an effective value at receiver end without any loss of performance.

L.R.Bahl et. al., [2] proposed the MAP or Forward-Backward Algorithm to decode the convolutional as well as parallel concatenated codes in 1974. In this algorithm, technique used to decode the data works on every bit separately rather decoding technique of Viterbi algorithm [5] which decodes data as a sequence.

Claude Berrou et. al., [1] proposed a new king of codes known as turbo codes in 1993. These codes provided Bit error rate (BER) performance closed to the Shannon's limit. These codes used the parallel concatenation of two recursive systematic encoders.

Patrick Robertson et. al., [6] proposed a decoding technique known as Log-MAP algorithm in 1995, which provides decoding efficiency in vicinity of original MAP algorithm, while at same time reduces the implementation complexity by converting the usage of multiplications and divisions in additions and subtractions which are much easy to implement.

Andrew J. Viterbi [7] provided an easy intuitive way to understand MAP algorithm. He used the duality of Viterbi algorithm in forward and backward direction to explain MAP algorithm. He also proposed a time sharing technique which can optimize both memory requirement and latency of turbo decoder.

S. Banedetto et. al., [8] proposes the technique of sliding window to be used in MAP algorithm as it was previously used in Viterbi algorithm. In sliding window decoding technique, decoding can be started without receiving the termination trellis matrices.

H. Dawid et. al., [9] provided an architecture in this paper to effectively use sliding window implementation of MAP algorithm. He also proposed a method to compute the required number of initialization matrices for backward matrices computation.

Y. Wu et. al., [10] proposed the technique to normalize the different matrices of MAP algorithm. In this paper, he studied the numerical characteristics of MAP algorithm and found that algorithm basically functions upon the difference of different path matrices rather than their actual magnitude. He provided the normalization technique that is directly implementable in 2's complement arithmetic.

M.M. Mansour et. al., [11] in 2002, proposed a novel approach to design and analyze VLSI architectures of turbo decoders. His approach is based on the tile graph, and he divided the designing of turbo decoders as a three step process. He proposed the architectures for sliding window and parallel window for both high speed and low power.

A. Worm et. al. [12], in 2001, explored the novel parallel architectures of turbo decoders in their paper. They provided VLSI architectures of turbo decoders with highly optimized memory and power consumption, where they can be used over a large window of throughput from 300 Mbps to 45 Gbps and above.

Y. Tong et. al., [13] provided a thorough comparison between MAP, Log Max and Max Log MAP algorithms, and also proved a 12 bit fixed point architecture for Log MAP algorithm, and checked the performance against floating point arithmetic based MAP algorithm.

M. M. Mansour et. al., [14] provided a through comparison between different VLSI architecture of turbo decoders and also used Tile Graph based approach to estimate their memory and latency requirement. They also provided a new approach, Hybrid Tiling, which effectively reduces hardware and memory requirement.

Z. Wang et. al., [15] proposed a new retimed architecture for Add-Compare-Select unit, By that time, ACS unit was the speed bottleneck for the turbo decoders, they proposed the ACS unit with critical path one third less to the previous ones.

Yang Sun [16] proposed in this paper a novel architecture for QPP interleaver and applied this to different Radix-2, and Radix-4 Turbo decoders.

1.5 Organization

This report is organized into six chapters.

Chapter 1 introduces the basics of channel codes, followed by the motivation behind the problem, afterward objectives of the report, then literature survey and at last, report organization is given.

In Chapter 2, we start with the introduction of Turbo encoder and decoder, followed by the explanation of SISO MAP algorithm.

Chapter 3 starts with the basic standard architecture of SISO MAP Algorithm using tile-graph approach followed by the optimized Sliding window (SW) and Parallel Window (PW) architectures. Different types of SW and PW are also discussed. Finally, Hybrid Tiling, which uses composition of both SW and PW techniques in a single design has been discussed.

Chapter 4 describes the implementation of VLSI architectures of different turbo decoders, and their block level improvements.

Chapter 5 is composed of simulation and synthesis of different VLSI architectures of turbo decoders and analysis of their results.

Chapter 6 finally provides the conclusion of the thesis and future perspective about what can be dame in future of the problem concerned.

CHAPTER 2

ITERATIVE TURBO DECODING

In this chapter, we will start with the introduction of turbo encoder and decoder, and thereafter will discuss SISO MAP algorithm in detail.

2.1 Turbo encoder

In the general case, the output of turbo encoder consists of two parts: original information bits and a set of parity bits generated by convolution encoders which take their input either directly or via an interleaver. We can use both Non-recursive and recursive systematic encoders but we prefer Recursive Systematic encoders, generally.

A two component Turbo encoder is shown below in figure 2.1, in which two parity bits are generated, one which is generated with non-interleaved data sequence and other with interleaved data.

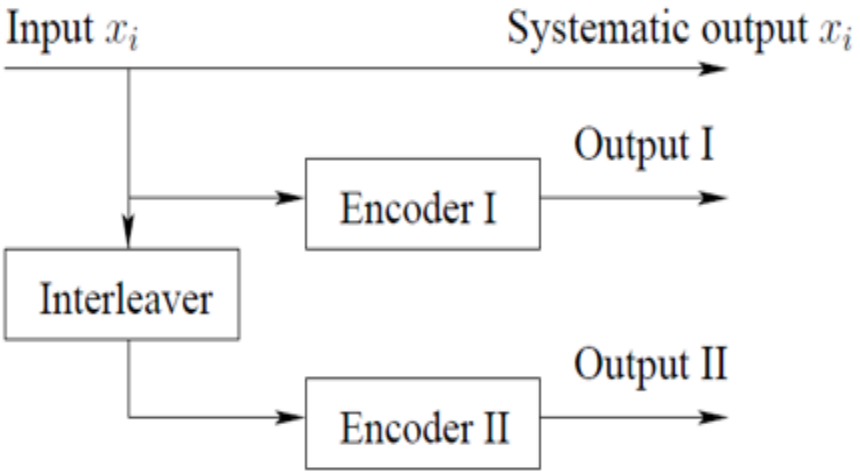

Figure 2.1: A two component turbo encoder

2.2 Recursive Systematic Convolutional (RSC) Encoder

The recursive systematic convolutional (RSC) encoder is obtained from the non-recursive nonsystematic (conventional) convolutional encoder by feeding back one of its encoded outputs to its input. Figure 2.2 is representing a Recursive systematic encoder.

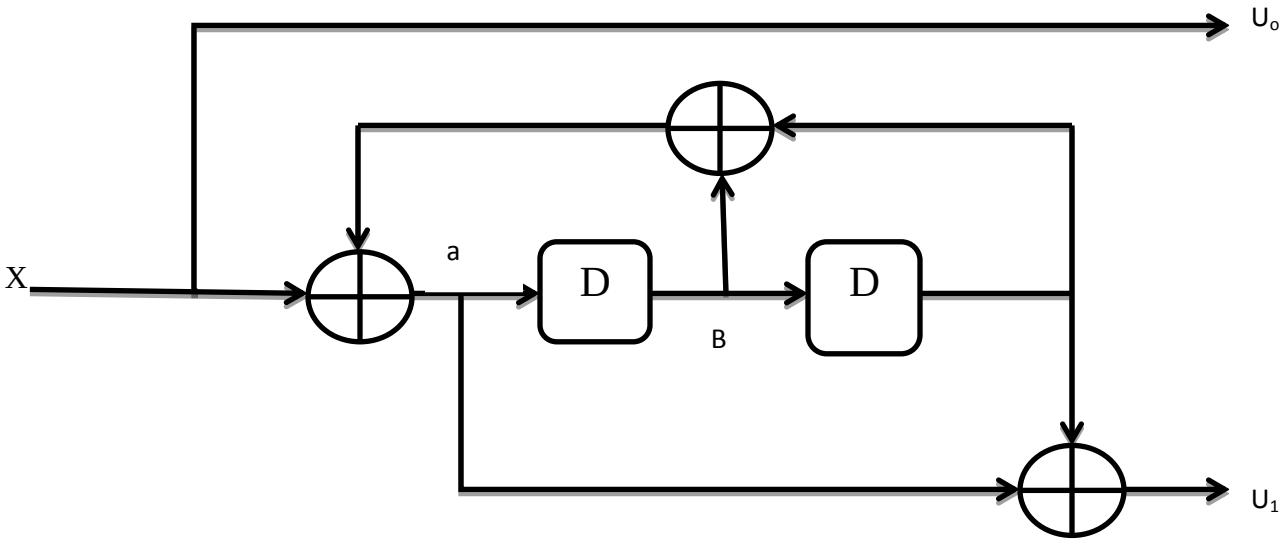

Figure 2.2: A Recursuive Systematic encoder with r=1/2 and K=3

The conventional convolutional encoder is represented by the generator sequences **g1** = [111] and **g2** = [101] and can be equivalently represented in a more compact form as **G**= [**g1**, **g2**]. The RSC encoder of this conventional convolutional encoder is represented as **G**= [1, **g2** / **g1**] where the first output (represented by **g1**) is fed back to the input. In the above representation, 1 denotes the systematic output, **g2** denotes the feed forward output, and **g1** is the feedback to the input of the RSC encoder. Figure 2.2 shows the resulting RSC encoder.

2.3 Interleaver

For turbo codes, an interleaver is used between the two component encoders. The interleaver is used to provide randomness to the input sequences. The interleaver [16] serves two purposes. First, the interleaver provides buffering between the outer code and the inner code. Second, more importantly, interleaving spreads out burst errors so that error bursts are randomized and become more correctable.

Following are the some types of interleavers used in turbo codes:

- Block Interleaver
- Pseudo-random interleaver
- Circular-shifting interleaver
- Odd-even interleaver
- QPP Interleaver

2.4 Turbo Decoder

For the typical two component Turbo encoder of figure 2.1, the standard iterative decoder is shown in a simplified format in figure 2.3. The priori information is usually reset to be equi-probable before starting to decode a frame. The decoder blocks usually use a Bahl-Cocke-Jelinek-Raviv (BCJR) [2] algorithm or Soft-Output Viterbi algorithm (SOVA) [8, 9]. Here in this study, we will use SISO MAP technique based on BCJR algorithm.

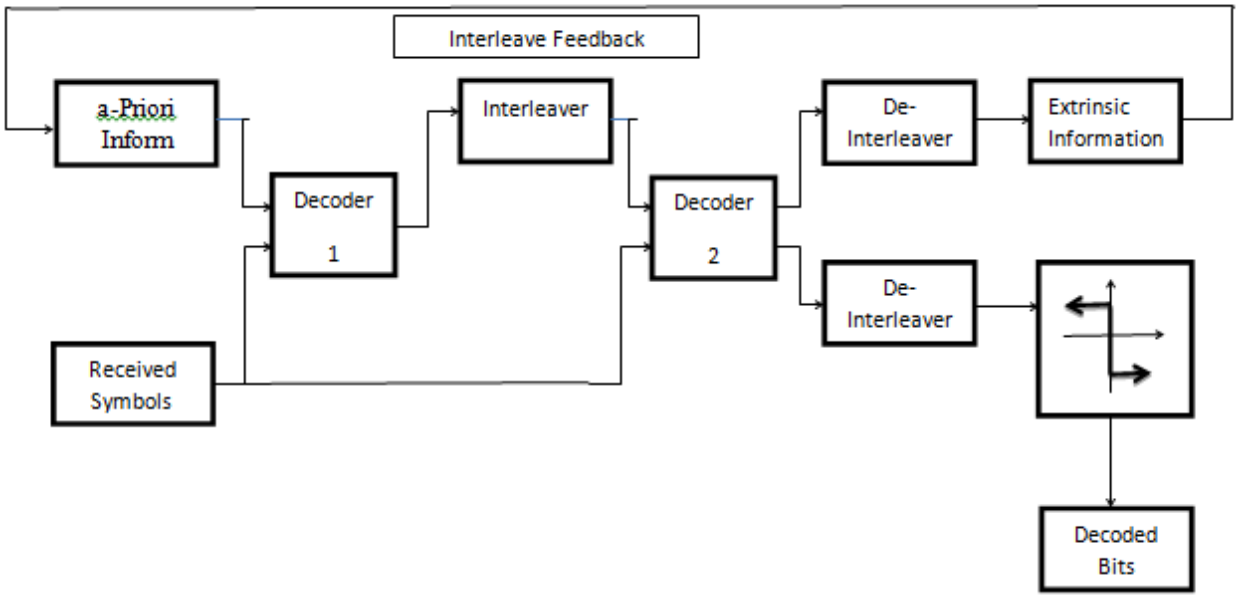

Figure 2.3: Two Component Turbo Decoder

If the Turbo code consists of more components, it is just a matter of inserting the relevant interleaver, decoder and a de-interleaver. From previous iteration to successive iteration of Turbo decoder, extrinsic information (a posteriori probability) is passed to be used as a priori probability. After enough number of iterations of decoder, a hard decision block associated with last decoder is used to provide decoded bits.

Turbo decoders have good performance but price paid is in terms of decoding delay and resources requirement; so we will discuss the different implementation of decoding algorithm, i.e. SISO MAP algorithm, but firstly we take an overview of algorithm.

2.5 Soft Input Soft Output MAP Algorithm

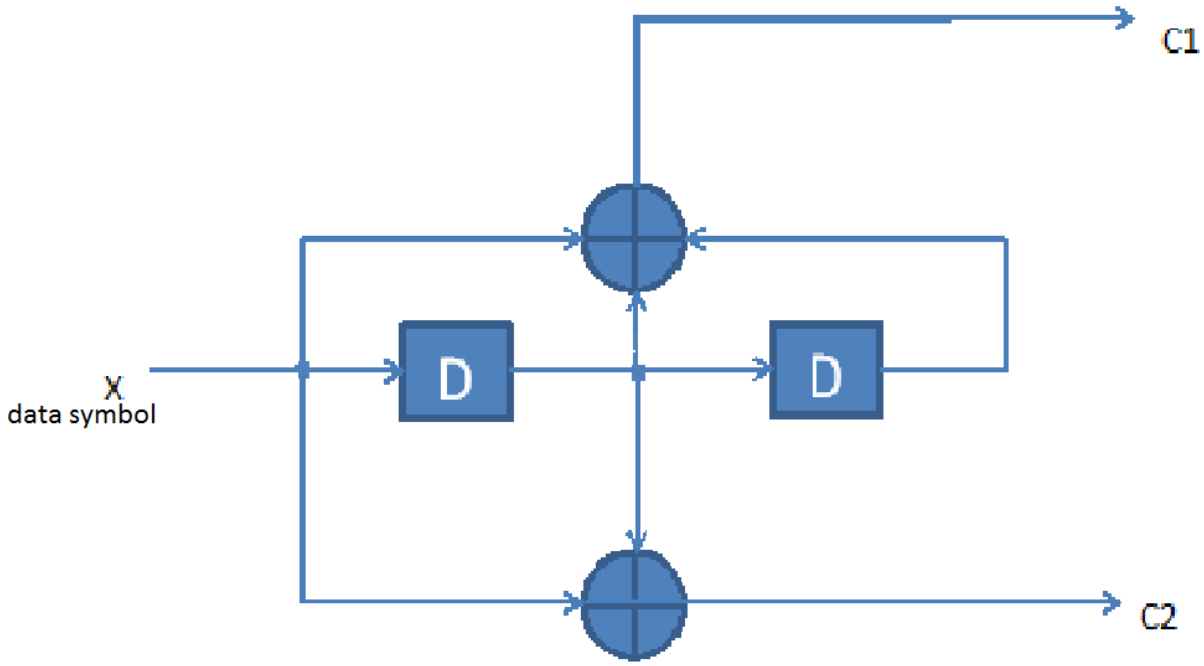

Figure 2.4: A [2, 1, 3] Convolution Code

Consider an convolution encoder with code (N_0,K_0,V)=(2,1,3),as shown in figure 2.4, where it codes K_0 information bits into N_0 encoded bits and V is called as the Constraint length of encoder. The code rate of encoder is defined as $R=K_0/N_0$. Numbers of storage elements in the encoder are one less than the constraint length. So at any time, encoder can be in any one state out of 2^{v-1} states. And further for every pair of input variable and current state (u,s_i), there is a unique pair of output variable and next state(c,s_j).

As input data proceeds from first element K_0 to last element K_{l-1}, encoder passes from stage S_0 to S_{l-1} and generate output code pairs C_0 to C_{l-1}.

State transitions can be graphically represented, as shown in figure 2.5, with the help of Trellis diagram as shown below for the convolution encoder of previous diagram.

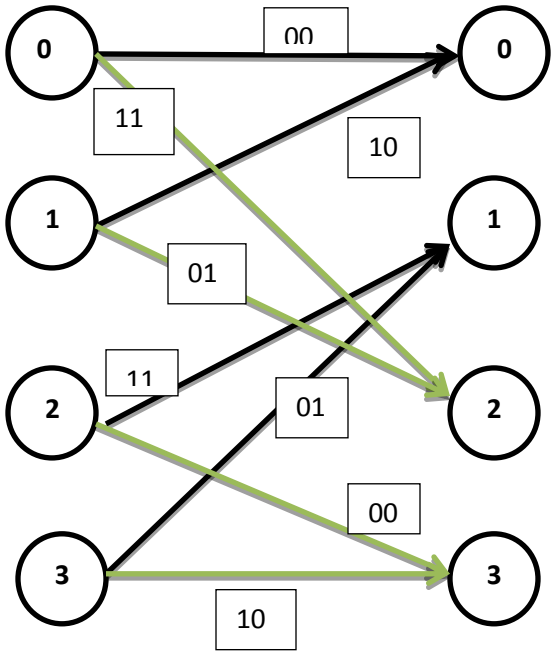

Figure 2.5: Trellis Diagram for (2, 1, 3) convolution encoder

Assume we transmit encoded data and receive noisy data patterns denoted by: Y ≜ (y_1, y_2……y_n). Now, the problem is to find out original data U out of noisy pattern Y.

According to MAP algorithm, we determine data sequence U by calculating each and every symbol U_k separately from whole received sequence Y. At any instant, survived value is the one which got maximum a-posterior probability P (U_k|y) and due to this reason, algorithm got the mane Maximum A-posteriori Probability Algorithm. As we use soft information from one decoder to another, it got the name Soft Input Soft Output.

MAP Algorithm decodes the data sequence in four steps as follows:

- Branch metrics computation
- Forward State Metrics Computation
- Backward state metrics computation
- Reliability Update

2.5.1 Branch Metrics Computation (S_k, S_{k+1})

For k= 1, 2….L, decoder computes the reliability for every code for 2^{n0} elements. It also accepts a-priori reliabilities from other component decoders working on same data with different

encoding. Basically, branch metrics shows the probability of an edge. For any given code element K, it shows the probability for switching from one state S_x to another S_y state.

For the hard decisions, branch metric computation uses Hamming Distance; and for soft decisions, it relies on Euclidean distance [17].

Hamming distance between two code words U and V is defined as the number of elements in which they differ. Soft decision decoders can't use Hamming distance due to its limited resolution, so we go for Euclidean distance for SISO decoders.

Euclidean distance for a noiseless sequence X={x_0,x_1} and a noisy sequence Y= {y_1, y_2} is defined as:

$$D=\sqrt{(y0 - X0)^2 + (y1 - x1)^2} \quad (2.1)$$

2.5.2 Forward State Metrics Computation (α_k[s])

From the above computed forward metrics, the algorithm computes a forward state metrics for each state (s_0…….s_l). Basically forward state metrics shows the probability of state s_j at time k given the received code from 0 to k time. As trellis starts with initial state 0, at time t0, we can define forward state metrics as follows:

$$\alpha_0[S_j] = \begin{cases} 0 & Sj = S0 \\ -\infty & Sj \neq S0 \end{cases}, \text{ j=0, 1,2……}2^{v-1} \quad (2.2)$$

Rest of the forward metrics can be calculated using the following equation

$$\alpha_k[S_j] = \max *_{e:E(e)=Sj}\{\alpha k - 1[S(e)] + \lambda k[uc(e)]\}, \text{ , j=0,1,2……}2^{v-1}, \text{ k=1,l-1} \quad - \quad (2.3)$$

Where function $\max^*$ is defined as [18]:

$$\text{Max}^*\{x,y\} = \ln(e^x + e^y) = \max(x,y) + \delta(x,y) \quad (2.4)$$

Where $\delta(x,y)$ *is a correction factor*[6].

2.5.3 Backward State metrics computation (β_k[s])

One more set of computational reliabilities are calculated from received code, which shows the probability of state S_j at any time K, given all future observation y_{k+1}…….Y_l. As the trellis is ended in 0 state, we can take the backward reliability at time l as follows:

$$\beta_l[S_j] = \begin{cases} 0 & Sj = S0 \\ -\infty & Sj \neq S0 \end{cases}, \text{ j=0,1,2……}2^{v-1} \quad (2.5)$$

Rest of the backward metrics can be calculated using the following equation

$$\beta_k[S_j] = \max *_{e:E(e)=Sj}\{\beta k - 1[S(e)] + \lambda k[uc(e)]\}, \text{ , j=0,1,2……}2^{v-1}, \text{ k=L……2} \quad (2.6)$$

2.5.4 Reliability Updates ($\Lambda_k[u]$, $\Lambda_k[c]$)

Using all three metrics i.e. Branch, forward and backward metrics, we compute both output data and code symbols in soft form i.e. there is a probability associated with every computed symbol. These computations known as the reliabilities are the a-posterior probability of the information bits; this information is used as the a-priori probability in the upcoming iteration. Reliability of data and code symbols are given by $\Lambda_k[u]$ and $\Lambda_k[c]$, respectively. The reliability equations are given as below:

$$\Lambda_k[u] = \max^*_{c:u(e)=u} \{ \alpha k - 1[S(e)] + \lambda k[uc(e)] + \beta k[E(e)]\} \quad (2.7)$$

$$\Lambda_k[c] = \max^*_{c:c(e)=u} \{ \alpha k - 1[S(e)] + \lambda k[uc(e)] + \beta k[E(e)]\} \quad (2.8)$$

For k=1,……l

These reliability values are used in further iterations, but before that, these have to be saved in de/interleaved memory in binary representation.

Equations (2.3), (2.6), and (2.7) & (2.8) show the key equation for forward, backward metrics and reliability computation. Equation (2.4) gives the equation for $\max^*$ function.

2.5.5 QPP Interleaver

As we know that interleaving and de-interleaving of data is an important operation in turbo codes. But when we design parallel turbo decoder, this operation becomes two complicated as more than one parallel turbo decodes try to access the same location. These new QPP interleavers [16] as they are based on algebraic computation via permutation polynomials, they generate contention free interleaver i.e. every interleaver length can be used in parallel operations.

Mathematical Description of QPP Interleaver

Operation of QPP interleaver can be explained as a simple formula. If we assume that we have a data sequence of length N, then x-th interleaved position can be given as:

$$f(x) = (f_2.x^2 + f_1.x) \bmod N \quad (2.9)$$

where f_1 and f_2 are two integer constants depending upon the value of sequence length N. Values of these constants are defined for all possible sequence lengths.

As per LTE standards, all sequence length or block sizes are divisible by 4 and 8 and at the same time divisible by 16, 32 and 64 as block sizes becomes equal or greater to the values 512, 1024 and 2048 respectively.

QPP interleaver has following algebraic properties:

1. f_1 and f_2 are always odd and even numbers respectively.
2. f(x) will have the same parity as x.
3. Remainders of f(x)/4, f(x+1)/4, f(x+2)/4 and f(x+3)/4 will always be unique.

Implementation of all these blocks will be later discussed in chapter 4.

CHAPTER 3

SISO MAP DECODER ARCHITECTURE

In this chapter, we will start with fundamental architecture of MAP decoder and after that, will see optimized Sliding Window (SW) [7] and Parallel Window (PW) [9] architectures. And we will also study Tile Graph methodology [11] that is used to perform data flow analysis on Data Flow Graphs (DFG), which is used to optimize SW-DFG for minimal decoding delay and requires least metric storage area.

3.1 Standard SISO MAP algorithm

We will discuss the SISO architectures in term of DFGs because DFGs provide flexibility in estimating resource-time tradeoffs without affecting design style. A DFG for SISO MAP decoder [19] is shown in the following figure 3.1, where we assume that data sequence contains L symbols, Data sequence proceeds from left to right side and time index runs from top to bottom.

Input branch matrix is supplied from the top side and forward metrics are calculated from node 1 to node L-1 of data sequence and stored in the buffers. At time L, calculation of backward metrics get started and output reliabilities got computed in reverse order i.e. from node L to 1.From the DFG shown below, we can calculate the decoding delay (Latency) by calculating the height of it and storage requirement can be calculated by the number of buffer used in DFG.

In figure 3.1, the dark shaded area corresponds to the buffers required for branch and forward metrics, while the light shaded area corresponds to the storage requirement for branch metrics and output reliabilities. These buffers basically shows the hardware utilization as a function of time and due to this, it is better to consider storage lifetime rather than considering total number of buffers.

Limitation: It requires whole data sequence of L symbols to be received before decoding can start.

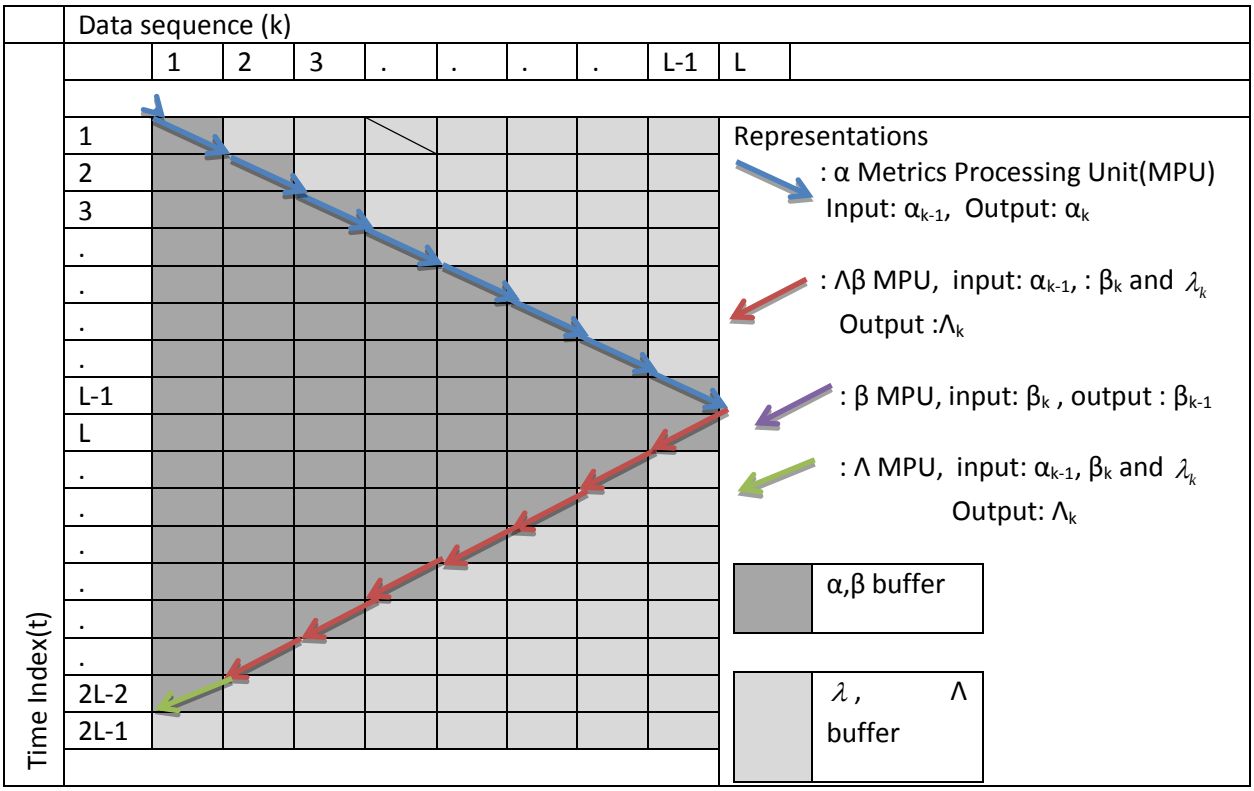

Figure 3.1: A simplified DFG representation of Standard SISO MAP algorithm showing the resource utilization of α and β recursion flow with time

As we can see from the DFG of figure 3.1, delay is directly proportional to the length of data sequence, latency become the bottleneck for the uses of Standard SISO-MAP based architecture for the long sequences. So we have to look for more optimized architectures to overcome this limitation.

3.2 Modes of Communication

Depending upon Decoder setup, communication can be differentiated in three modes:

- Terminated Frame mode
- Truncated Frame mode
- Continuous mode

3.2.1 Terminated Frame Mode

In this mode, initial and final states of the decoder are already known to decoder, so we can assure the α and β probability at t=0 and t=L, respectively. Standard MAP algorithm works on the principle of terminated frame mode.

Terminated frame mode has a series limitation that it requires the whole data sequence before the decoding can even get started. Due to this reason, this mode is only used for small data sequences.

3.2.2 Truncated Frame mode: In this mode, we assume only α is known at time t=0, and final state of encoder is not known to the decoder. This mode overcomes the buffering the whole data sequence limitation of Terminated frame mode, and due to which decoding delay or latency get reduced to a large instant.

Basically, Idea behind this mode is to run the β recursion for a limited portion of data sequence in spite of whole sequence. It has been shown experimentally that after a particular recursion depth, metrics converges to their reliability values, which depends on both code parameter (n,k,v) as well as channel SNR ratio.

Sliding window (SW) SISO algorithm is based on this approach.

3.2.3 Continuous mode: In this mode, we assume that data sequence is of infinite length and initial and final stage of encoder is not known to the decoder. So, in this approach, decoder splits the whole data sequence into parts and assume some initial approximations for both α and β. And similar to Truncated mode, after a certain recursion depth, both forward and backward metrics become valid.

Parallel window (PW) SISO algorithm is based on this approach.

Following are the two optimization at the behavioral level in MAP algorithm:

- Sliding Window (SW) SISO MAP Algorithm
- Parallel Window (PW) SISO MAP Algorithm

3.3 Sliding Window SISO MAP Algorithm

This technique uses the Truncated Frame mode of communication. There are three versions of SW-SISO MAP algorithm known as SW-SISO-1 [14], SW-SISO-2 and SW-SISO-3 [20,21], which differs from each other in term of β- recursion initialization and number of valid metrics produces per β- recursion flow.

SW-1 and SW-2 decodes only one code for every β- recursion.

3.3.1 SW-1 SISO Algorithm

In SW-1, first β- recursion is scheduled only after M+1 step after the start of α-recursion flow. And after first β- recursion, a new recursion is scheduled after every step. β- recursion is initialized after M+1 steps with the initial metrics same as α metrics, i.e.:

$\beta_k[S_i] = \alpha_k[S_i]$ for I = 0,….,2^{m+1}

Decoding Delay of SW-1 SISO MAP algorithm is directly proportional to (L+2M)+1.

3.3.2 SW-11 SISO Algorithm

In SW-2, β-recursion is initialized before M+1 steps after the start of α computation, so there is no storage required for any forward or backward metrics. B-recursion is initialized as:

$$\beta_k[S_i] = \text{constant} \quad \text{for } i = 0,\ldots,2^{m+1}$$

we can see that decoding delay of SW-2 algorithm is proportional to (L+M)+1.

3.3.3 SW-111 SISO Algorithm

In SW-3, again as in SW-2, β-recursion is initialized before M+1steps after the start of α computation, but this time, each β-recursion flow produce more than one decode.

Delay of SW-111 algorithm is proportional to (L-1+δ) where δ refers to $\frac{M+|d-G|+|d-G-M|}{2}$.

Delay of SW-111 will be discussed in more detail in subsequent topics.

Advantages of SW-SISO Algorithm

- Decoding Delay improves in multiple value of M.
- There is a significant saving in storage in SW-2 and SW-3.

Limitation: Still as the decoding Delay is proportional to L, Speed limits after applying SW techniques lying near to 200 mbps, So beyond this speed requirement we have to look for better techniques.

Here comes the third mode of communication, continuous mode. In this mode, Decoder divides the data sequence into subparts, and decoded these subparts parallel. But now in this case, we don't know both β_l and α_0. So, approximations have to be made on both ends. We will discuss this technique after analyzing some optimization techniques for SW-SISO algorithm.

3.4 Optimization of Sliding Window Architecture

We know that by using SW-2 and SW-3 SISO MAP algorithm, a significant improvement in decoding delay and storage requirement can be achieved. Now, we will make some analysis to get the best decoding delay and storage requirement. The analysis basically deals with parameter (d,M,G) of sliding window. We will analyze the DFGs using a graphical approach called Tile-graph approach.

3.4.1 Tile Graph Methodology

Tile Graph methodology [11] is a methodology to analyze the SW-DFG, which is made up of single α recursion flow and multiple β recursion flow or inversely, single β recursion flow and multiple α recursion flows for the calculation of minimal delay and storage requirements.

Up to now, we have partitioned the L symbol data sequence into smaller blocks, where every β recursion decode G symbols for G<L. The β recursion flow in the DFG are independent to each other. For every β recursion phase, there is a warm-up phase of length M and a decode phase of Length G.

If we have given a single recursion pattern, we can design a whole DFG by tiling this single recursion pattern. A DFG constructed by composition of tiles is called as Tile Graph. Recursion pattern is defined with points (A,B,C,D), where A and B with respective coordinates ($K_{a,}$ t_a) and ($K_{b,}$ t_b) designates the starting and ending point of α-recursion flow. while C and D represents same for the β recursion. Now we can conclude that to optimize the performance of a SW-MAP decoder, we just have to optimize a single recursion pattern.

3.4.2 Sliding Window DFG Design

Design of a Tile graph is a process of three steps as follows:

- Recursion pattern Design
- Determination of Numbers of recursion pattern
- Tiling the recursion pattern

3.4.2.1 Recursion pattern Design

A recursion pattern of SW-MAP algorithm is configured based on three parameters (d,M,G) where d is the difference between start of α recursion and end of β recursion, M is the number of metrics required to initialize β-recursion flow, G is the number of decoded symbols for every

β recursion. And all three are independent to each other. Coordinates of the (a,b,c,d) points can be calculated as follows:

The forward recursion must run for G steps along symbols and backward recursion have to run for extra M steps. Assume A as the reference point. And Separation d determines the ending point D of backward recursion. G determines both ending point of forward recursion, B and starting point of backward recursion. M determines warm-up starting point, C, of β. Thus, we can correlate all for points. Now, we will find the exact location of all points on k-t plane, as follows:

- Points A and D are located at k=0.
- Point B lies on k=G and point C lies on k=G+M.
- Now, if d ≥ G+M, then t_a=0 and t_c=d-G-M;

  Else, d<G+M, then t_c=0 and t_c=G+M-d.

  Hence, t_a = (-d+G+M+|d-G-M|)/2 and t_c= (d-G-M+|d-G_M|)/2.
- Next t_D= t_a + d, and t_B= t_a + G.
- Finally, point E is given by: k_E= k_D – G and t_E = t_D – G.

And further, any recursion pattern is only feasible if it satisfies at least one of following constraints [19]:

d≥ $2G \quad and \quad d \geq G + M$

$d \geq 2G \; and \; d < G + M$

$G \leq d < 2G \; and \; d \geq G + M$

$G \leq d < 2G \; and \; d < G + M$

$0 \leq d \geq G$

$d < 0$

Width and Height of any recursion pattern can be calculated as follows:

Width, W, is defined as follows: W=G+M,

And from feasible recursion patterns as shown above, height of a pattern is given as follows:

$$H=\begin{cases} d, & if\ d \geq G + M \\ G + M, & if\ G \leq d \geq G + M \\ G + (G + M - d), & if\ d \leq G \end{cases}$$

As a result:

H = (2G+M+|d - G|+|d-G-M|)/2

3.4.2.2 Determination of Numbers of recursion pattern

Total number of recursion patterns can be calculated from decoded bits per recursion and total number of bits, as $\lceil (L - 1/G) \rceil$.

3.4.2.3 Tiling the recursion patterns

Last step is to compose the architecture by tiling recursion patterns, Since, recursion patterns are tiled diagonally, tiling spacing has to be determined. Consider two patterns ABCD and A'B'C'D' as shown in figure above, and let (K_a',t_a') designates the coordinates of point A' of A'B'C'D'. Then quantity K_a' – K_a is the horizontal offset of A'B'C'D' from ABCD. Since forward recursion in pattern ABCD runs for G steps, forward recursion in pattern A'B'C'D' can start G steps later, resulting in K_a'-K_a=G.

The decoding delay of a single recursion pattern is its height plus one. Delay between two adjacent patterns is equal to G, i.e. number of valid metrics per recursion.

3.5 Parallel Window Architecture

In the SW-SISO MAP algorithm, we used only β recursion to increase the speed and storage optimization, but still the minimal decoding time is proportional to is L+M, where L is the length of data sequence, M is the warm-up period of β recursion. In SW algorithm, we have used diagonal tiling, now we will use both diagonal as well as lateral tiling and we will do this by using continuous mode operation. It is apparent from the DFGs of sliding window that delay achieved L+M by SW can only be optimized and more importantly, made independent of frame length. It should be noted that after a particular amount of optimization, interleavers becomes the performance bottleneck. A good work on interleavers has been done by [22,23]. The resulting DFG is called as Parallel Window DFG (PW-DFG).

Parallel Architecture for SISO MAP algorithm were firstly considered in [9], and later on, modified as D architecture [24] and X architecture [12] to reduce storage requirement. D architecture and X architecture corresponds to Lateral and Hybrid Tiling in terms of Tile Graph methodology. Basically, X architecture is a special case of Hybrid Tiling with two sliding windows per lateral window.

Basically, PW-MAP algorithm can be implemented by two different methodologies of architectures, named as:

- Lateral Tiling of recursion pattern

- Hybrid Tiling of recursion pattern

3.5.1 Lateral Tiling of recursion patterns

In case of SW-SISO, there was a constraint imposed to initialize the α recursion in each pattern using the exact α metrics from the previous pattern. In this case, we drop this constraint and make use of the continuous mode. We will start both α and β recursion by some approximation and after a certain warm-up length, we start the decoding process. This process is shown in figure 3.2 below:

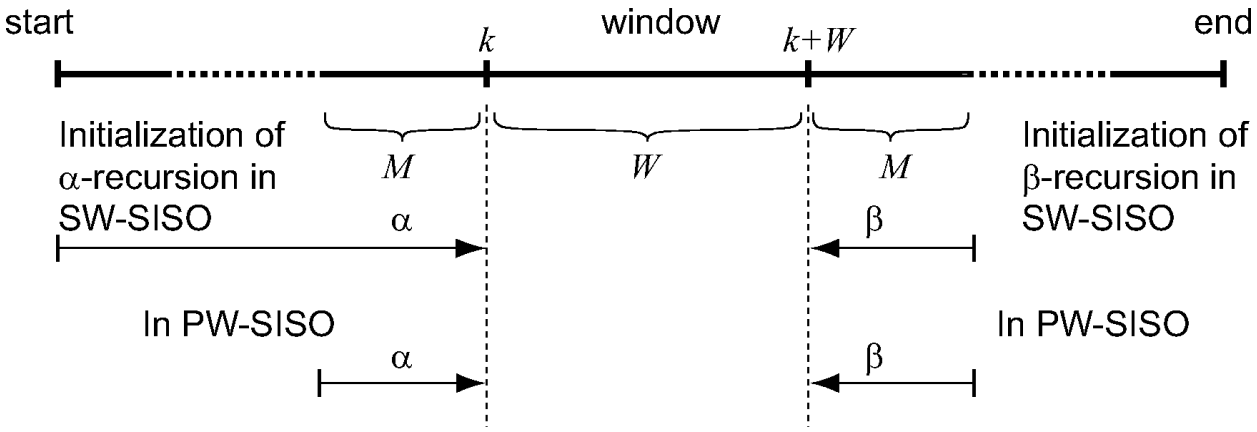

Figure 3.2: Initialization of α and β recursions in the SW-SISO algorithm versus the PW-SISO algorithm

β-recursion flow has its own warm-up phase window, which means that again as in SW-algorithm, recursion patterns can be tilled together. This recursion pattern is just same as the recursion pattern of SW architecture with added α warm-up phase.

As we have to tile the recursion patterns, we can better express the recursion pattern with their warm-up phases in the neighborhood windows as shown in figure 3.9 and 3.10.

Width of the recursion window, W= d=G taken is the most optimum window size. In case where M > W, say qW < M <= (q+1)W for some positive integer q, warm up phases has to be folded into q different windows. Decoding Delay for Lateral PW architecture is M+G.

3.5.1.1 Comparison between SW-SISO and PW-SISO MAP Algorithm

Decoding Delay: decoding Delay which is a function of length of data sequence L, can be L-M at its minimal, while at same time, delay in PW algorithm is just a factor of window width which is much smaller than decoding delay of SW- algorithm.

Storage Requirement: There is only a slight improvement in storage required compared to optimized SW-algorithm, that is due to the fact that no longer we have to save the last α-metrics of a recursion pattern for d clocks to be used in successive window.

3.5.2 Hybrid Tiling to reduce storage requirement

In the lateral tiling style of parallel window algorithm , we partitioned whole data sequence of length L into parts with each of length W, and decoded them in parallel, but as the sequence length get increased, again the latency get increased in terms of window size. So, we can further use the SW approach in each lateral window. The grouping of multiple diagonally-tiled patterns into a window and then tiling these windows further is termed as Hybrid Tiling [25].

A PW DFG which is employing a hybrid tiling can be characterized by two parameters, Window size W and number of recursion pattern per window ρ, with <=W. Figure 3.12 shows a Lateral PW DFG with hybrid tiling of ρ= 3.

Parameters (d,M,G) of recursion patterns in window are (W/ ρ, M,W/ ρ). interface the first pattern in each window to the last pattern of its adjacent window. In terms of decoding delay, delay of Hybrid tiled PW decoder is W+M for all values of ρ. And in terms of storage required, as the number of ρ decreases, the number of storage element required also decreases at the cost of increased area. A new technique using Both SOVA and MAP algorithm in a single architecture is discussed in [25], which sustains the performance but reduces complexity.

3.5.2.1 Comparison between Lateral and Hybrid Tiling

Decoding delay for both remains same to W+M; but in terms of storage elements required, Hybrid Tiling is much more efficient then Lateral tiling.

	Standard SISO MAP	SW-1 SISO MAP	SW-2 SISO MAP	SW-3 SISO MAP	Lateral PW SISO MAP	Hybrid PW SISO MAP
Decoding Delay	2L-1	L+2M+1	L+M+1	L-1+δ	M+G	M+G

Table 3.1: Comparison between Decoding Delay of all Architectures:

where $\delta\ refers\ to\ \frac{M+|d-G|+|d-G-M|}{2}$.

CHAPTER 4

VLSI IMPLEMENTATION OF TURBO DECODERS

In this chapter, we will discuss the VLSI implementation of different architectures discussed in the last chapter. We will start with the general block diagram of turbo decoders followed by standard Turbo decoder, Sliding Window(SW) and Parallel Window (PW) turbo decoders. Finally, we will discuss an area optimization technique for parallel window turbo decoders.

I have followed all 3gpp specification while implemented the design, which includes the usage of 8 stage encoders and QPP interleaver.

4.1 General block diagram of Turbo decoder

General block diagram of turbo decoder is shown below in figure 4.1. Block diagram shows that we receive the analog data from antenna at receiver end, convert this information into digital form using analog to digital converter (ADC). Then, de-multiplex this data into three different data parts named as systematic data, parity 1 and parity 2; and save these data elements into three different memories. From here, we process 2 out of 3 data elements at one time, we choose these elements using Multiplexer, and data elements choosed are depend upon the control signals coming from turbo control unit.

From here on, Log map decoder process the sequence of these two data elements as a complex computational process and stores the output a-posteriori (app) information into LLR RAM where LLR stands for logarithm of likelihood ratio. Then, again we choose two data elements from multiplexer, and send them to Log map decoder and simultaneously sends the app information of last stage using interleaver. Again, MAP decoder decodes the data sequence and generates a-posteriori information which is again get saved to LLR RAM. After a few number of iterations, we make the final decision using Logarithm of likelihood ratio, and transmits as the final decoded bits.

4.2. Implementation consideration

When we start to implement any architecture into hardware, we have to take care of many factors which affect a lot the implementation. Following are the some factors which i took into consideration while designing the turbo decoder:

Algorithm used: There are three variant of MAP algorithm, knows as standard map algorithm, log map algorithm and max log algorithm. Out of these three, Log map algorithm provides the best trade-off between performance and hardware complexity, as will have already discussed in previous chapter.

Data Representation: Data can be represented into two forms in digital circuits either in fixed point representation or floating point representation. Floating point number system provides us better range but hardware complexity increases rapidly. So, we have used fixed point number system in the implementation.

Arithmetic Used: As per the requirement of the algorithm, we have to use signed number, where we can use either 1's complement or 2's complement arithmetic. Here in this implementation, I have used 2's complement arithmetic.

HDL used: There are two main HDLs used to define hardware are VHDL and Verilog. I have used Verilog to implement my design.

4.3 Standard SISO MAP Turbo Decoder

In standard SISO MAP turbo decoder, we start with calculation of Branch matrices from the input data, after that these branch matrices are used in the α-MPU (Matrices Processing Unit) where we calculate alpha matrices and store the alpha matrices for whole sequence into α-RAM. After this, we again start to calculate the Branch matrices, and use them in β-MPU to calculate beta matrices and use these beta matrices, these branch matrices and alpha matrices from α-RAM to calculate the final log-likelihood ratio(LLR) and store these values into LLR RAM from where it is used again in next iteration.

Description of different blocks is given below:

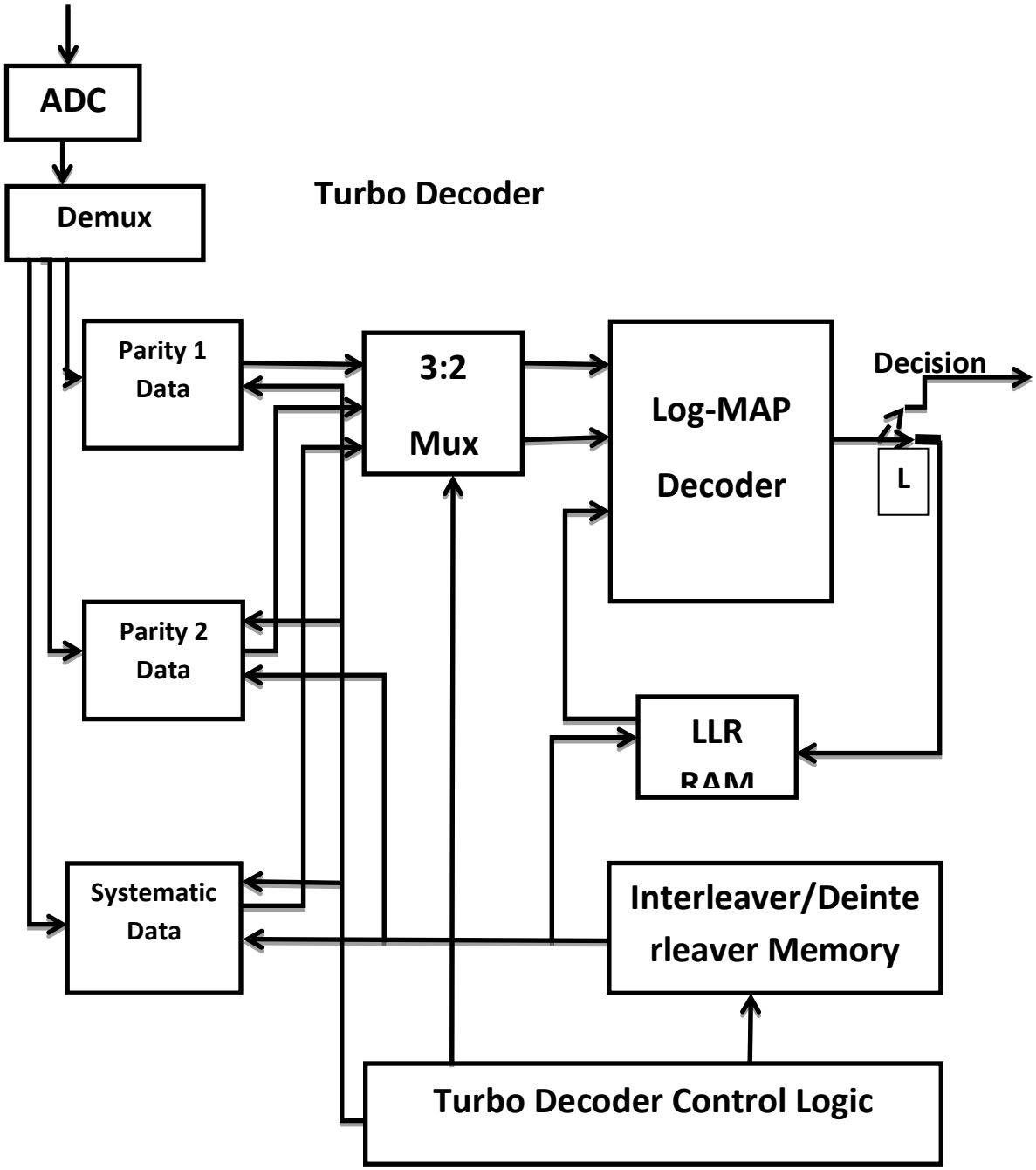

Figure 4.1: General Block Diagram of VLSI Turbo Decoder

4.3.1 γ *Calculattion Unit* As we have assumed the soft inputs, and we have to calculate Euclidean distance as mentioned in second unit, which includes just the usage of two adders to compute the branch matrices as can be seen by the equation of Euclidean distance 2.1 from chapter 2.

4.3.2 α/β–Matrices Processing Unit

Equations for alpha and beta matrices proves that there implementation is quite identical, So in spite of discussing them differently, we can explain them in a single discussion. The implementation diagram for Alpha/Beta MPU is given above in figure 4.2.

From the diagram, we can see that we choose two branch matrices out of four and similarly two α/β matrices out of available eight matrices and process them in ACS (Add-Compare-Select) unit and store the calculated value for further use in a register.

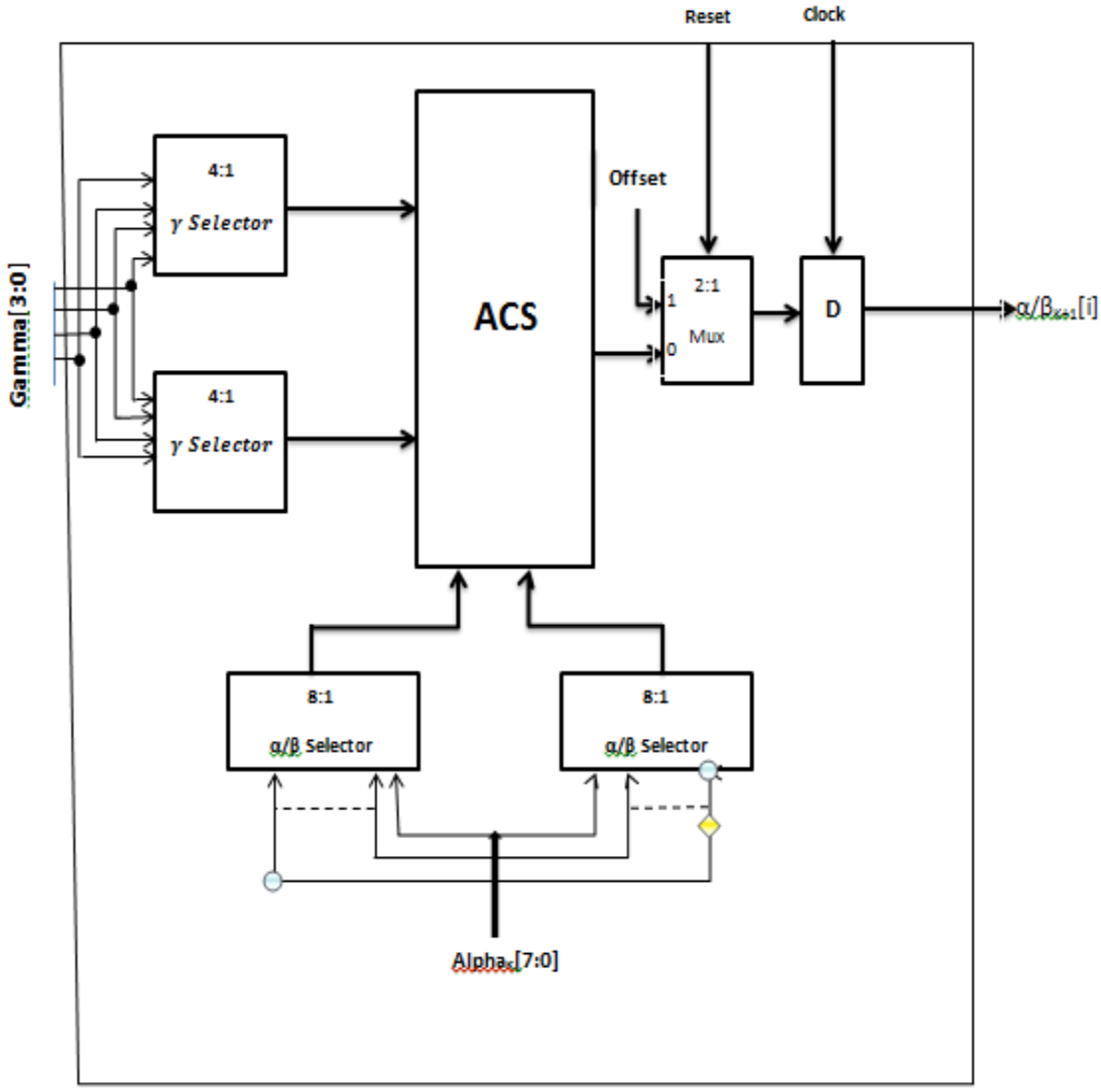

Figure 4.2: implementation of α/ β–Matrices Processing Unit (MPU)

4.3.3 Add-Compare-Select (ACS) Unit

ACS is the main computational unit of the α/β MPUs. It takes the input from the four different multiplexers and performs binary operations over them. This Block is the main bottleneck for the high speed turbo decoder and reason behind this is that pipelining can not be applied to this operation and at the same time, standard retiming techniques can not be applied to this circuit as there is only one delay in it; but still, performance of this block can be improved by using some special retiming technique, that retiming architecture will be discussed later in the same chapter. Implementation of standard ACS is shown below in figure 4.3.

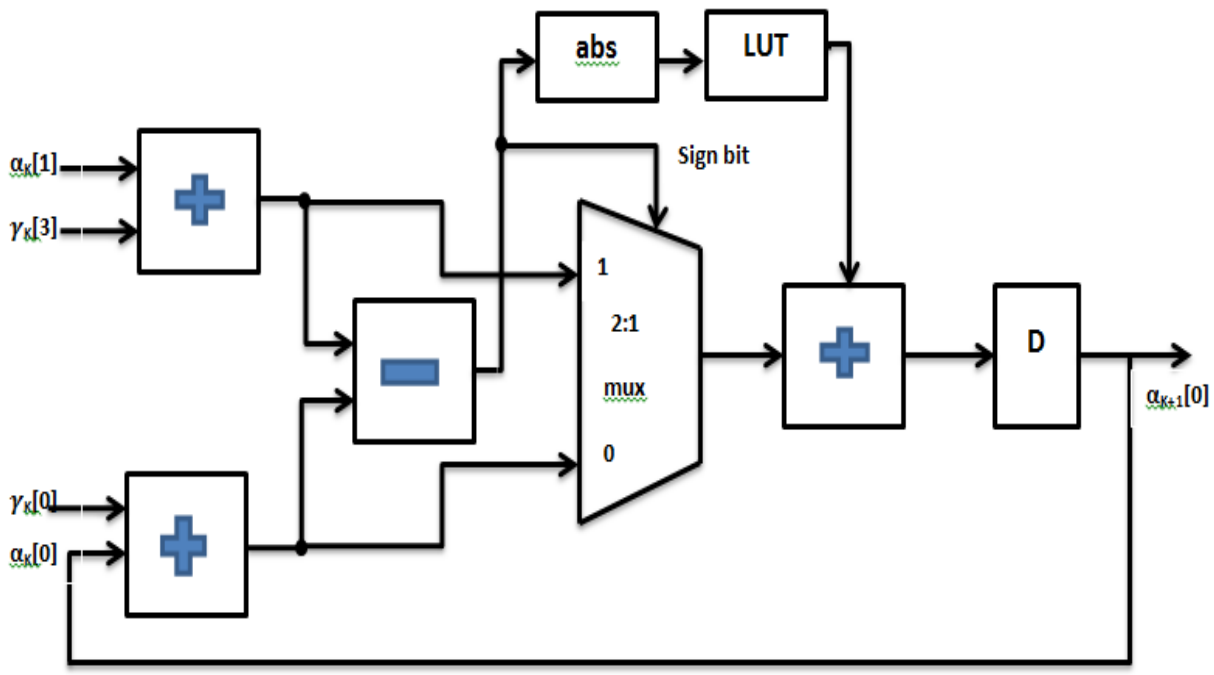

Figure 4.3: Implementation of Standard ACS Unit

4.3.4 LLR (Logarithm of Likelihood Ratio) Unit

This unit takes its three inputs from three different block names as alpha RAM, Beta MPU and BMG unit. This unit processes these three data elements and correspondingly provides a soft value of approximation whether the bit transmitted was a 1 or a 0. This a-posteriori probability (app) is saved into LLR RAM and used in the next iteration as the a-priori information. Implementation diagram of LLR unit is shown in figure 4.4.

4.4 Sliding Window (SW) SISO MAP Decoder:

As we have already studied in previous chapter that Standard SISO MAP decoder suffers from the major problem of high latency and huge memory requirement, we shift from terminated frame mode of communication towards truncated and continuous mode of communication. In the sliding window we use the truncated mode of communication.

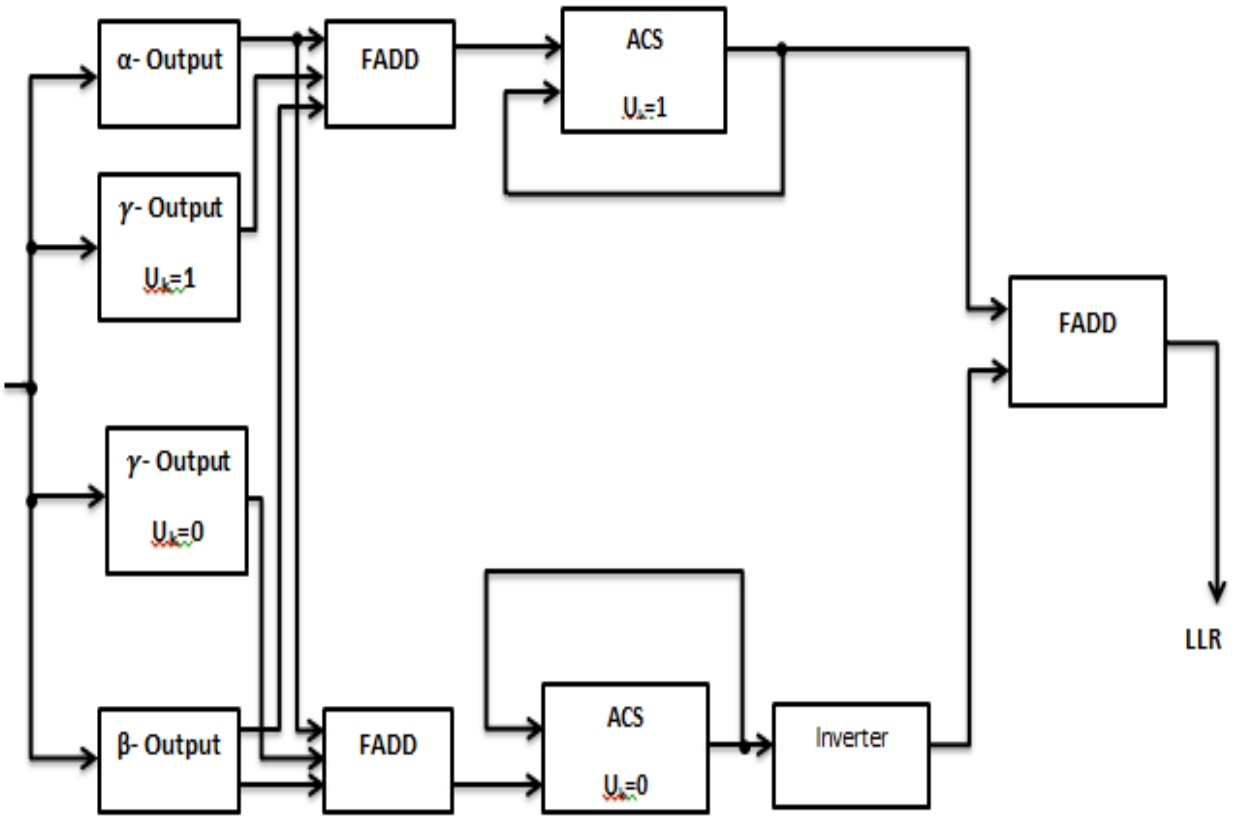

Figure 4.4: Implementation Diagram of LLR Unit

If we compare the implementation of SW SISO MAP decoder with the standard SISO MAP decoder, we can easily find out that hardware requirement is higher than in case of Standard SISO MAP decoder; It is primarily due to the fact that time multiplexing has been used in this implementation to reduce the latency.

From the diagram, we can find out that this decoder calculates the alpha matrices in a very similar way to the standard turbo decoder, but there is a lot of change for beta matrices computation.

Main difference for beta computation is that in spite of only one beta MPU, we used here three MPUs, where first two works in the harmony to each other and the third one is used to calculate the beta matrices for the last window in which case we have termination matrices available.

Next we will discuss the hardware implementation of QPP interleaver which is the standard interleaver used according to the 3gpp specifications [26].

4.5 Hardware Implementation of QPP Interleaver

As we know that in turbo decoding, interleaved addresses are generated in consecutive orders. So we can generate the QPP interleaved addresses in recursive pattern.

Assume, starting address to calculate interleaving address is x_0, we firstly pre-compute $f(x_0)$ as:

$$f(x_0) = (f_2.x_0^2 + f_1.x_0) \bmod N \quad (4.1)$$

Now, in the next cycle, x will be increased by value j, we can compute f(x+j) recursively as:

$$f(x+j) = (f_2.(x+j)^2 + f_1.(x+j)) \bmod \quad (4.2)$$

$$= (f(x) + y(x)) \bmod N \quad (4.3)$$

Where y(x) is defined as:

$$y(x) = (2.j.f_2.x + j^2.f_2 + j.f_1) \bmod \quad (4.4)$$

now, y(x) again can be calculated recursively as:

$$y(x+d) = (y(x) + (2.j^2.f_2)) \bmod N \quad (4.5)$$

where initial value of y(x) can be calculated as:

$$y(x_0) = (2.j.f_2.x_0 + j^2.f_2 + j.f_1) \bmod N \quad (4.6)$$

Now, we know that from equation 4.3 and 4.5, that mod operation is highly complex to implement in hardware, but as we know that maximum value of f(x) or y(x) can have maximum value equal to N, so we can calculate the mode just as a subtraction.

For these calculation, we need three pre-computed values of $f(x_0)$, $y(x_0)$ and $(2.j^2.f_2) \bmod N$.

The hardware to compute forward interleaved address is shown in figure 4.5.

Similarly we can calculate the interleaved address in reverse direction as:

$$f(x-d) = (f(x) - y(x-d)) \bmod N \quad (4.7)$$

Similarly from forward interleaving, we can calculate y(x-d) as:

$$y(x-d) = (y(x) - 2.j^2.f_2) \bmod N \quad (4.8)$$

The hardware implementation to compute forward interleaved address is shown in figure 4.6.

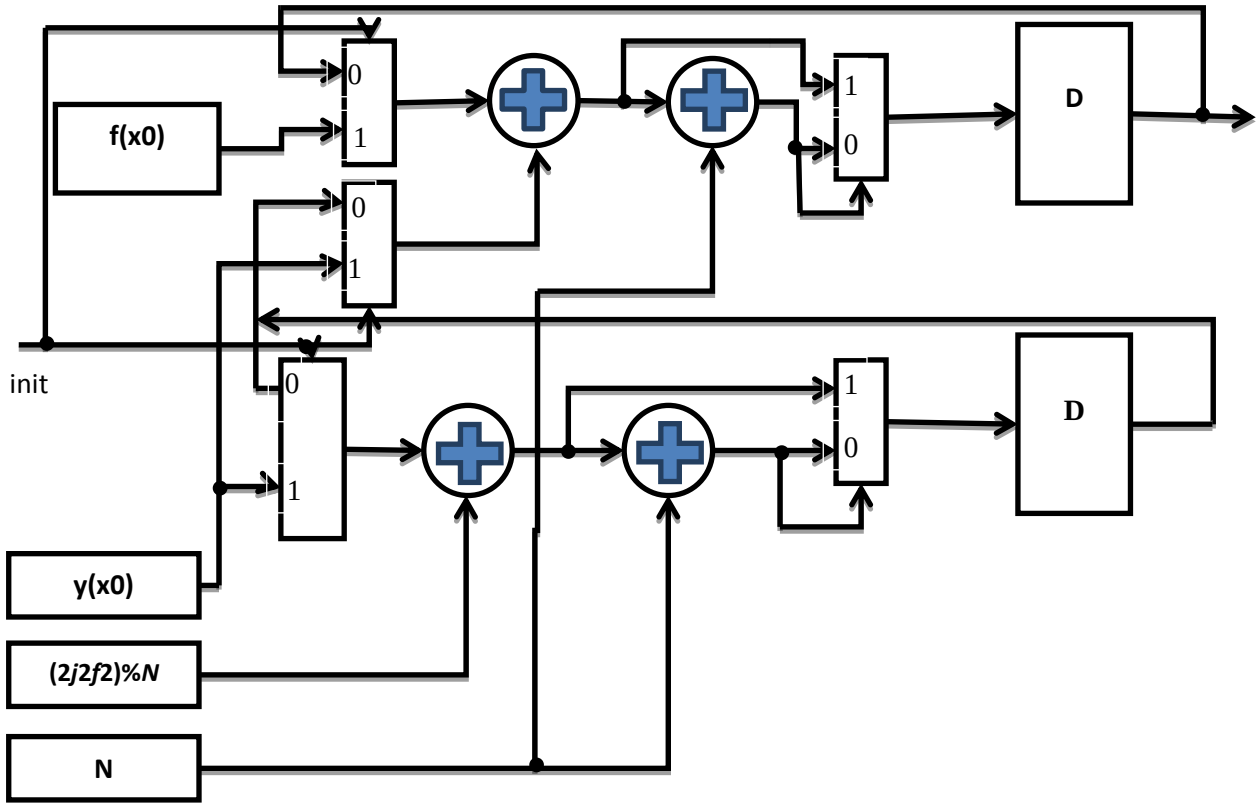

Figure 4.5: Hardware Implementation of Forward QPP Address Generator

4.6 Parallel Window SISO MAP Decoder

As we can see from implementation, there are four parallel BMG, beta, alpha and LLR units are present, which parallel executes the data sequence in four parts. As, in every clock pulse, we have to access four data elements in parallel, we have to use four port RAM. Similarly, to write four apps in every cycle from LLR, we have used four port LLR RAM.

Main advantage of using the parallel Window turbo decoder is that it increases throughput as a direct function of parallel windows working.

I have implemented the lateral parallel window turbo decoder, we can further reduces the latency of the decoder by using the hybrid parallel window turbo decoder.

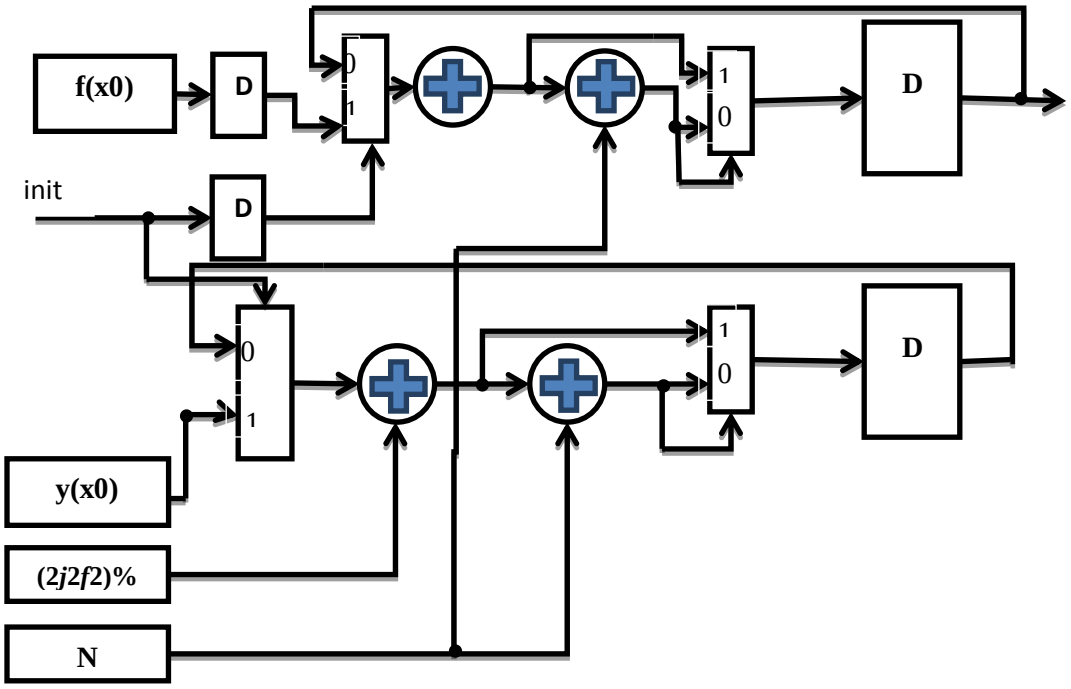

Figure 4.6: Hardware Implementation of Backward QPP Address Generator

4.7 Block Level Improvements

By the time, we have done all the up gradations in standard window architecture at system level. Now, to further improve the speed, we upgrade the different blocks of turbo decoder, which helps us to reduce critical path and hence to increase the maximum operating frequency. Following are the two improvements:

4.7.1 Improved LLR (LOG of Likelihood Ratio) Unit:

As we can see from figure 4.5 that LLR unit is the longest computational unit and further it is composed of four cascaded units, now if use pipelining after every cascade structure and convert the block from one stage to four stage pipelined block, we would be able to increase our system performance up to three times.

4.7.2 Retimed ACS (Add Compare Select) Unit:

Next to LLR unit, ACS unit is the second computationally largest unit, and further problem is that pipelining can not be applied to this unit; So there is very less that we can do to improve the speed of this unit. A retimed ACS unit [15] is proposed based on special retiming technique can reduce the delay of ACS unit from three adders and one LUT to two adders and one LUT.

CHAPTER 5

SIMULATION AND SYNTHESIS

In this chapter, we will discuss the simulation and synthesis of the turbo decoder's different architectures.

5.1 Description of Turbo Decoder architectures

I have implemented a radix 2 turbo decoder and used the block length of 80.

I have implemented six different architectures of turbo decoders as follows:

1. Standard Turbo decoder
2. Sliding Window (SW) Turbo decoder
3. Parallel Window (PW) based Turbo decoder
4. Parallel Window Turbo decoder using improved Interleaving technique
5. Parallel Window based turbo decoder with pipelining
6. Retimed Parallel Window based turbo decoder with pipelining

Parallel Window based Turbo decoder: I have used four parallel windows in this architecture.

Parallel Window Turbo decoder using improved Interleaving technique: In this architecture, I have used a single QPP interleaver and calculated remaining three addresses using adders.

Parallel Window based turbo decoder with pipelining: In this architecture, I have used four stage pipelining in LLR unit and simultaneously used two stage pipelining i.e. one delay between branch matrices generator (BMG) and Alpha MPU to reduce critical time.

Retimed Parallel Window based turbo decoder with pipelining: In this architecture, to further reduce the critical path, I used a retimed design of ACS (Add Compare Select) unit.

Tools Used: I have used ModelSim 10.1 Student edition for simulation, and used Xilinx ISE 13.1.

5.2 Synthesis Results on FPGA

The Xilinx ISE is used for implementation of all the circuits. The Working Environment for the design is :

- Target Device xc4vlx200-11ff1513
- Technology : 90 nm
- Tool Version : ISE 13.1
- FSM Style : LUT
- Optimization Goal : Speed
- Design Strategy : Balanced
- Total Slices 89088
- Total Slice Flip-Flops: 178176
- No. of 4 input LUTs: 178176

Table 5.1 shows the synthesis result for different architectures on Xilinx ISE 13.1:

	Standard Turbo Decoder	Sliding Window based approach	Parallel Window approach	Parallel window approach with improved interleaving	Parallel Window approach with pipelining	Parallel Window approach with retiming
No. of slices used	12331	17476	23100	23061	23314	27143
No. of slice ffs	13123	20066	18756	18714	20453	25775
No. of 4 i/p LUT	10936	19891	42725	42664	42985	49704
Critical Path Delay	33.398 ns	33.398ns	30.672ns	30.672 ns	9.951 ns	7.892 ns
Max. operating frequency	29.9 MHz	29.9 MHz	32.6MHz	32.6 MHz	100.49 MHz	127.7 MHz

Table 5.1: Synthesis Result for Different Architectures

5.3 Analysis of Synthesis Results

From the synthesis result of both Xilinx ISE and Synopsys Design Compiler, we can make following analysis:

5.3.1 Area Consumption

Standard turbo decoder takes the least design area, after it Sliding window, parallel window, parallel window with pipelining and then retimed parallel window consumes the area in incrementing order.

5.3.2 Speed

From these synthesis results, we can only get a little idea about speed i.e. about maximum operating frequency. We will discuss about speed later in this chapter when we will discuss the simulation results. We can see from table 2 that maximum operating frequency is same for first four cases, this is due to the reason that the block with critical path is same in all four architectures, but PW with pipelining architecture, I used four stage pipelining in block with critical path, due to which operating frequency increased and in last, Retimed PW architecture, In this architecture, I used a retimed architecture of new critical path block, in which pipelining was not possible, So it further increases maximum operating frequency.

5.3.3 Power Consumption

From the synthesis result , we can make simple analysis that power consumption increase is directly proportional to the area consumed, but if we closely analyze last three architectures i.e. PW, PW with pipelining and retimed PW, we notice that areal increment is very less, but power consumption varies hugely.

Now, if we notice these architectures, we see that area is of course not increasing, but maximum operating frequency is varying over a wide range and same is the reason for varying power consumption.

Performance Analysis

I have worked for speed improvement in my research work. Following are the two main factors which represents performance of a system: (i) Latency

(ii) Throughput

Latency: Latency represents the time difference between when we provide the system input to the time when it started to respond to those inputs.

Throughput: Throughput represents the number of operations performed per unit time or in our case number of bits decoded per unit time.

Below in Table 5.2, we represent Latency and Throughput in terms of clock cycles.

I have decoded a sequence of 80 bits, So the throughput would be in terms of bits decoded per clock cycle.

	Standard Turbo Decoder	Sliding Window based approach	Parallel Window based approach	Parallel window approach with pipelining	Parallel window approach with retiming
Latency	172	128	91	94	94
Throughput (single iteration) (bits/cycle)	80/674 =0.1186	80/594 =0.1346	80/266 =0.3007	80/286 =0.2797	80/286 =0.2797

Table 5.2: Latency and Throughput in terms of clock cycle

From Table 5.3, we can notice that latency improves from standard to SW to PW architectures, but there is no improvement, even deteriorates a little for the last two cases. Similarly, in case of throughput, improvements are clear from Standard to SW to PW but for last two architectures again throughput decreases, then how last two structures are advantageous.

It is right that last two structures take little extra cycles, but how they improve performance is depends upon the factor that they improve performance by using higher operating frequencies which is not possible for former architectures.

Now if we take a look on latency and throughput in terms of time as shown in table 5.4, which is based upon maximum operating frequencies as shown in table 5.2, it would become more clear.

Below are given the simulation results corresponding to Standard, SW, PW and PW with pipelining architectures; simulation results for PW with improved interleaver are similar to PW architecture and retimed PW have results similar to PW with pipelining , so there results has been skipped.

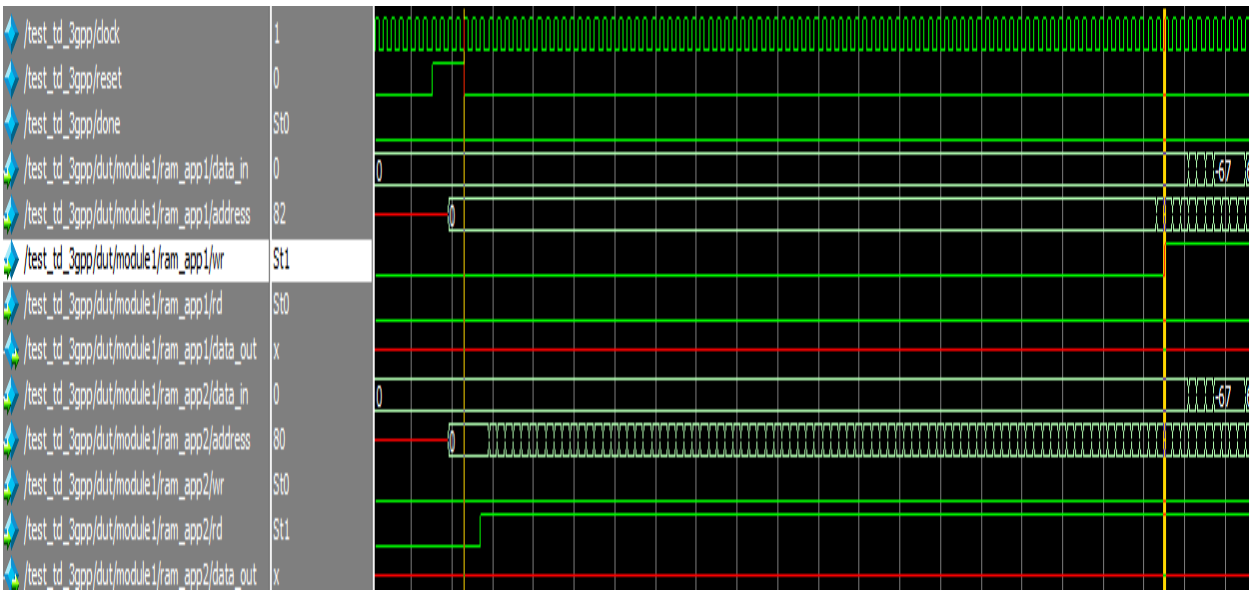

Figure 5.1 : Simulation Diagram Corresponding to latency for Standard SISO MAP Turbo Decoder

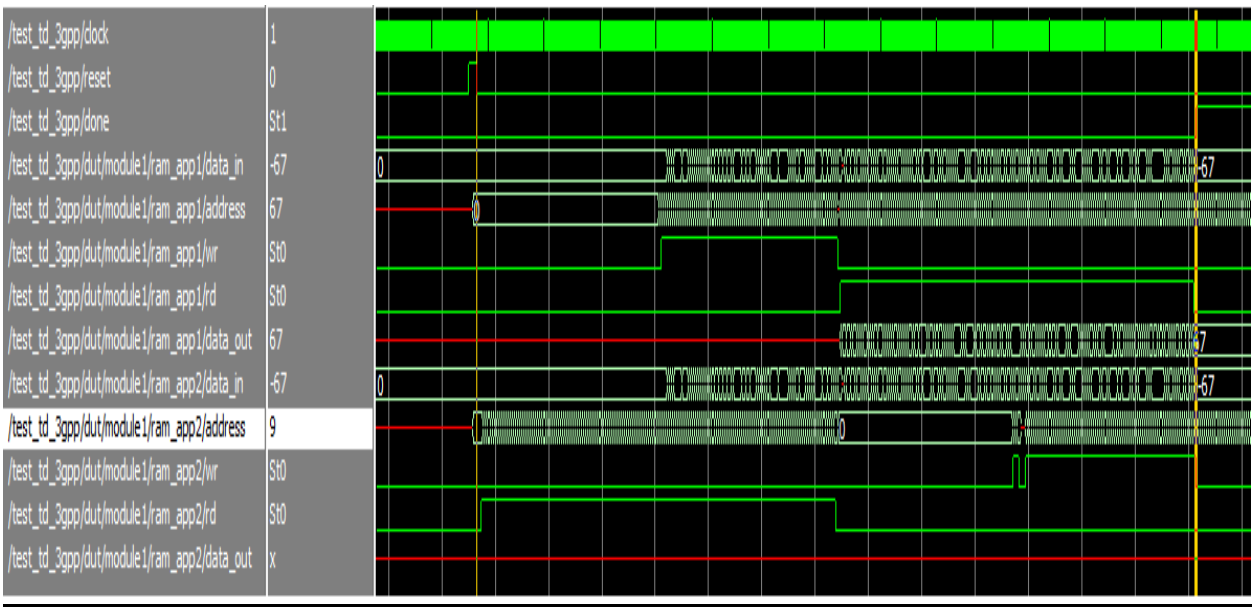

Figure 5.2 : Simulation Diagram Corresponding to throughput for Standard SISO MAP Decoder

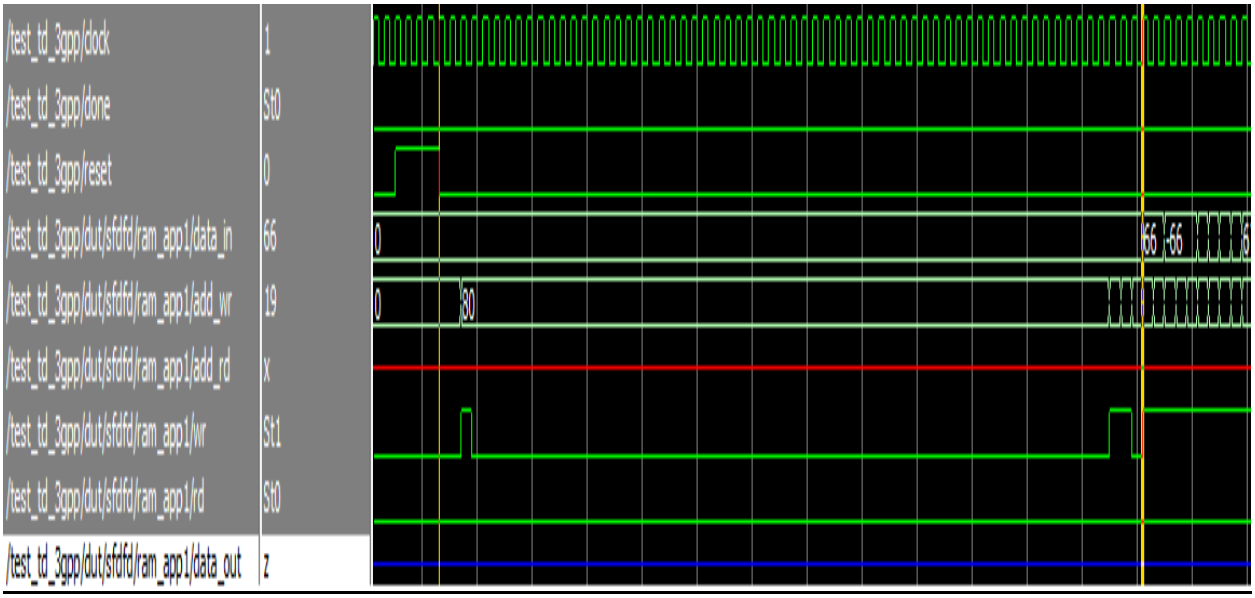

Figure 5.3 : Simulation Diagram Corresponding to latency for SW SISO MAP Turbo Decoder

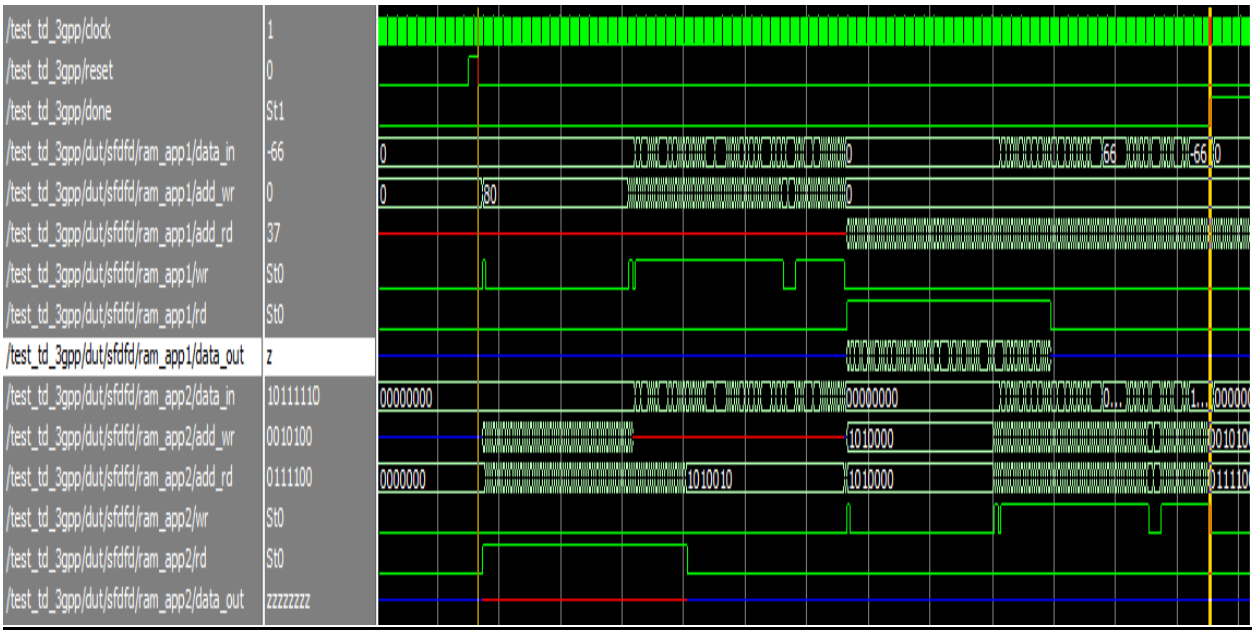

Figure 5.4 : Simulation Diagram Corresponding to throughput for SW SISO MAP Turbo Decoder

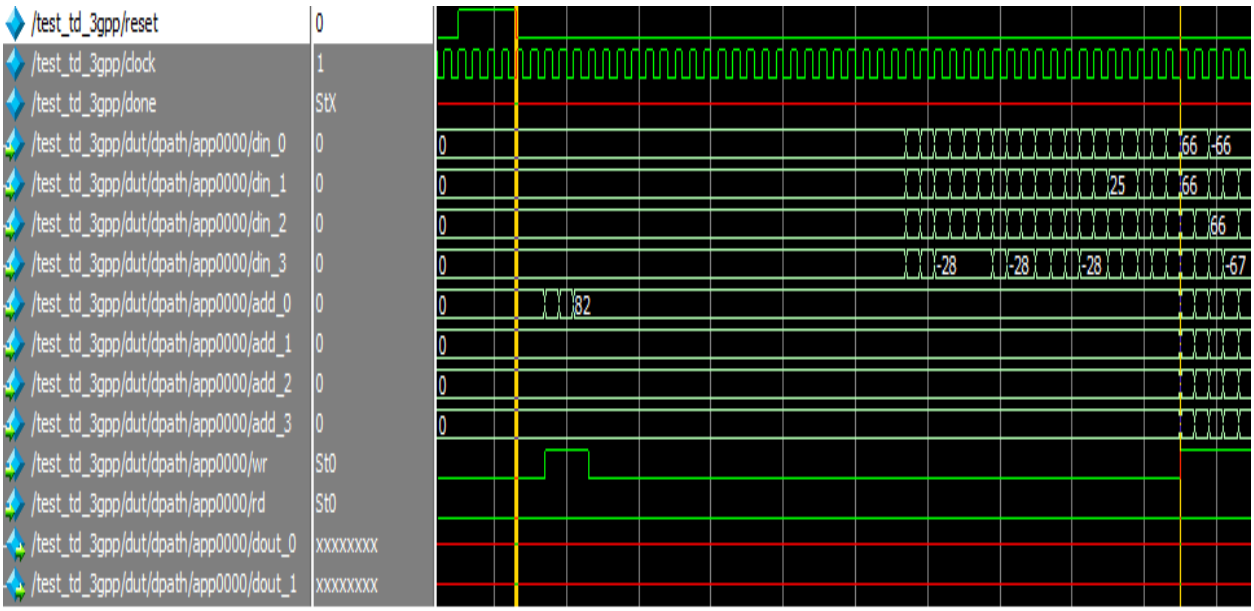

Figure 5.5 : Simulation Diagram Corresponding to latency for PW SISO MAP Turbo Decoder

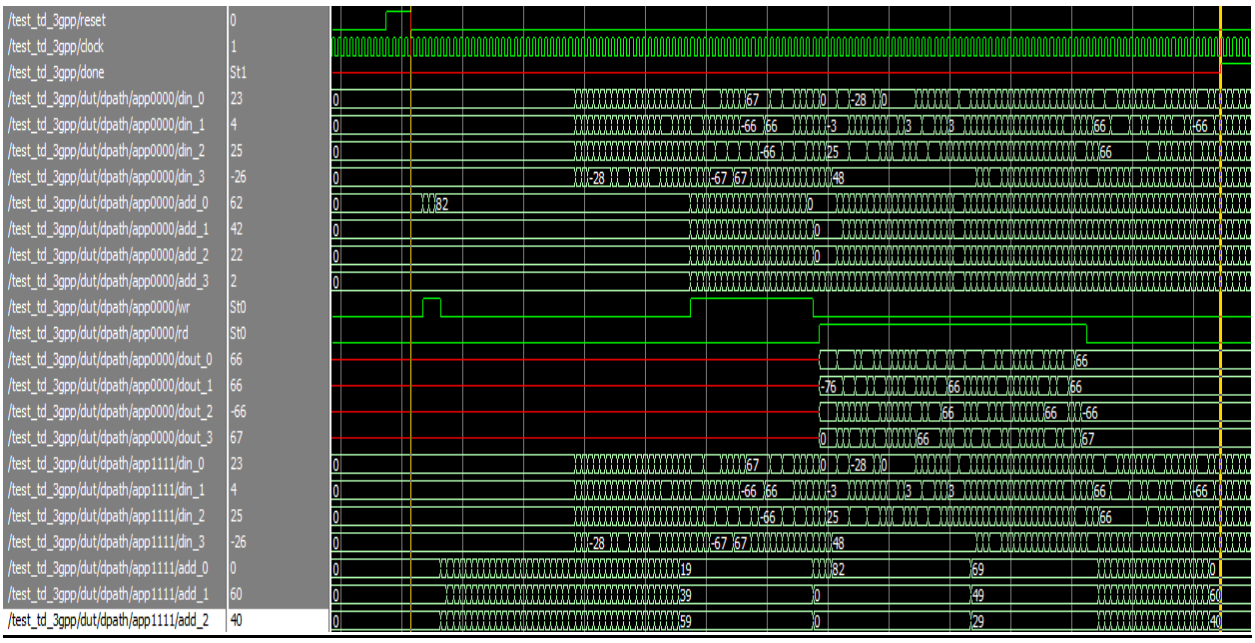

Figure 5.6 : Simulation Diagram Corresponding to throughput for PW SISO MAP Turbo Decoder

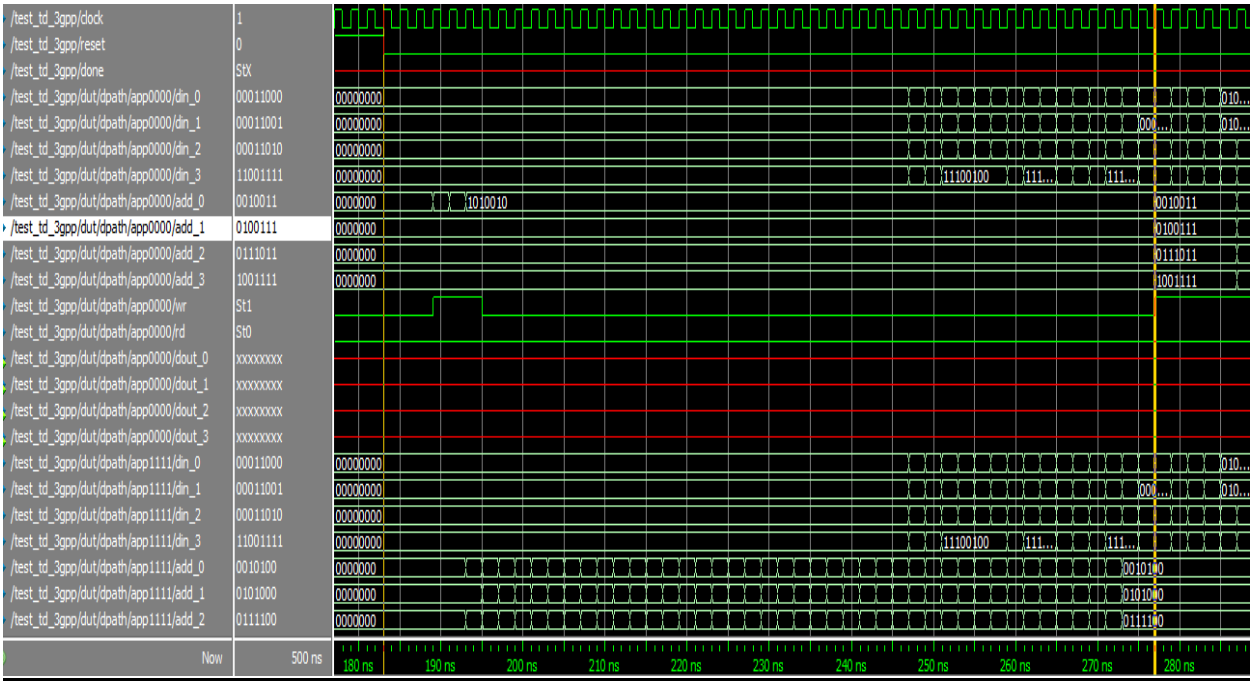

Figure 5.7 : Simulation Diagram Corresponding to latency for PW with Pipelining SISO MAP Turbo Decoder

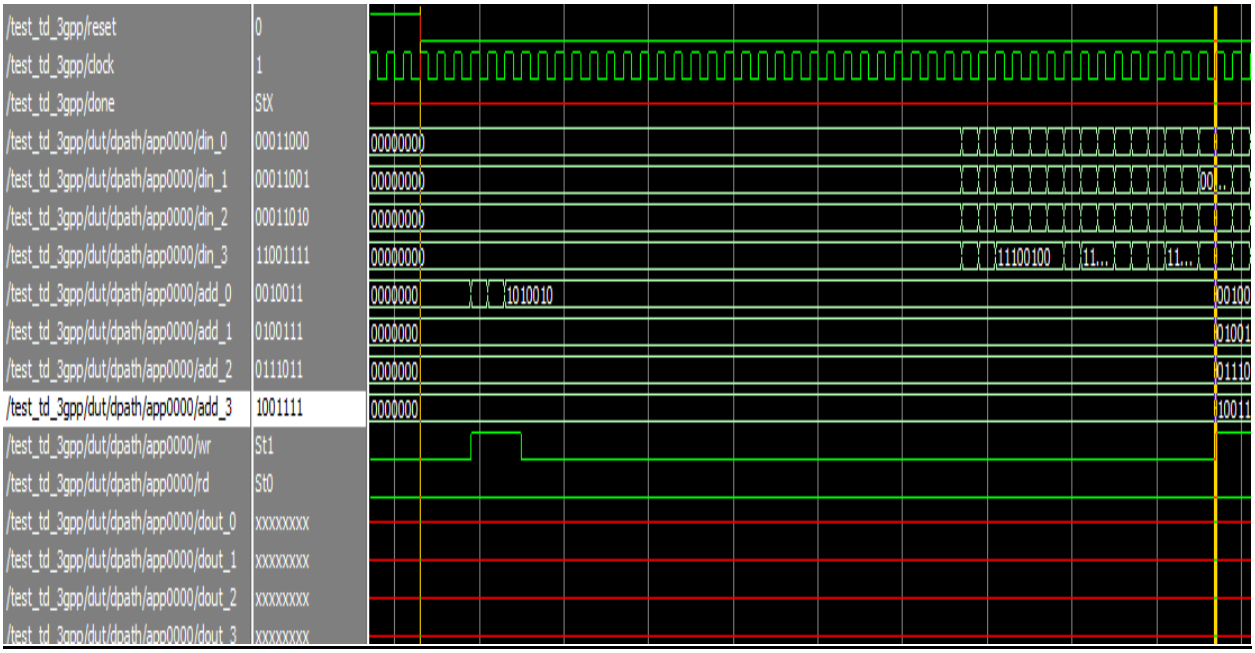

Figure 5.8 : Simulation Diagram Corresponding to Throughput for PW with Pipelining SISO MAP Turbo Decoder

CHAPTER 6

CONCLUSION

This chapter concludes what have been done in thesis and what can be done in future.

6.1 Conclusion

Design of Standard, Sliding window (SW), Parallel Window, Parallel Window with pipelining and retimed Parallel Window have been synthesized over Xilinx ISE for Virtex 4, and it has been analyzed that area requirement increases as we move from standard towards Sliding window and parallel window architectures, but at the same time speed also increases in direct proportion with the complexity or area.

	Standard Turbo Decoder	Sliding Window based approach	Parallel Window based approach	Parallel Window based approach using improved interleaving	Parallel Window based approach with Pipelining	Parallel Window based approach with Retiming
No. of slices used	12331	17476	23100	23061	23314	27143
No. of slice ffs	13123	20066	18756	18714	20453	25775
No. of 4 i/p LUT	10936	19891	42725	42664	42985	49704
Critical Path Delay	33.398 ns	33.398ns	30.672ns	30.672 ns	9.951 ns	7.892 ns
Max. operating frequency	29.9 MHz	29.9 MHz	32.6MHz	32.6 MHz	100.49 MHz	127.7 MHz

Table 6.1: Synthesis Result for Different Architectures on Xilinx ISE 13.1

It has been analyzed too, that we can achieve an effective increase in if we use pipelining and retiming while keeping the area same as the lower counterpart.

I also analyzed the latency and throughput improvement as we move from Standard architecture to more advanced architectures. These results are shown below in table 6.1.

6.2 Future Work

In this thesis work, main design consideration was the improvement of the speed, but as turbo codes are mainly used in portable devices, a lot can be done about improvement in power consumption.

Another area for future work could be the designing of Radix-4 Turbo decoders, which can provide throughput improvement up to a factor of two with additional complexity.

References

[1] Claude Berrou, Alain Glavieux and Punya Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes,"IEEE International Conference on Communication, May 1993, pp. 1064–1070.

[2] L. Bahl, J. Cocke, F. Jelinek and J. Rajiv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. On Inform. Theory, vol. IT-20, pp. 284–287, Mar.1974.

[3] C.E.Shannon, "A Mathematical Theory of Communication", Bell System Technology Journal,Vol. 27, pp.379-423, July-Oct 1948.

[4] G. D. Forney, Jr., "Concatenated codes", MIT Press, 1966.

[5] Andrew J. Viterbi, "Convolution codes and their performance in communication system", IEEE Transaction on Comm. Technology, Vol. 5, October 1971, pp. 751-772.

[6] Patrick Roberston, Emmanuelle Villebrun, and Peter Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in Proc. IEEE Int. Conf. Communications, 1995, pp. 1009–1013.

[7] Andrew J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," IEEE J. Select. Areas Commun., vol. 16, pp. 260–264, Feb. 1998.

[8] Sergio Benedetto, D. Divsalar, Guido Montorsi and F. Pollara, "Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes," JPL, TDA Progress Report 42-124, Feb. 1996.

[9] H. Dawid, G. Gehnen, and H. Meyr, "MAP channel decoding: algorithm and VLSI architecture," in Proc. Workshop VLSI Signal Processing VI, 1993, pp. 141–149.

[10] Yufei Wu, B. D.Woerner, and T. K. Blankenship, "Data width requirements in SISO decoding with modulo normalization," IEEE Transaction of Communication, vol. 49, Nov. 2001.

[11] Mohammad M. Mansour and N. R. Shanbhag, "Design methodology for high speed iterative decoder architectures," in Proc. ICASSP, Orlando, FL, May 2002, pp. 3085–3088.

[12] Alexander Worm, Holger Lamm, and Norbert Wehn ,"VLSI architectures for high-speed MAP decoders," in Proc. 14th. VLSI Design, 2001, pp. 446–453.

[13] Yanhui Tong, Tet-Hin Yeap and Jean Yves Chouinard, "VLSI Implementation of a turbo decoder with LOG-MAP-based iterative decoding", IEEE Transaction on Instrumentation and Measurement, Vol.13, August 2004.

[14] Mohammad M.Mansour, Naresh R. Shanbhag,"VLSI Architectures for SISO-APP Decoder", in IEEE Transactions on VLSI Systems,Aug. 2003, pp. 627-650.

[15] Zhongfeng Wang and Xinming Huang, "VLSI Architectures for Turbo decoders", VLSI, 1st edition, In-Tech Publication. Feb. 2010.

[16] Yang Sun and Joseph R. Cavallaro, "Efficient Hardware Implementation of a Highly-Parallel 3gpp LTE/LTE Advanced Turbo decoder", Integration, The VLSI Journal, Vol. 44, pp. 305-315.

[17] Bernard Sklar, "Digital communications: Fundamentals and Applications", Second Edition, Prentice Hall Publication, January 2001.

[18] Mihai Timis,"Design of LOG-MAP/ MAX-LOG-MAP Decoder," in Annals. Computer Science Series, 5th Tome 1st Fasc. -2007, pp.63-69.

[19] Sanjay Sharma, Sanjay Attri and R.C. Chauhan, "A Simplified and Efficient Implementation of FPGA Based Turbo Decoder," IEEE International Proceedings of Performance, Computing, and Communications Conference, 2003,pp. 207-213.

[20] H. Dawid and H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," in Proc. Personal, Indoor, and Mobile Radio Communications, PIMRC'95.Wireless: Merging onto the Information Superhighway, vol. 1, 1995, pp. 193–197.

[21] Sergio Benedetto and Guido Montorsi, "Soft-input soft-output modules for the construction and distributed iterative decoding of code networks," Eur. Trans. Telecomm., vol.-9, 1998, pp. 155–172.

[22] Oscar Y. Takeshita,"On Maximum Contention-Free Interleavers and Permutation Polynomials Over Integer Rings Oscar," IEEE Transactions on Information Theory, VOL. 52, 2006, pp.1249-1253.

[23] Oscar Y. Takeshita, "A New Metric for Permutation Polynomial Interleavers,", IEEE International Symposium on Information Theory, 2006, pp-1983-1987.

[24] Alexander Worm, Holger Lamm, and Norbert Wehn, "A high-speed MAP architecture with optimized memory size and power consumption," in Proc. IEEE Workshop Signal Processing Systems (SiPS)2000, 2000, pp. 265–274.

[25] Zhenchuan Zhang, Nana Jiang, "Design and Performance Analysis of an Improved Decoding Algorithm for Turbo Codes", 7th International Conference on Wireless communications, Networking and Mobile Computing 2011, pp.1-4.

[26] General UMTS Architecture, 3GPP TS 23.101 version 7.0.0, June 2007.