UNIVERSITY OF GÄVLE

FACULTY OF ENGINEERING AND SUSTAINABLE DEVELOPMENT

Enhancement of Viterbi decoder

Syed Haider Abbas

January 2014

Master's Thesis in Electronics (15 credit hrs)

Master's Program in Electronics/Telecommunications

Examiner: José Chilo

Supervisor: José Chilo

Preface

I would like to thank my supervisor for his guidance and encouragement to make me work hard and smartly. I found him very helpful while discussing the optimization issues in this dissertation work. His critical comments on my work certainly made me think of new ideas and techniques in the field of optimization and software simulation.

Finally, it should not be fair not to acknowledge the support of our parents and friends, who stood behind us all the times, both financially and morally.

Abstract

The thesis work designs and implements an optimized solution for a widely used forward error correction algorithm known as Convolutional Encoder with Viterbi decoder. Convolutional encoding with Viterbi decoding is a good forward error correction technique suitable for channels affected by noise degradation. After the implementation of viterbi decoder, another decoder based on Finite State Machine has been successfully implemented. The new decoder performs significantly better than the Viterbi decoder in terms of error correction capability for some tackled cases.

The main thesis objective was to design a solution which is more optimized with less complexity and efficient in terms of area utilization and maximum operational frequency respectively.

The motivation of this thesis has been derived from the concept of improving a error correction technique in some specific areas. For this initially we implemented viterbi decoder and after that we marked the areas which can be improved like processing time and device utilization. It is also observed during the implementation of viterbi decoder that, when there is continuous burst of errors viterbi decoder malfunctions and more redundancies are involved during the path calculation. Where as in the case of random distributed errors viterbi involves computational complexity.

A successful solution by changing the architecture design, and then by following a new scheme in path calculation. Which results in efficient device utilization and better timing. Due to this we are able to reduce the redundancies and improve the factors like device utilization, computational complexity and processing time. Also from the output results of new desing decoder its is observed that when there is random distributed errors, the new decoder is correcting more errors as compared to conventional viterbi decoder.

To obtain this objective various platforms were used. Verilog (hardware language) is used for the implementing both decoders. The second platform used is Xilinx ISE ® (targeted device is SPARTAN III from S200 Series and a speed grade of 4), this is used to obtain the synthesis reports.

A chapter on Testing later in the report gives a detailed comparison of the two implementations through device utilization and timing summaries.

Abbreviations

MLE – Maximum Likelihood Estimation

FEC – Forward Error Correction

BEC – Backward Error Correction

ARP – Automatic Repeat Request

VA – Viterbi Algorithm

FPGA – Field Programmable Gate Array

MIMO – Multiple Input Multiple Output

TCM – Trellis Coded Modulation

STTC – Space Time Trellis Code

Table of contents

Table of figures

Introduction

Now a days communication systems use signal processing for the improvement in their performances. Two important factors used by signal processing for performance improvement are

a. Channel equalization
b. Error correction coding

MLE (maximum likelihood estimation) is considered to be the most effiecient technique for equalization. Where as for error correction, convolutional coding with viterbi decoding is a commonly used method.

1.1 Introduction to Viterbi Algorithm

The viterbi algorithm was proposed in 1967 as a method of decoding convolutional codes [1]."Viterbi Algorithm (VA) decoders are currently used in about one billion cellphones. This is probably the largest number in any application. However, the largest current consumer of VA processor cycles is probably digital video broadcasting. A recent estimate at Qualcomm is that approximately 10^{15} bits per second are now being decoded by the VA in digital TV sets around the world, every second of every day" [2].

The viterbi algorithm is an efficient way of performing maximum likelihood decoding by reducing its complexity. Also viterbi algorithm is known as optimum algorithm since it reduces the probability of errors. It eliminates the least likely trellis path at each stage which leads to reduce the decoding complexity. Also by the early rejection of unlike paths decoding complexities are reduced in this algorithm. The efficiency of viterbi algorithm is based upon the paths of the trellis diagram.

1.2 Introduction to Viterbi decoder

Viterbi decoders are actually implementations of the viterbi algorithm, which is used for decoding convolutional codes. Viterbi algorithm is used by the viterbi decoder for decoding the information that is encoded by a convolutional encoder. Viterbi decoders are widely used as forward error correction devices in most digital communication systems e.g cellular phones, radio communications, satellite communication, automatic speech recognition etc.

1.3 Role of viterbi decoder in Digital communication systems

In a digital communication systems when data is transmitted from source to destination it is therefore susceptible to noise. To combat this problem forward error correction techniques are used. Convolutional coding with viterbi decoding is a forward error correction technique, which is used for error correction.
The block diagram which illustrates the use of viterbi decoders in digital communication systems is given below.

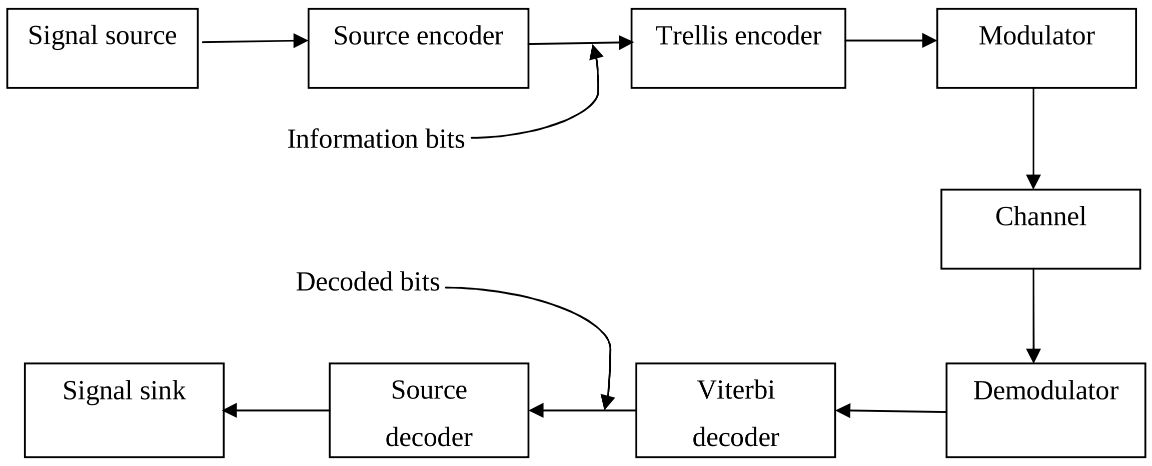


Figure 1.1 Viterbi decoder in digital communication

The signal emitted from the source is encoded by the source encoder in the initial step. The compressed information is then given to trellis encoder for the channel coding process. The redundant bits are removed by the source encoder to reduce the transmission rate. Then the redundancies from the channel encoder are introduced to protect the signal from transmission impairments. The encoded symbols from the channel then enters in to the modulator which modulates the signal according to the desired modulation scheme. After that the signal is transmitted to the channel and demodulated first. Then the resulting information from the demodulator is given to the viterbi decoder. Viterbi decoder performs the correction and the corrected information bits are then given to source decoder.

1.4 Previous research work

Different improved versions of the conventional viterbi decoder have been presented in the literature [3, 4]. These versions provide enhanced performance over conventional viterbi algorithm in different aspects for a number of applications. The scheme in version [3] generalizes the viterbi algorithm to find N globally best estimates of the transmitted sequence. This algorithm is implemented on fixed point DSP. The scheme in version [4] modifies the viterbi algorithm in a way that, a reliable value for each bit in MLE is obtained. This algorithm is used for viterbi equalization on IS – 54 radio channels.

In 2002 M. Kivoja and et.al [8] presented a efficient version of viterbi decoder, redundancies were reduced along with the memory requirements. This efficient decoder can’t detect the errors and also with two bit continuous burst of errors whole decoding algorithm fail.

In 2008 M. Kamuf, V. Owall and J. B. Anderson presennted their work on optimization and implementation of a viterbi decoder under flexibility constraints [5]. The work focussed on the impact of flexibility while designing a viterbi decoder for convolutional and TCM (Trellis Coded Modulation) codes. Optimized hardwae goal is achieved by sacrificing the speed of the trellis unit. In 2012 P. Uma Devi and P. Seshagiri Rao presented a low comlexity algorithm, computational burden is reduced for MIMO (Multiple input Multiple output) configurations [6]. The work focussed on the STTC (Space Time Trellis Code), which is widely applied to coded MIMO (Multiple input Multiple output) systems.

Implementation of viterbi in hardware with minimum architecture and low complexity is a vital task in 2012 M. Jabeen and S. Khan presented a reconfigurable viterbi decoder [7]. Convolutional encoder along with viterbi decoder is implemented in FPGA (field Programmable Gate Arrays) with a goal of reduction in complexity.

1.5 Problem statement

Conventional viterbi decoder during the path selection, in case of matrices of two paths having the equal value, one path is chosen for elimination using arbitrary rule. Because of using matrices for path elimination it struggles, if the initial bits are distorted by noise, which further can cause false decoding because of early decision to choose between paths. But converges and self corrected itself as path length increases.

It leads to the increase in complexity as trellis length increases, which cause processing delays. Also more values are added into the matrices to simplify the decision.

1.6 Objective

Viterbi decoder is used in a large number of applications. Many researchers have expanded the work of viterbi by working on specific areas according to the application need. Mostly work done focussed on the modification in architecture, optimize implementation in hardare and exploration of speed performance. All these different solutions presented are helpful in specific areas according to the application need. Implementation of viterbi decoder on FPGAs for performance improvement is very demanding task .

Target of the proposed research work is to design & implement an optimized solution for a widely used forward error correction algorithm known as Convolutional Encoder with Viterbi Decoder Algorithm. After implementing the Viterbi Decoder following parameters will be improved in the new design

A. Decoding complexity
B. Decoding time
C. Device utilization

A new solution resulting better performane will be presented along with its comparison with conventional viterbi decoder and some other previous research works done in year 2012.

The platforms used in our project are Verilog(for hardware designing), Xilinx ISE ® (for simulations and synthesis) and the target FPGA device is SPARTAN III from S200 Series and with a speed grade of 4.

1.7 Outline

The thesis presents the designs & implements an optimized solution for a widely used forward error correction algorithm known as Convolutional Encoder with Viterbi Decoder. Later on in enhancement process another solution based only on Finite State Machine has been successfully implemented that performs significantly better than the viterbi decoder in terms of error correction capability for some tackled cases.

Detailed introduction of the thesis work is explained in **chapter 1**.
Chapter 2. describes the detailed theoratical background which involves the brief description of convolutional codes, tree diagram and working of viterbi etc.
Chapter 3. discusses the architecture design and implementation of the viterbi and the enhanced solution presented. It also involves the pseudo code and results drawn from synthesis reports and major differences between the both solutions.
Chapter 4. is the final chapter it deals with the conclusion drawn from the thesis work. It involves the testing and limitation sections as well as the future work that is possible to take the project further.

2 Theory

In this chapter the literature study for the project is discussed. The chapter starts with the basic communication theory, later on the working of viterbi algorith, tree diagram and viterbi decoder trellises are described.

2.1 Communication systems

The emerging world of telecommunication needs growth in reliability and speed in communication. In a communication systems the reliability in transmission and information storage is provided by different coding techniques. The communication media carrying the information bits is susceptible to errors because of the noise, which is present in the analog portion of the channel. Therefore the detection and correction of errors is needed during the decoding process. The task of encoding to detect the errors is performed by channel coding. Channel coding is a process in which we add some redundant bits to our data which is to be transmitted through the channel is called channel coding. It is the job of channel coding to make the transmitted signal insensitive to errors introduced by the channel. This is achieved by carefully adding redundancy to the information that is to be transmitted.
Channel coding is used in many everyday devices, such as Compact disc players, computer modems, computer memories and mobile telephones.

There are two categories of coding methods:

a. Backward error correction codes
b. Forward error correction codes

2.2 BEC (backward error correction)

Backward error correction codes are capable of the single task which is error correction. It is commonly used in the cases when the transmitted data is lost or corrupted, a request is made by the reciever to retransmit the data. BEC is also known as automatic repeat request (ARP). BEC is bandwidth efficient and mostly used in the systems with small data and minimum number of errors.

2.3 FEC (forward error correction)

FEC is used for the improvement in the channel capacity. In this process we add some designed information to our data which is to be transmitted through the channel. For the improvement of channel different techniques are used one of them is channel coding. The performance of FEC is multi sender and multi path scenario depends heavily on the correlation of packet loss between paths [9].

2.4 Convolution codes

Convolution codes were invented in 1955 by P.ELIAS [10]. Convolution codes are generally error correcting codes, that are used to improve the performance of many digital systems e.g to improve the performance of digital radio, mobile phones and the Bluetooth implementations. Viterbi decoder together with its improved versions is one of the best application of convolutional codes.

When a signal is transmitted over the channel, it is affected by any of three channel impairments i.e. noise, interference, fading. To reduce the number of bit errors in the information is accomplished by introducing redundant bits into the transmitted information stream. These bits will allow detection and correction of received data, and provide reliable transmission of information.

Convolution codes are also used in real time error correction to improve the performance of digital radio, mobile phones, satellite links etc. The simplicity and performance of convolution codes for Gaussian channel is very close to the accurate. They are one of the most widely used channel codes in the practical communication systems.

Convolution codes convert the entire data stream into one single codeword. The encoded bits depend not only on the current input bits but also on past input bits as well. Convolution code structure is easy to draw from its parameters. Convolution encoding and decoding is implemented in different manners also convolution codes are applied in applications that require good performance with low implementation cost.

There are three other methods to describe a convolutional code

1. Tree diagram
2. Trellis diagram
3. State diagram

2.5 Tree diagram of convolutional encoder

A *tree diagram* also makes it possible to represent the operation of a convolution encoder. This diagram is represented below in Figure for the convolution encoder from example discussed above making the assumption that its initial state S0 is equal to a = (00). Two branches associated to the presence of the symbol “1” (respectively “0”) at the encoder input leave from each state Sk of the tree diagram, just as for the trellis diagram. This diagram is also used to decode convolutional codes, in particular, when the encoder has a large memory m (typically more than 10).

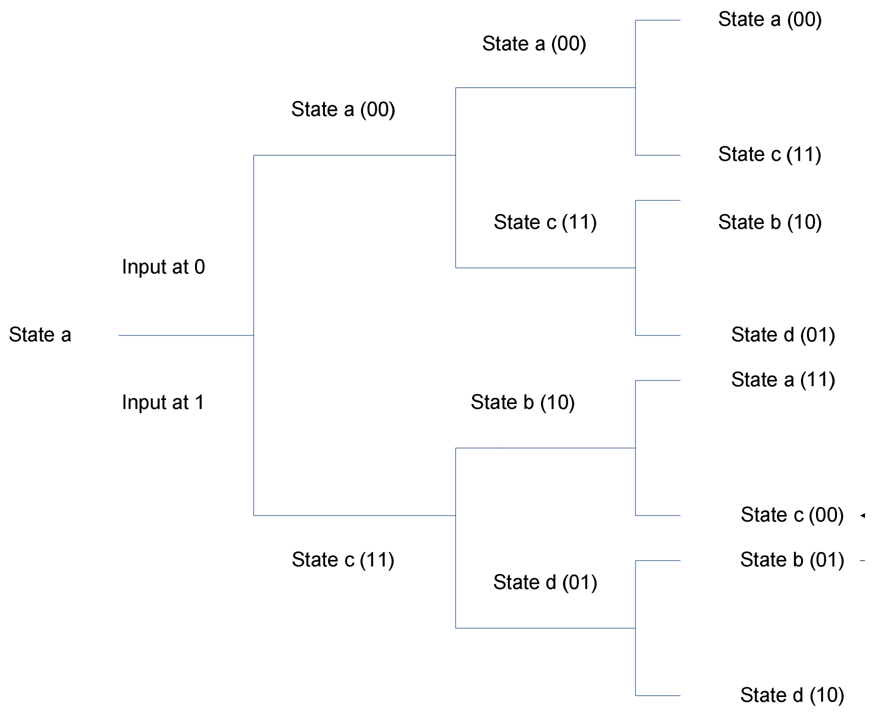


Figure 2.1 Tree diagram for convolutional encoder

The architecture of VITERBI and its description is discussed in detail in the next chapter. Also the architecture and detail description of new designed decoder which works better than the VITERBI in certain aspects is the part of further chapters.

2.6 Trellis diagram of convolutional encoder

The trellis diagram of the convolutional encoder gives us the information that “how each possible input to the encoder influences both the output and the state transitions of the encoder”

2.6.1 Maximum likelihood decoding

Maximum likelihood decoding means finding the code branch in the code trellis that was most likely to transmit. Therefore maximum likelihood decoding is based on calculating the hamming distances for each branch forming encode word. The most likely path through the trellis will maximize this metric. Also the trellis module increases exponentially due to (k) and (m) factor which leads to the complexity when we go for high rate codes, to overcome this problem punctured convolutional codes (PCC) were introduced by cain et al [11].

Example:

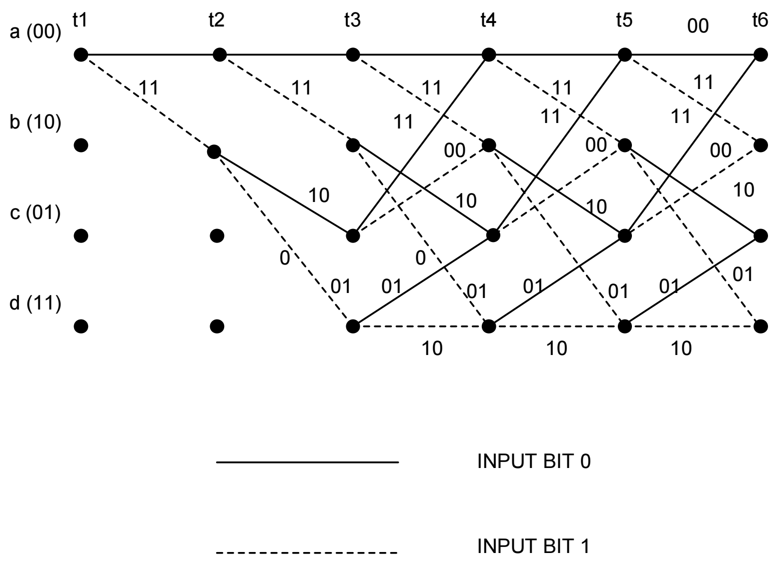


Figure 2.2 Trellis diagram of convolutional encoder

The trellis diagram of the convolutional code for above state diagram is plotted above with the assumption that S0(k = 0) is a = (00) generally referred to as the "all at zero" state. A binary pair whose first symbol is ck,1 and the second is ck,2 is associated to each branch of the diagram.

Working Explanation

From each trellis node, corresponding to a value of the state Sk and noted by a point in Figure, leave two branches associated to the presence of a "1" symbol (dotted) and of a "0" symbol (solid) at the encoder input respectively.

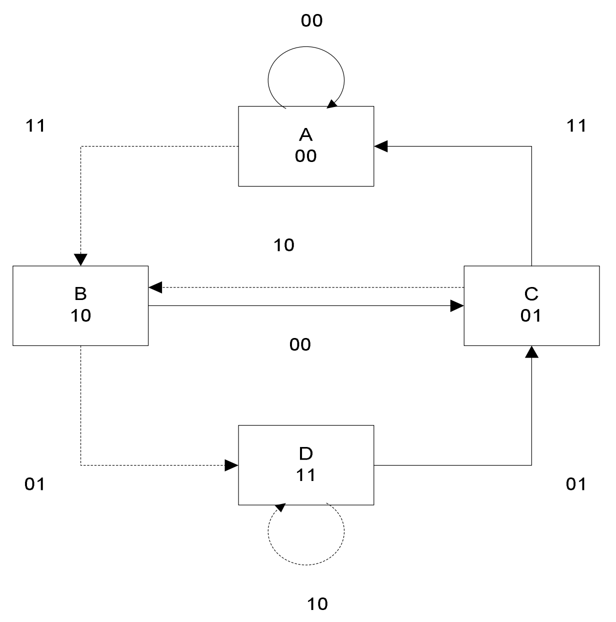
The succession of branches constitutes a path in the trellis diagram; each path is associated to particular sequence transmitted by the encoder (here, sequence indicates the *sequence* of coded symbols transmitted by the encoder, placed in a series).

Examining Figure we can notice that the encoder produces particular sequences known by the decoder. If the sequence received by the decoder does not correspond to a path of the trellis diagram, the decoder will be able to detect the presence of transmission error(s). Moreover, as we will see later on, it will be able to correct the error(s) by finding in the trellis diagram the sequence "nearest" to the received sequence.

The complexity of the trellis diagram depends on the number of states of the encoder and the number of branches that leave (or merge towards) each node (2K). Its complexity thus grows exponentially with the memory of the encoder but also with the length K of the coded blocks. For packet transmissions, the state of the encoder is generally initialized at zero at the beginning of each coded block. That can be achieved by adding m K zero symbols at the end of each coded block. All the paths of the trellis diagram then merge towards the zero state. These m K symbols are called *tail-biting symbols* of the trellis diagram. The convolution encoder then associates a unique coded sequence with each information sequence. Convolutional codes are in this case perfectly similar to block codes and the coded sequences are code words. The trellis diagram is mainly used for the decoding of convolutional codes.

2.7 State transition diagram of convolutional encoder

The operation of a convolution encoder is represented by a graph called a *state transition diagram*. Convolutinal codes are also represented by the state diagram. Following below is the state transition diagram of a convolutional code which describes the transition between the states.



Input bit 0 straight lines

Input bit 1 dotted line

Figure 2.3 State transition diagram for convolutional encoder

The state of the encoder at the moment k noted Sk = (dk, dk−1) can take four values regardless of k. We will note them here: a = (00) b = (01) c = (10) d = (11). Two transitions are possible for each state according to the value of the symbol presented at the encoder input. Binary pairs carried by the transitions between states, or buckling on the same state, correspond to the blocks transmitted by the encoder at every moment k. The transitions in solid (respectively dotted) lines correspond to the presence of one “1” (respectively “0”) at the encoder input.

2.8 Decoding convolutional codes

First convolution encoding is applied to the channel in which the transmitted signal is corrupted with AWGN (additive white Gaussian noise). Then decoding is performed by the algorithms listed below: Two algorithms which are most commonly used for decoding convolutional codes:

a. Viterbi decoding
b. Sequential decoding.

The first method proposed for decoding convolutional codes is sequential decoding. Viterbi decoders are most commonly used for relatively small values of (k) constraint length .Similarly for the long constraint length (k) values sequential decoding is used. Viterbi decoding is a very suitable option for hardware implementation because of the fixed decoding time. Viterbi decoding is a common technique used for decoding when convolution encoding is applied.because of optimum performance and its easy implementation.

2.9 Viterbi algorithm a common digital signal processing function

Viterbi algorithm is one of the most common used digital signal processing function, which is used to design a decoder known as Viterbi decoder. Viterbi algorithm performs decoding of a convolutional codes as well as the maximum like-lihood estimation of the transmitted sequence. Viterbi maximum like-lihood algorithm is considered to be one of the best technique for decoding convolutional codes. Implementation of viterbi decoder in DSP provides reasonable operational flexibility.

The two DSPs (56300 and 56600) are specifically designed for communication functions. Because of Viterbi decoder prominent position in communication systems these processors have Viteri Shift Left instruction .

2.10 Viterbi decoder

The Viterbi decoder consists of three main units branch matrices unit, add control select unit and survivor management unit. Figure shown below shows the general structure of viterbi decoder.

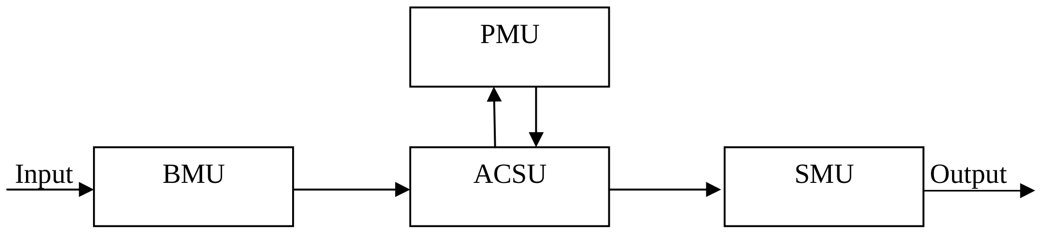


Figure 2.4 Block diagram of viterbi decoder

The branch matric unit (BMU) is responsible for the computation of matrices, second block add compare select unit (ACSU) selects the survivor paths for each trellis state, third block survivor management unit (SMU) performs the selection of output which is based on the most optimum path metric.

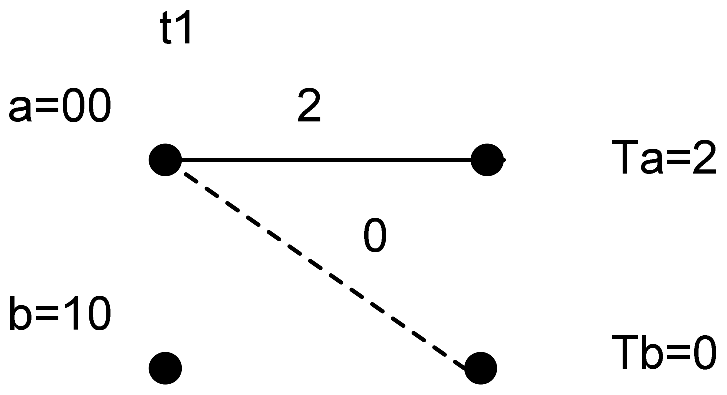
2.11 Viterbi Algorithm working

In digital communication viterbi algorithm is the maximum likelihood sequence detector (MLSD) [12]. Also is a efficient way of performing maximum likelihood decoding by reducing its complexity. It eliminates the least likely trellis path at each stage which leads to reduce the decoding complexity.

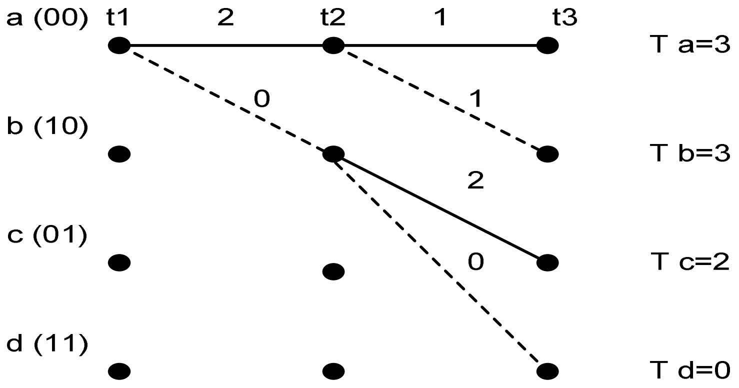
Also by the early rejection of unlike paths decoding complexities are reduced in this algorithm. The efficiency of VITERBI algorithm is based upon the paths of the trellis diagram. There are four major steps involved in viterbi decoding algorithm.

1. Calculate the trellis, which means obtaining the weight of the trellis branches by calculating branch matrices.
2. Obtaining the last state with the minimum path.
3. Traceback which means calculating the entire weight path.
4. Reordering the bits into correct format.

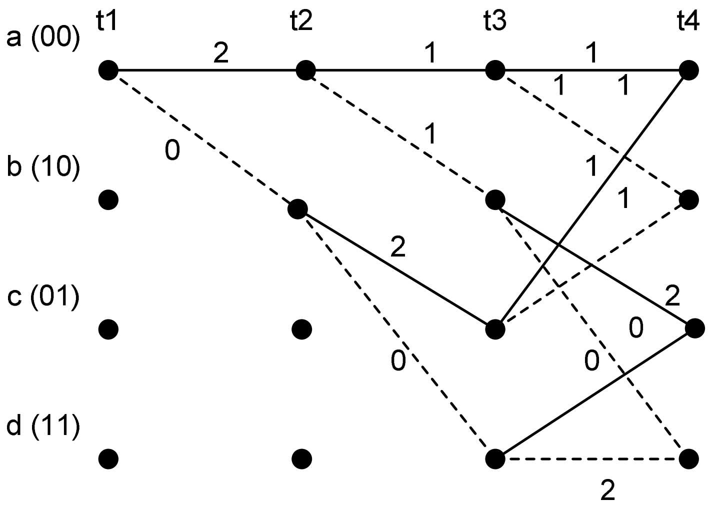
Following below are the viterbi decoder trellis diagrams. It follows the steps stated above and help to understand how viterbi decoder algorithm works.


t1
a=00
2
Ta=2
0
b=10
Tb=0

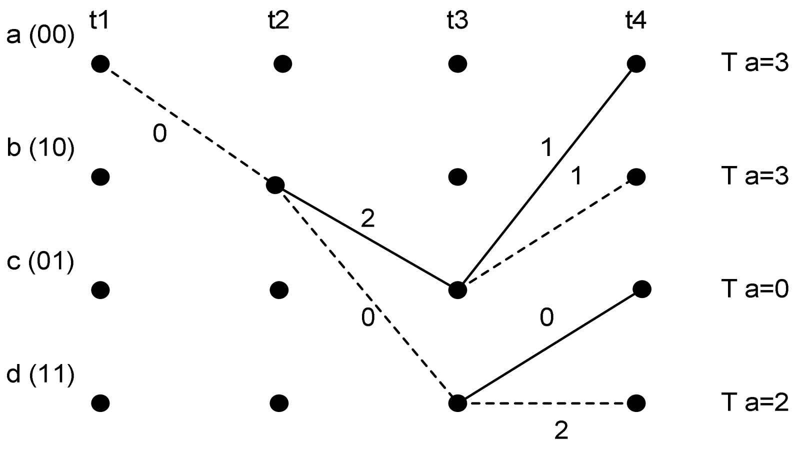
(a)


a (00)
t1
2
t2
1
t3
T a=3
0
1
b (10)
T b=3
2
c (01)
0
T c=2
d (11)
T d=0

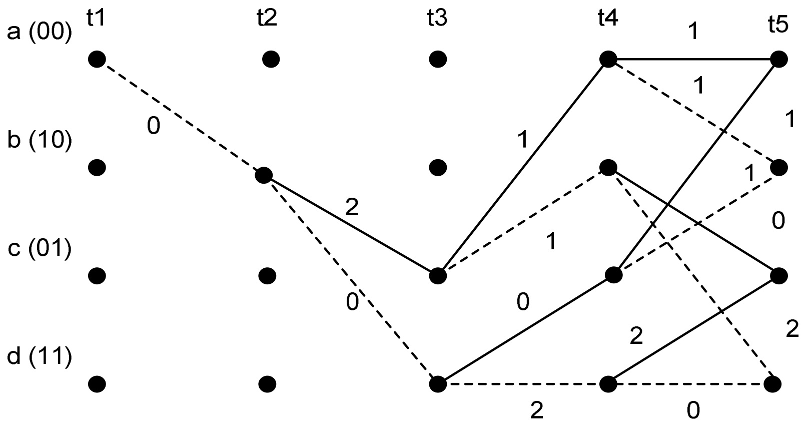
(b)


a (00)
t1
t2
t3
t4
2
1
1
1
1
0
1
b (10)
1
1
2
c (01)
2
0
0
0
d (11)
2

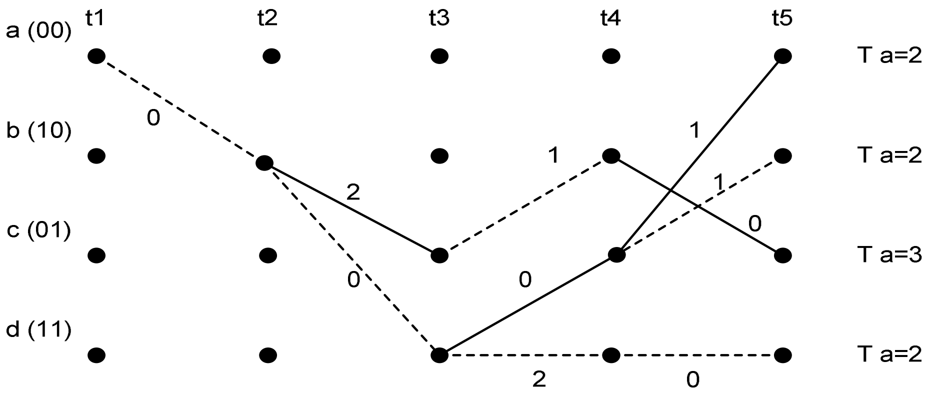
(c)


a (00)
b (10)
c (01)
d (11)
t1
t2
t3
t4
T a=3
T a=3
T a=0
T a=2
0
2
0
1
1
0
2

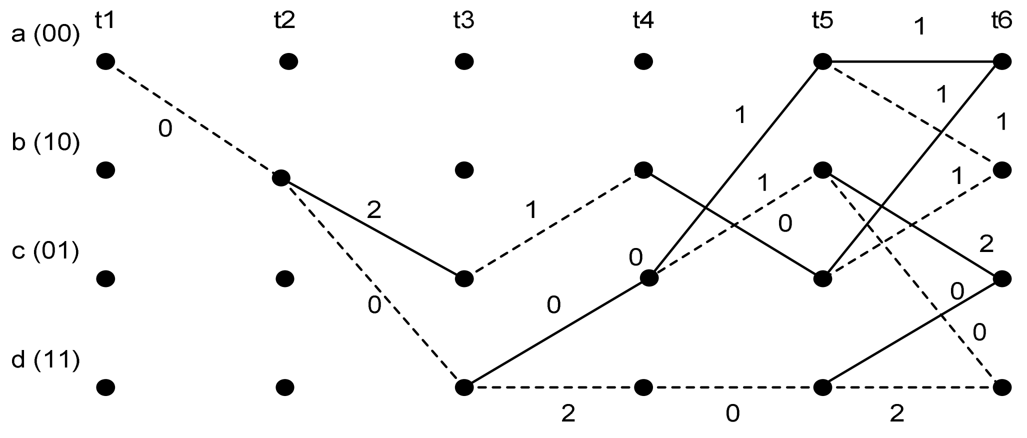
(d)


a (00)
b (10)
c (01)
d (11)
t1
t2
t3
t4
t5
0
2
0
1
1
0
2
1
1
1
1
0
2
2
0

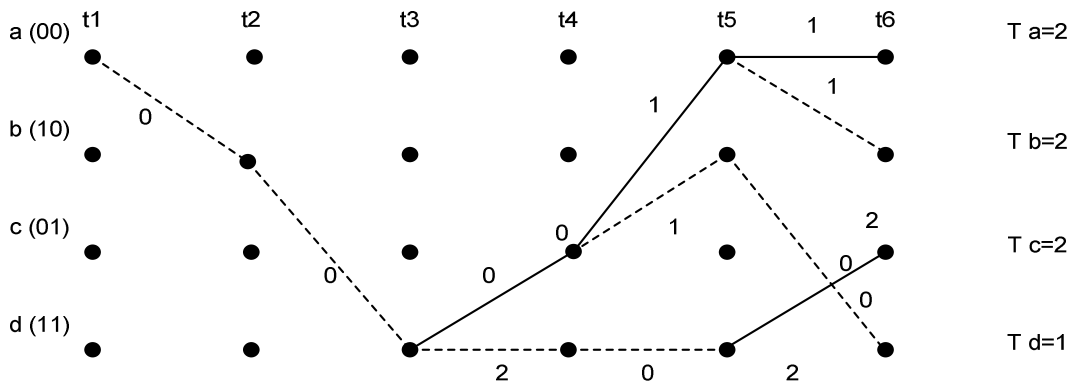
(e)


a (00)
b (10)
c (01)
d (11)
t1
t2
t3
t4
t5
T a=2
T a=2
T a=3
T a=2
0
2
0
1
0
2
1
1
0
0

(f)



(g)



(h)

Figure 2.5 Trellis schemes for Viterbi decoder

In the above ***figure a*** the state 00 →00 transition has branch metric 2: state 00→10 transition has branch metric 0. At time t2 there are two possible branches leaving each state as shown in ***fig b***. the cumulative metrics of these branches are labeled state metrics Ta Tb Tc Td corresponding to the terminating state. At time t3 in ***fig c*** there are again two branches diverging from each state which results in two paths entering each state at time t4. One path entering each state can be eliminated,which is the one having the larger cumulative path metric. The surviving path into each state is shown in ***fig d***. at this point in the decoding process there is only a single surviving path termed the common stem, between time t1 and t2.

Therefore the decoder can now decide that the state transition which occurred between t1 and t2 was 00→10. Since the transition is produced by an input bit one, the decoder outputs a one as the first decoded bit. The ***fig e*** shows the next step in the decoding process again at time t5 there are two paths entering each state and one of each pair can be eliminated. ***Figure f*** shows the survivors at time t5. In ***fig g*** again the pattern of remerging paths. ***Fig h*** shows the survivors at time t6.

2.12 FPGAs (Field Programmable Gate Arrays)

FPGAs are pre fabricated silicon devices that can be programmed to any digital system according to the user requirement. Due to design flexibility and minimum device utilization FPGA is a very suitable option for implementing many signal processing tasks. Verilog is most commonly used design entry Language for FPGA. The programmable logic components in FPGA are called logic blocks. The complex and combinational functions are performed in these logic blocks. Memory elements which may be simple flip flops or complete blocks of memory are also included in logic blocks. FPGAs are better in flexibility, performance and area utilization as compared to DSPs.

2.13 Why FPGAs?

There are many different approaches for implementing convolutional encoder and viterbi decoder. Viterbi can be implemented using ASIC and DSP too. As an ASIC viterbi decoder implementation is very efficient in power but its fixed desing restricts for operational flexibility. This problem is efficiently solved while implementing in DSP but over a cost of loss in performance and efficiency. Both ASIC and DSP has fixed constraint length and code rate. To solve these issues FPGA implementation has been proposed. FPGAs are very much better in performance, flexibility and area utilization as compared to the both ASIC and DSP.

2.14 Important factors

Factors to be focussed carefully while designing an efficient error controll coding scheme are discussed below:

2.14.1 Error detection capability

The error detection capability of a code describes how well it can detect errors introduced by the channel. In some communications schemes, errors are not corrected, but upon detection of an error, a retransmission is requested. Such schemes are known as backward-error-correction (BEC).

2.14.2 Error correction capability

The error correction capability of a code describes how well it can recover the transmitted message without error, after errors are introduced by the channel. There are some schemes which can correct errors without sending any request for retransmitting. Such schemes are known as ***forward-error-correction (FEC).***

2.14.3 Encoding complexity

In situation where the transmitter must be very cheap, the complexity of the encoder is important. This may be measured in how many transistors or gates are required for implementation, or how much computer memory is needed.

2.14.4 Decoding complexity

Decoding complexity is one of the main issues in error control coding. If we pick a code at random, then probability of getting efficient and good error correction capabilities is very high. If decoding a random code is extremely complex, then we have to trade off decoding with error correction rate.

2.15 Xilinx/modelsim (software used)

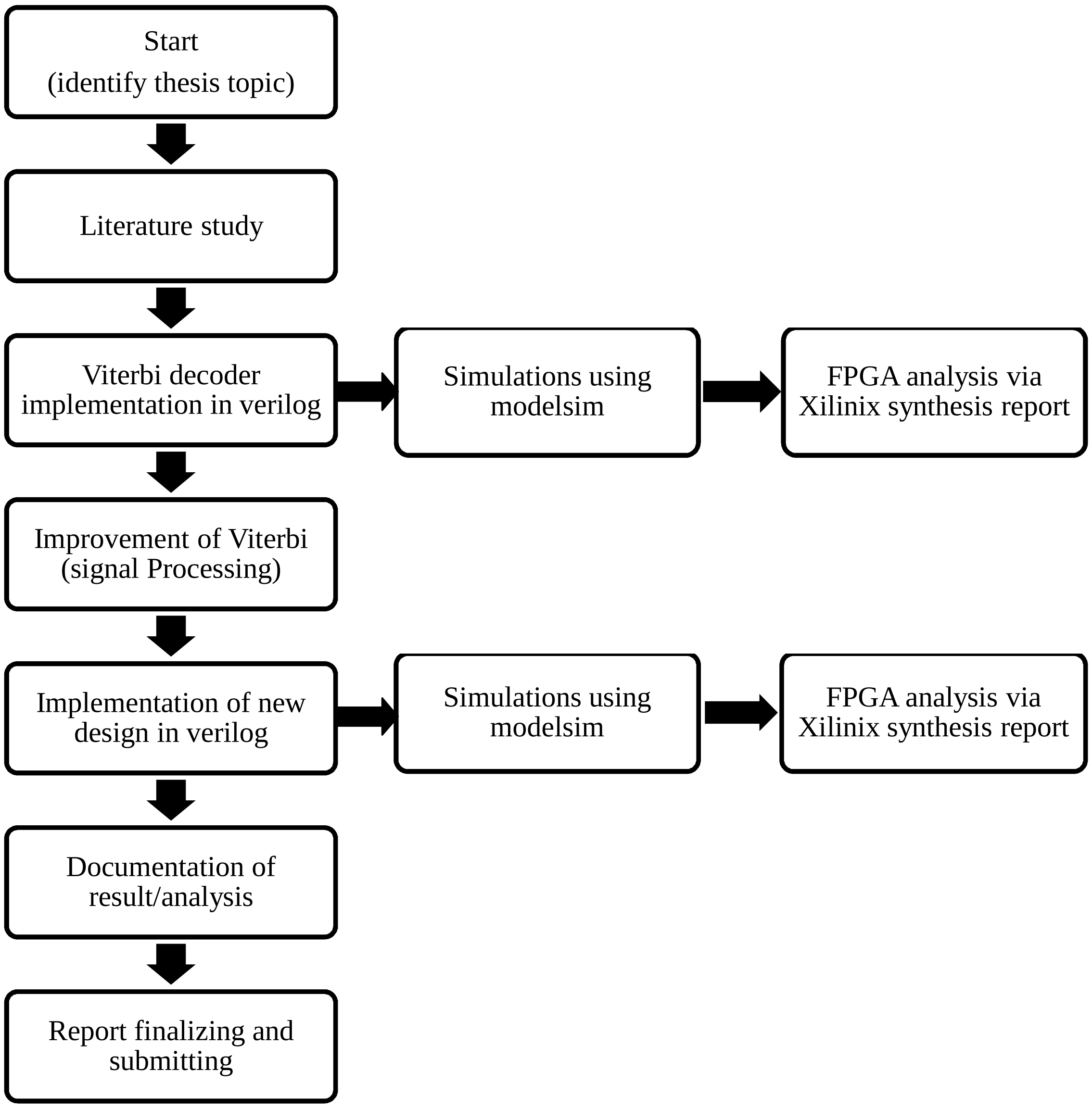
The software used for the synthesis and analysis is xilinx. This is an easy way to compile the designs and perform timing analysis according to the targeted FPGA device of our design compatibility.
Thes design is simulated using modelsim and then synthesized using Xilinx. Because of this design time can be reduced by allowing a high level design specifications instead of specifying the design by the FPGA base elements.

3 System design and implementation

This chapter includes the architecture diagrams of the both decoders along with their complete description and the synthesis report. Pseudo code of enhamced design is also presented in the following chapter.

3.1 Flow chart

The working scheme of the thesis can be easily analyzed in the flow chart given below.


Start
(identify thesis topic)
Literature study
Viterbi decoder implementation in verilog
Simulations using modelsim
FPGA analysis via Xilinix synthesis report
Improvement of Viterbi (signal Processing)
Implementation of new design in verilog
Simulations using modelsim
FPGA analysis via Xilinix synthesis report
Documentation of result/analysis
Report finalizing and submitting

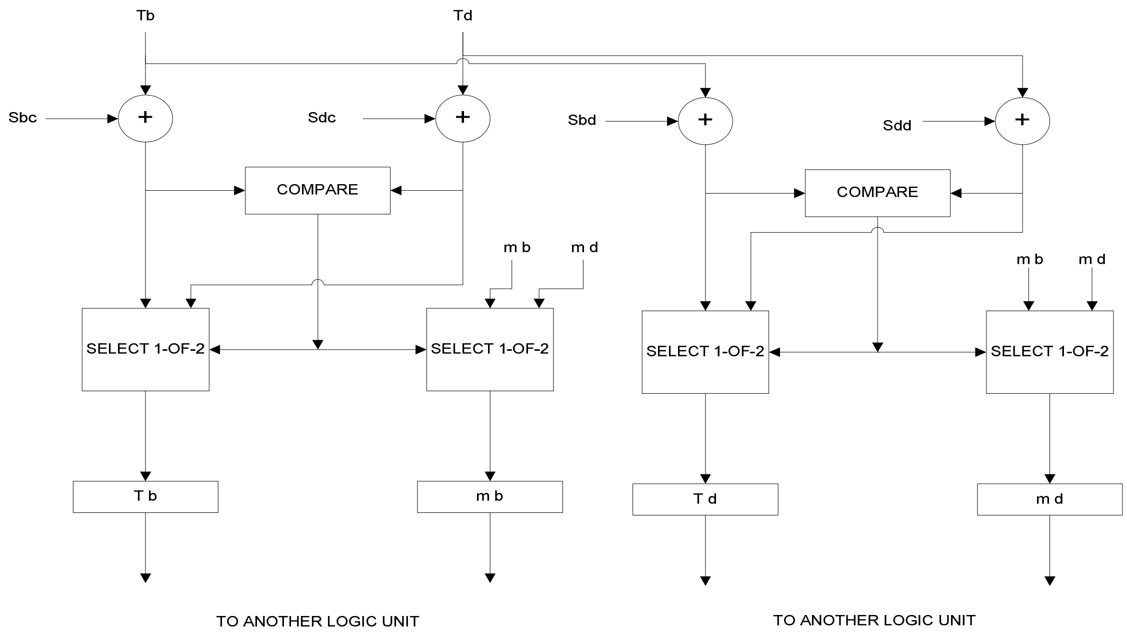
We divided the thesis work into three major phases the initial phase involved the thesis identification and area of interest for work. In the second phase the implementation work is performed along with the architecture modification and literature study. In the final phase report finalizing and writing is done along with final results and analysis.

3.2 Implementation

The implementation of viterbi decoder and later on the improved version is done using verilog HDL. The design is simulated and synthesized in Xilinx. The synthesis reports of both decoders are presented, which shows the device utilization and timing constraints.

3.3 Viterbi decoder architecture

The viterbi decoder architecture is presented below with all the units that are involved in the implementation. As it can be seen that there are two main cells present in the diagram, both of them are very much identical so by explaining the working of one cell will describe the working of both of the cells in first cell as it can be seen that state transition metric Ta of next state is being calculated by adding the branch metric Saa with state transition metric Ta of previous state and adding Tc with Sca. The result of both these additions e.g (T1 = Ta + Saa and T2 = Tc + Sca where T1 nd T2 are results of additions) are compared and the path metric with smallest distance is assigned to next state transition metric Ta, and also the corresponding information bits of the path is stored in ma for state a. Where ma is the message path history of the winning path.



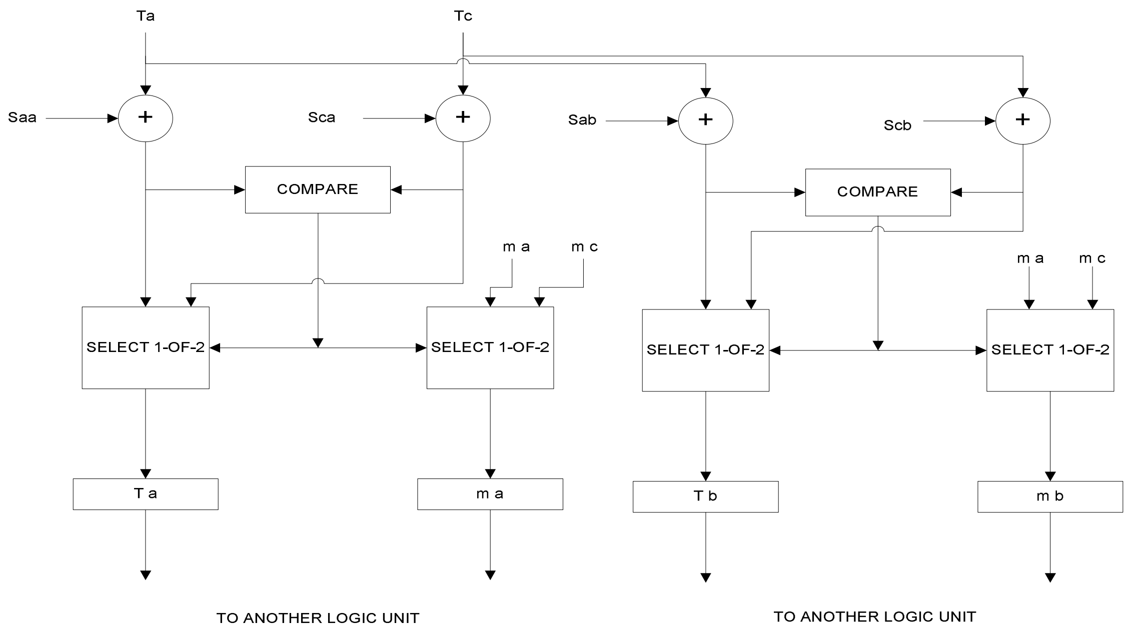


Figure 3.1 Viterbi decoder architecture

As it is shown from the figure that same operation will be performed to get Tb and mb, also similar operation will be performed for second cell in the diagram.

3.4 Synthesis results of viterbi decoder

Following below is the synthesis report generated in xilinx after performing the simulations using the modelsim. It invloves the information about the device used and the factors upon which we are able to make analysis.

Device utilization summary:

Selected Device : 3s700anfgg484-4

Number of Slices:	99 out of 5888	1%
Number of Slice Flip Flops:	64 out of 11776	0%
Number of 4 input LUTs:	188 out of 11776	1%
Number of IOs:	12	
Number of bonded IOBs:	12 out of 372	3%
Number of GCLKs:	1 out of 24	4%

Partition Resource Summary:

No Partitions were found in this design.

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.

Timing Summary:
Speed Grade: -4

Minimum period: 10.720ns (Maximum Frequency: 93.284MHz)
Minimum input arrival time before clock: 13.140ns
Maximum output required time after clock: 5.558ns
Maximum combinational path delay: 7.812ns

3.5 Architecture of the enhanced decoder

The alchitecture of the enhanced decoder is presented below. It can be analyzed easily by focussing the architecures of both decoders that the new presented one is more optimized than the viterbi.

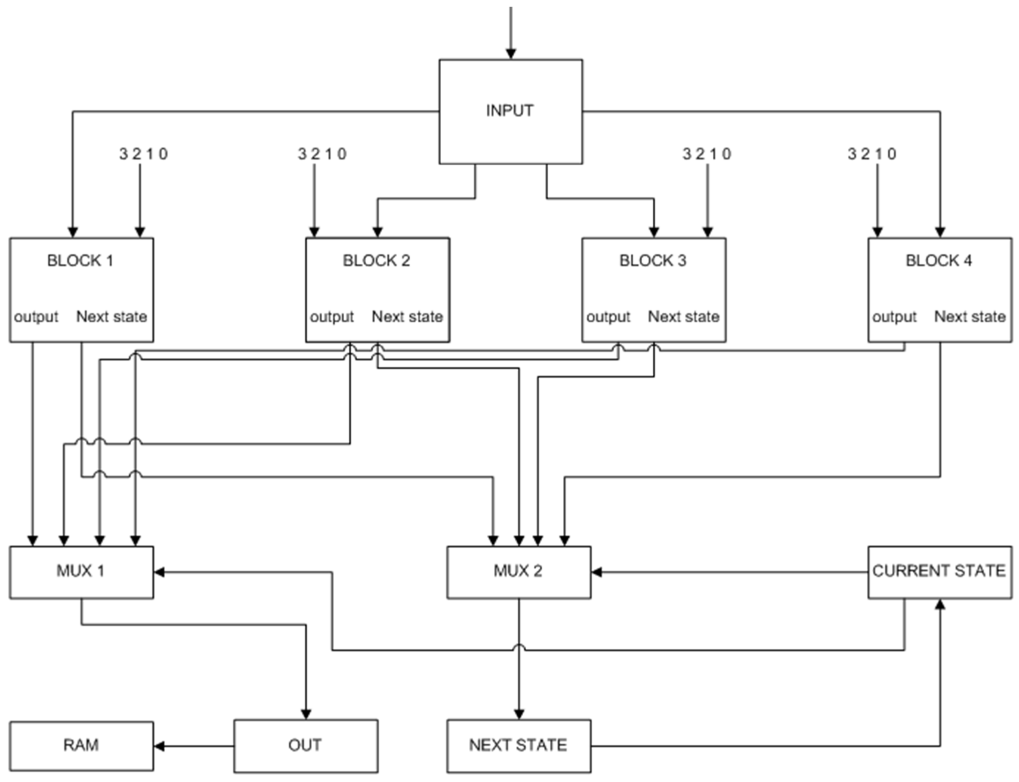


Figure 3.2 New decoder architecture

As input is of two bits but is pipelined in four bit register “INPUT”, and on 2nd and 3rd bit of input register. Which is in fact an input which is buffered in a reg “INPUT” the next state and the output is decided.

If an error occurs e.g if current state is state0 then input “0” or ”3” is expected but it becomes “2” or “1” because of an error, then this unexpected output is compare with first and zeroth bits of register “INPUT” which in fact is a next incoming input this will decide the next state and once next state is decided then expected output can be decided easily because outputs of all states are known after this the corresponding state is assigned to next state and output is assigned to the output register. Using Mux 0 and Mux1 current state will decide the final outputs which will be stored in register out and next state then output is stored in ram and current is assigned a next state.
The circuitry of the each block shown in above design is also presented below. From which we can get the basic idea of the logic involved in the design.

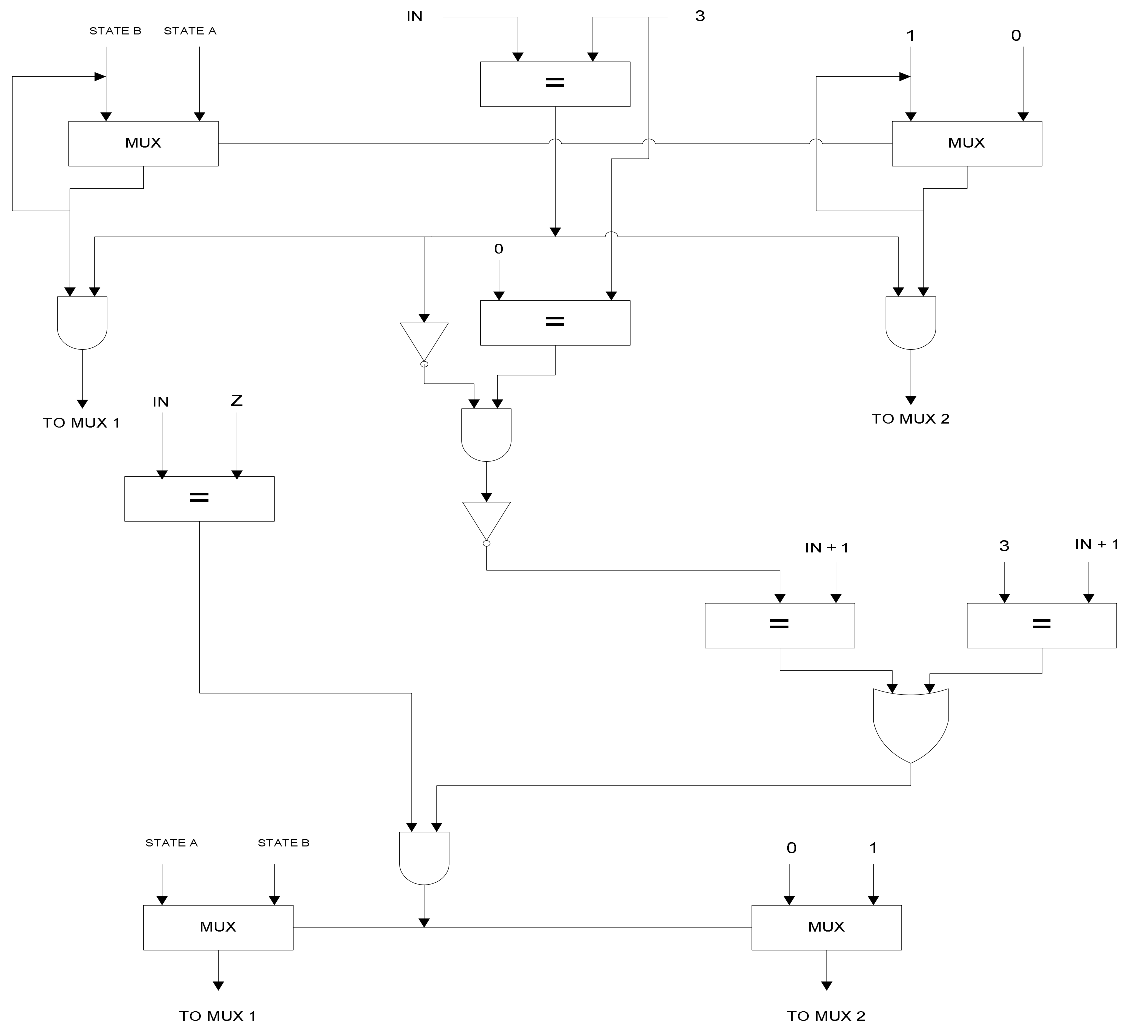


Figure 3.3 Internal circuitry of blocks shown in figure 3.2

3.6 Synthesis results for the enhanced algorithm

Following below is the synthesis report of the enhanced design generated in xilinx. It involves all the major factors like speed, timing and number of units involved upon which we are able to make final analysis

Device utilization summary:

Selected Device : 3s700anfgg484-4

Number of Slices:	17 out of 5888	0%
Number of Slice Flip Flops:	18 out of 11776	0%
Number of 4 input LUTs:	25 out of 11776	0%
Number of IOs:	12	
Number of bonded IOBs:	12 out of 372	3%
Number of GCLKs:	2 out of 24	8%

Partition Resource Summary:
No Partitions were found in this design.
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT GENERATED AFTER PLACE-and-ROUTE.
Timing Summary:

Speed Grade: -4

Minimum period: 2.191ns (Maximum Frequency: 456.413MHz)
Minimum input arrival time before clock: 5.936ns
Maximum output required time after clock: 5.779ns
Maximum combinational path delay: No path found

Timing Detail:
All values displayed in nanoseconds (ns)

The detailed comparison of both decoders on the basis of synthesis reports is presented in the next ssections.

3.7 Previous work presented (comparison)

In september 2012 M. Jabeen and S. Khan [7] presented their work on reconfiguration of viterbi decoder. Following results were presented after synthesizing the new reconfigurable Viterbi decoder.

Device Utilization Summary		
Logic utilization	**Used out of available**	**Utilization**
Number of Slices	86 out of 7681	1%
Number of Slice Flip Flops	17 out of 1536	1%
Number of 4 input LUTs	142 out of 1536	9%
Number of bonded IOBs	14 out of 63	22%
Number of GCLKs	1 out of 8	12%

The results obtained the enhanced decoder in the proposed work are:

Device Utilization Summary		
Logic utilization	**Used out of available**	**Utilization**
Number of Slices	17 out of 5888	0%
Number of Slice Flip Flops	18 out of 11776	0%
Number of 4 input LUTs	25 out of 11776	0%
Number of bonded IOBs	12 out of 372	3%
Number of GCLKs	2 out of 24	8%

By analyzing the synthesis report generated by Xilinx it can be seen that the number of slice flip flops is less than 1% in the proposed work and its 1% in the work presented before in 2012. It means that new design prposed is saving silicon area by effiecient device utilization. The new proposed solution is very much favorable to now adays FPGAs because of its chip area utilization providing a good trade off between silicon area and performance. Also the lesser number of slices indicates that the new design is compact as compare to the design presented in the previous work [7].

3.8 Major differences between both decoders

In this section the major differences in the design and working of both decoders are discussed in detail. Later on the differences in the technical aspects are discussed.

3.8.1 Viterbi decoder

The basis of Viterbi algorithm is the following observations: if any two paths in the trellis merge to a single state, one of them can always be eliminated in search for an optimum path [13]. It means Viterbi decoder in case of matrices of two paths having same value; it chooses one path to eliminate following the arbitrary rule. Because of using matrices for path elimination it struggles if the initial bits are distorted by noise, which further can cause false decoding because of early decision to choose between paths, but converges and self-corrected itself as path length increases and more values are added in matrices to simplify the decision.

3.8.2 New designed decoder

In our case we presented a solution first by modifying the architecture presented in figure 3.2 and figure 3.3 and later on by following a new scheme, in which decision is made on next incoming bits instead of matrices. We use the current state and the input bit sequence to select the path. If you know your current state then on coming input the next state can be decided. At this stage if next state is not as expected to be then decision is taken on the base of next buffered bits to which state it should go.
This can be analyzed in the design of enhanced decoder as well.

Also the synthesis report results show that the enhanced decoder is working efficiently, the test sequences and the technical comparison aspects on the basis of synthesis reports of both decoders are listed in upcoming sections.

3.9 Pseudo code of new design decoder

The pseudo code version of new designed decoder is as below

Input:

The external clock : clk

The reset signal : rst_n

The encoded Input data stream : In1

Output:

The decoded output :out

```
Define four states as State_0, State_1, State_2 and State_3
Function fsm_dec(clk, rst_n, In1, out)

For every positive edge of the clock or negative edge of reset signal Do
    If reset signal has a negative edge
        Initialize current state to high impedance state
        Initialize next state to State_0
        Initialize Output register to high impedance state
        Initialize other registers
    Else
        Receive first 2 bit of input data to temporary register

        Assign next state to current state

End For
For any change in current state or reset signal Do
    If reset signal has negative edge
        Assign State_0 to next state

    Else
        If current state is in State_0 or State_1 or State_2 or State_3
            If the 2bit of the input encoded data stream stored in
            temporary buffer is same as expected according to the current
            state
                Decide and assign decoded output and next state
                accordingly

            Else
```

```
                If because of any error 2bit encoded data is not same as
                expected then check next 2bits of incoming encoded data
                    Decide and assign decoded output and next state
                    accordingly
End For
End Function
```

4 Testing limitations and conclusions

This final chapter includes the major differences in previous and our work. Final results are also presented in detail along with the analysis and conlusions.

4.1 Testing on different test samples

We test both decoders with different test samples and obtain the results. Following below are three different test samples which are decoded by both decoders and results obtained are also listed. It can be seen easily that in the case of **test sample 1**, when there is random distributed error , Viterbi malfunctioned and the new design gave the correct results.
In the **test sample 2** when there is burst of errors the new designed decoder is decoding accurately as compared to conventional viterbi.
In the test **sample 3** again with continuous burst of errors the new design decoder performs better by recovering more bits as compared to the viterbi.

Input Sequence: 1 1 0 1 1 0 0 0
Transmitted Sequence: 11 01 01 00 01 01 11 00

Test Sample 1 : 10 01 00 00 00 01 10 00
Errors/Byte: 2
After decoding results
Viterbi Decoded: 1 0 1 0 1 1 1 1
Fsm Decoded: 1 1 0 1 1 0 0 0

Test Sample 2: 10 01 11 00 00 01 00
Errors/Byte: 2
Viterbi Decoded: 1 0 1 0 1 1 1
Fsm Decoded: 1 1 0 1 1 0 1

Test Sample 3: 11 01 01 10 01 01 00
Errors/Byte: 1
Viterbi Decoded: 1 1 0 1 1 0 0
Fsm Decoded: 1 1 0 1 1 0 0

The results of both decoders with different test samples show that new designed decoder is working better than the viterbi, in case of distributed random errors. Also when there is continuous burst of errors new designed decoder managed to obtain better correction as compared to the viterbi. If we are having continuous burst of errors in first two bits of the sequence the new decoder is working accurate in correction, whereas viterbi fails.

4.2 Limitations

Above discussed both decoders can not recover the continuous errors i.e two consecutive errors in a sequence. Both decoders deals with random distributed errors but in lesser number of bits more errors are corrected by new designed Fsm decoder. Also in the case when both decoders are performing correction completely the new design decoder is taking less time and is better in technical aspects which are compared in the following section.

4.3 Conclusions/results

We have succesfully enhanced the Viterbi in a way that it works better in several aspects as compared to the previous one. From the study of device utilization summary section in synthesis report it can be seen that the major goal achieved is less processing time and minimum device utilization. Number of slice flip flops shows that it takes less chip area as compared the viterbi and other decoder presented in [7]. Also by following the new technique in following the trellis, we are able to reduce redundancies. The compact design of new decoder helped us with reduction in computational complexity and memory requirements. Following table shows the comparison of both decoders in terms of device utilization.

Device utilization summary:

Selected Device: 3s200pq208-4

DEVICES	**Usage of Viterbi Decoder**	**Usage of new designed decoder**	**Conclusions**
Number of Slices	67 out of 5888	17 out of 5888	Better
Number of Slice Flip Flops	54 out of 11776	18 out of 11776	Better
Number of 4 input LUTs	117 out of 11776	25 out of 11776	Better
Number of bonded IOBs	11 out of 372	12 out of 372	
Number of GCLKs	1 out of 24	2 out of 24	

In the above table the comparison is made by focussing on the basic technical aspects. Which shows that the new design takes less processing time and less device utilization as compared to the viterbi. Lesser number of slices indicates the more compact design and lesser number of slice flipflops shows that the new design of proposed work is saving the silicon area.

4.4 Suggested future work

The future work can include the improvement of viterbi algorithm by using the puncturing techniques. Improvement of viterbi algorithm used in Hidden Markov Model (HMM) activity recognition is also a pivotal task, the improvemnt in processing time and device utilization will be followed after that.

References

[1] A. J. Viterbi, "*Errors bounds for convolutional codes and an asymptotically optimum decoding algorithm,*" IEEE Trans. Inform. Theory, vol. IT-13, pp.260-269, Apr. 1976.

[2] G. David Forney, "*The Viterbi Algorithm A Personal History,*" Jr., Presented at the Viterbi Conference, University of Southern California, Los Angeles, March 8, 2005.

[3] N. Seshadri, and C-E. W Sundberg, "*Generalized Viterbi Algorithm for Error Detection With Convolutional Codes,*" in Proc. IEEE Global Telecommunication Conference 1989, Dallas, Texas, Nov 1989 pp.43.3.1-43.3.4.

[4] J. Hagenauer and P. Hoeher, "*A Viterbi Algorithm with Soft-Decision Outputs and Its Applications,*" in Proc. IEEE Global Telecommunication Conference 1989, Dallas, Texas, Nov 1989, pp.47.1.1-47.1.7.

[5] M. Kamuf, V. Öwall and J. B. Anderson, *"Optimization and Implementation of a Viterbi Decoder under Flexibility Constraints,"* IEEE Transaction on Circuits and Systems I: Regular Papers, Vol.55 No.8, September 2008.

[6] P. U. Devi and P. S. Rao, *"Viterbi Decoder with Low Power and Low Complexity for Space-Time Trellis Codes,"* International Journal of Engineering and Science, Vol.2, No.3, May-June 2012.

[7] M. Jabeen and S. Khan, *"Design of Convolutional Encoder and Reconfigurable Viterbi Decoder,"* International Journal of Engineering and Science, Vol.1, No.3, September 2012.

[8] M. Kivioja, J. Isoaho, and L. Vanska, *"Design and Implementation of Viterbi Decoder with FPGAs,"* Journal of VLSI Signal Processing 21(1):5–14, May 1999.

[9] T. Nguyen and A. Zakhor, "*Distribution Video Streaming with Forward Error Correction,"* in Pocket Video Workshop, Pittsburg, PA, USA, April 2002.

[10] P.Elias, *"Coding For Noisy Channels,"* 1955 IRE International Convention Record (Part IV), pp.37-46.

[11] J. B. Cain, J. C. Clark, Jr, and J. M. Geist, *"Punctured convolutional codes of rate (n-1)/n and simplified maximum likelihood decoding," IEEE* Trans. Inf. Theory, vol. IT-25, pp. 97-100, Jan. 1979.

[12] G. D. Forney Jr, "*Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbolic interference,*" IEEE Trans. Inform. Theory, vol. IT-18, pp.363-378, March 1972.

[13] B. Sklar, "*Digital Communications: Fundamentals and Applications,*" 2nd Edition, Prentice Hall of India, 2002, pp.402.

[14] J.G. Proakis, "*Digital communications,*" Newyork: McGraw-Hill, 1989.

[15] B.P.Lathi “*Modern Digital and Analog Communication Systems*,” 2nd Edition, Prentice Hall of India, 2002.

Appendix A

New decoder Implementation

```
module fsm(clk, rst_n, in1, out);
input   clk;
input   rst_n;
input   [1:0]in1;   // module input
output  [7:0]out; // module output
reg     [7:0]out;
reg     [1:0] current_state; // 4 possible states
reg     [1:0] next_state; // used in states transitions
reg     S0,S1; // S0=0, S1=1
reg     [1:0] in, temp; // for buffering 2 bit input
reg     [7:0] ram [0 : 255]; // not part of code (additional)
reg     [2:0]count; // to limit 8bit bit by bit access
reg     [7:0] address; // to access ram
parameter [1:0]
    STATE_0=0,
    STATE_1=1,
    STATE_2=2,
    STATE_3=3; // state declaration
//Combinational process
always @( current_state or posedge clk ) // STATE is selected on the base of current state at every
//posedge of the clk
begin
    case(current_state)   //synopsys parallel_case full_case
    STATE_0: // default state = state0 as current_state= state0 assigned on reset
        begin
            if({in[1],in[0]}==3) // comparing 2bit input to decide decoder output and    //next_state
                begin
                    next_state<=STATE_1; //  if in = 2'b11 then next  state is state1
                    out<={out,S1}; // as in encoder state diagram if transition from state0 to state1 occures
                                   // because input 1 is received at state0, in decoder case it will be an
                                   // output
               end
```

```
         else
            begin
               if({in[1],in[0]}==0) // comparing 2bit input to decide decoder output and next_state
                  begin
                     next_state<=STATE_0; // according to encoder state diagram if in = 2'b00 then next
                                         // state is state0
                     out<={out,S0}; // as in encoder state diagram if transition from state0 to state0
                                   //occurs because input 0 is received at state0, in decoder case in(0)
                                   //will be an output
                  end
               else
                  begin
                     if(({in1[1],in1[0]}==0 || {in1[1],in1[0]}==3) && in != 2'bz)
// as at state0 input 2'b11 or 2'b00 is expected but because of noise if bit combination changed then we
//will compare next 2bit input to decide the correct output and state transition
                        begin // this is done as when in = 1st input then in1 = next or second input as
                              //temp = in1, and in = temp so in1 has latest input to the module and in reg
                              //has previous input
                           next_state<=STATE_0; // in case of error possible next state
                           out<={out,S0}; // in case of error possible output

                        end
                     else
                        begin
                           if(({in1[1],in1[0]}==1 || {in1[1],in1[0]}==2) && in != 2'bz)
                              begin
                                 next_state<=STATE_1;
                                 out<={out,S1};
                              end
                        end
                  end
            end
         end
   STATE_1:
      begin
         if({in[1],in[0]}==2'b01)
            begin
```

```
                next_state<=STATE_3;
                out<={out,S1};
            end
        else
            begin
                if({in[1],in[0]}==2)
                    begin
                        next_state<=STATE_2;
                        out<={out,S0};
                    end
                else
                    begin
                        if({in1[1],in1[0]}==2 || {in1[1],in1[0]}==1)
                            begin
                                next_state<=STATE_3;
                                out<={out,S1};
                            end
                         else
                             begin
                                 if({in1[1],in1[0]}==0 || {in1[1],in1[0]}==3)
                                     begin
                                         next_state<=STATE_2;
                                         out<={out,S0};
                                     end
                             end
                    end
            end
    end
STATE_2:
    begin
        if({in[1],in[0]}==0)
            begin
                next_state<=STATE_1;
                out<={out,S1};
             end
        else
            begin
```

```
                if({in[1],in[0]}==3)
                    begin
                        next_state<=STATE_0;
                        out<={out,S0};
                    end
                else
                    begin
                        if({in1[1],in1[0]}==0 || {in1[1],in1[0]}==3)
                            begin
                                next_state<=STATE_0;
                                out<={out,S0};
                            end
                        else
                            begin
                                if({in1[1],in1[0]}==1 || {in1[1],in1[0]}==2)
                                    begin
                                        next_state<=STATE_1;
                                        out<={out,S1};
                                    end
                             end
                     end
              end
    end
STATE_3:
    begin
        if({in[1],in[0]}==2)
            begin
                next_state<=STATE_3;
                out<={out,S1};
            end
    else
        begin
            if({in[1],in[0]}==1)
                begin
                    next_state<=STATE_2;
                    out<={out,S0};
                end
            else
                begin
                    if({in1[1],in1[0]}==0 || {in1[1],in1[0]}==3)
                        begin
                            next_state<=STATE_2;
                            out<={out,S0};
                        end
                    else
                        begin
                            if({in1[1],in1[0]}==1 || {in1[1],in1[0]}==2)
                                begin
                                    next_state<=STATE_3;
                                    out<={out,S1};
                                end
                        end
                end
        end
    end
    endcase
end
//sequential process
always @(posedge clk or negedge rst_n)
    begin
        if(!rst_n)
            begin
                current_state=STATE_0;
                next_state=STATE_0;
                out=8'bz;
                S0=0;
                S1=1;
                in=2'bz;
                temp=2'bz;
                count=0;
                address=0;
            end
        else
            begin
```

```
                    temp = in1;//(cause half cycle delay) not non blking asst is used but output is right
                                      //because of error correcting circuit in state 0 but will not work properly if
                                      //cnts error occurs in first 2 input symbols
                     in = temp;
                     current_state=next_state;
               end
     end
always @(out)
     begin
          count<=count+1;
               if(count==7)
                    begin
                         ram[address] <= out;
                         address <= address + 1;
                    end
     end
endmodule
module stimulus;
reg    clk;
reg    rst_n;
reg    [1:0]in1;
wire   [7:0]out;
fsm fsm1
          (
          .clk(clk),
          .rst_n(rst_n),
          .in1(in1),
          .out(out)
          );
initial
begin
clk=1'b0;
forever #10 clk=~clk;
end
initial
begin
rst_n=1'b0;
```

```
# 5
rst_n=1'b1;

@(posedge clk or negedge clk )
in1=3;
# 10 in1=1;
# 10 in1=1;
# 10 in1=2;
# 10 in1=1;
# 10 in1=1;
#1000
$finish;
end
initial
$monitor("current_state=%d, next_state=%d, input=%d,
output=%d",fsm1.current_state,fsm1.next_state,in1,out);
endmodule
```

Viterbi Implementation

```
module fsm(clk, rst_n, enable, in, out);
input    clk;
input    rst_n;
input    [1:0]enable;
input    [1:0]in;
output  [7:0] out;
reg                 [7:0] out, data_in;
reg                  sel1, sel2, sel3, sel4;
reg                  maa, mab, mbc, mbd, mca, mcb, mdc, mdd, write_en;
reg                  [1:0] Saa, Sab, Sca, Scb, Sbc, Sbd, Sdc, Sdd;
reg                  [4:0] Ta, Tb, Tc, Td, Taa;
reg                  [7:0] ma, mb, mc, md;
reg                  [7:0] m1, m2, m3, m4, m5, m6, m7, m8;
reg                  [4:0] T1, T2, T3, T4,T5, T6, T7, T8;
reg                  [7:0] address;
wire                 [7:0]data_out;
reg                  [31:0] Tbb;
reg                  [7:0] ram [0 : 255];
//Combinational process
always @(posedge clk or rst_n or negedge clk)
begin
if (!rst_n)
begin
ma=8'bz;
mb=8'bz;
mc=8'bz;
md=8'bz;
Ta=5'b0;
Tb=5'b0;
Tc=5'b0;
Td=5'b0;
out = 8'bz;
address = 9'b0;
write_en=0;
data_in = 0;
```

```
maa = 0;
mab = 1;
mbc = 0;
mbd = 1;
mca = 0;
mcb = 1;
mdc = 0;
mdd = 1;
Taa = 0;
Tbb = 0;
end
else
begin
if(in==0)
begin
Saa = 2;
Sab = 0;
Sbc = 1;
Sbd = 1;
Sca = 2;
Scb = 0;
Sdc = 1;
Sdd = 1;
end
else
begin
if(in==1)
begin
Saa = 1;
Sab = 1;
Sbc = 2;
Sbd = 0;
Sca = 1;
Scb = 1;
Sdc = 0;
Sdd = 2;
end
```

```
else
begin
if(in==2)
begin
Saa = 1;
Sab = 1;
Sbc = 0;
Sbd = 2;
Sca = 1;
Scb = 1;
Sdc = 2;
Sdd = 0;
end
else
begin
if(in==3)
begin
Saa = 2;
Sab = 0;
Sbc = 1;
Sbd = 1;
Sca = 0;
Scb = 2;
Sdc = 1;
Sdd = 1;
end
end
end
end
if( enable == 0 )//||enable == 1 ||enable == 2 ||enable == 3 )
begin
Ta = Saa;
Tb = Sab;
ma = {ma,maa};
mb = {mb,mab};
end
else
```

```
begin
if( enable == 1  )
begin
Ta <= Ta + Saa;
Tb <= Ta + Sab;
Tc <= Tb + Sbc;
Td <= Tb + Sbd;
ma <= {ma,maa};
mb <= {ma,mab};
mc <= {mb,mbc};
md <= {mb,mbd};
end
else
begin
if( enable == 2 )
begin
T1 = Ta + Saa;
T2 = Tc + Sca;
m1 = {ma,maa};
m2 = {mc,mca};
if (T1 > T2)
sel1 = 1;
else
sel1 = 0;
case (sel1) // synopsys parallel_case
1'b0 : Ta <= T1;
1'b1 : Ta <= T2;
endcase
case (sel1) // synopsys parallel_case
1'b0 : ma <= m1;
1'b1 : ma <= m2;
endcase
T3 = Ta + Sab;
T4 = Tc + Scb;
m3 = {ma,mab};
m4 = {mc,mcb};
if (T3 > T4)
```

```
sel2 = 1;
else
sel2 = 0;
case (sel2) // synopsys parallel_case
1'b0 : Tb <= T3;
1'b1 : Tb <= T4;
endcase
case (sel2) // synopsys parallel_case
1'b0 : mb <= m3;
1'b1 : mb <= m4;
endcase
T5 = Tb + Sbc;
T6 = Td + Sdc;
m5 = {mb,mbc};
m6 = {md,mdc};
if (T5 > T6)
sel3 = 1;
else
sel3 = 0;
case (sel3) // synopsys parallel_case
1'b0 : Tc <= T5;
1'b1 : Tc <= T6;
endcase
case (sel3) // synopsys parallel_case
1'b0 : mc <= m5;
1'b1 : mc <= m6;
endcase
T7 = Tb + Sbd;
T8 = Td + Sdd;
m7 = {mb,mbd};
m8 = {md,mdd};
if (T7 > T8)
sel4 = 1;
else
sel4 = 0;
case (sel4) // synopsys parallel_case
1'b0 : Td <= T7;
```

```
1'b1 : Td <= T8;
endcase
case (sel4) // synopsys parallel_case
1'b0 : md <= m7;
1'b1 : md <= m8;
endcase
end
end
end
if((ma[7] == mb[7] && mb[7] == mc[7] && mc[7] == md[7]))
begin
out = {out,ma[7]};
Taa = Taa + 1;
if(Taa == 8)
begin
address = address + 1;
ram[address] = out;
Taa = 0;
Tbb = ram[address];
end
end
end
end
endmodule

module stimulus;
reg    clk;
reg    rst_n;
reg    [1:0]in;
reg    [1:0]enable;
wire   [3:0]ma, mb, mc, md;
fsm fsm1
(
.clk(clk),
.rst_n(rst_n),
.enable(enable),
.in(in)
);
initial
begin
clk=1'b0;
forever #10 clk=~clk;
end
initial
begin
# 10 enable = 0;
# 10 enable = 1;
# 10 enable = 2;
end
initial
begin
rst_n=1'b0;
# 5 rst_n=1'b1;
@(posedge clk or negedge clk)
begin
in=3;
# 10 in=1;
# 10 in=1;
# 10 in=2;
# 10 in=1;
# 10 in=1;
# 10 in=0;
# 10 in=2;
# 10 in=0;
# 10 in=2;
# 10 in=1;
end
#1000
$finish;
end
initial
$monitor("ma=%d, mb=%d, mc=%d, md=%d, input=%d",ma, mb, mc, md,in);
endmodule
```