Design and Implementation of Viterbi Decoder using FPGAs

A Thesis
Submitted towards the partial fulfillment of the requirements of the degree of

Master of Technology
in
VLSI Design and CAD

by
Ajay Sharma
Registration No.-6050402

Under the Guidance of :
Ms. Harpreet Vohra

Electronics and Communication Engineering Department
THAPAR UNIVERSITY
PATIALA (Punjab) - 147004 (INDIA)

MARCH , 2008

DECLARATION

I hereby declare that the thesis entitled "Design And Implementation of Veterbi Decoder using FPGA " is an authentic record of my study carried out as requirements for the award of degree of M.Tech. (VLSI Design & CAD) at Thapar University, Patiala, under the guidance of Mrs.Harpreet Vohra during July 2007 to March 2008.

Ajay Sharma
Roll. No. 6050402

Date: 10-03-2008

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

Thesis Guide
Mrs.Harpreet Vohra
Lecturer
Thapar University, Patiala

Countersigned by

HECED
Thapar University
Patiala

DOAA
Thapar University
Patiala

i

ACKNOWLEDGMENTS

I am highly grateful to Department of Electronics and Communication Engineering, Thapar University, Patiala for providing this opportunity to carry out this project.

The constant guidance and encouragement received from Mrs. Harpreet Vohra Lecturer in Department of Electronics & Communication Engineering, has been of a great help in carrying out this present project and is acknowledged with reverential thanks.

I would also like to express a deep sense of gratitude & thanks to Dr.Sanjay Sharma (Asst. Professor), Department of Electronics & Communication Engineering, as without his wise & able guidance it would have been impossible to present this project in this manner.

The constant guidance and encouragement received from Asst.Prof.Alpana Aggrawal, PG Coordinator, Department of Electronics & Communication Engineering, is acknowledged with reverential thanks.

AJAY SHARMA

Regd. No.: 6050402

ABSTRACT

Convolutional encoding is a forward error correction technique that is used for correction of errors at the receiver end. The two decoding algorithms used for decoding the convolutional codes are Viterbi algorithm and Sequential algorithm. Sequential decoding has advantage that it can perform very well with long constraint length. Viterbi decoding is the best technique for decoding the convolutional codes but it is limited to smaller constraint lengths.

It has been widely deployed in many wireless communication systems to improve the limited capacity of the communication channels. The Viterbi algorithm, is the most extensively employed decoding algorithm for convolutional codes. .

The availability of wireless technology has revolutionized the way communication is done in our world today. With this increased availability comes increased dependence on the underlying systems to transmit information both quickly and accurately. Because the communications channels in wireless systems can be much more hostile than in “wired” systems, voice and data must use forward error correction coding to reduce the probability of channel effects corrupting the information being transmitted. A new type of coding, called Viterbi coding, can achieve a level of performance that comes closer to theoretical bounds than more conventional coding systems. The Viterbi Algorithm, an application of dynamic programming, is widely used for estimation and detection problems in digital communications and signal processing.It is used to detect signals in communications channels with memory, and to decode sequential error control codes that are used to enhance the performance of digital communication systems.

Though various platforms can be used for realizing Viterbi Decoder including Field Programmable Gate Arrays (FPGAs) , Complex Programmable Logic Devices (CPLDs) or Digital Signal Processing (DSP) chips but in this project benefits of using an FPGA to Implement Viterbi Decoding Algorithm has been described. FPGAs are a technology that gives the designer flexibility of a programmable solution, the performance of a custom solution and lowering overall cost. The advantages of the FPGA approach to DSP Implementation include higher sampling rates than are available from traditional DSP chips, lower costs than an ASIC. The FPGA also adds design flexibility and adaptability with optimal device utilization conserving both board space and system power that is often not the case with DSP chips.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

In digital communication system, error detection and error correction is important for reliable communication. Error detection techniques are much simpler than forward error correction (FEC). But error detection techniques have certain disadvantages. Error detection pre supposes the existence of an automatic repeat request (ARQ) feature which provides for the retransmission of those blocks, segments or packets in which errors have been detected. This assumes some protocol for reserving time for the retransmission of such erroneous blocks and for reinserting the corrected version in proper sequence. It also assumes sufficient overall delay and corresponding buffering that will permit such reinsertion. The latter becomes particularly difficult in synchronous satellite communication where the transmission delay in each direction is already a quarter second. A further drawback of error detection with ARQ is its inefficiency at or near the system noise threshold. For, as the error rate approaches the packet length, the majority of blocks will contain detected errors and hence require retransmission, even several times, reducing the throughput drastically. In such cases, forward error correction, in addition to error detection with ARQ, may considerably improve throughput. Forward error correction may be desirable in place of, or in addition to, error detection for any of the following reasons:

(1) When a reverse channel is not available or the delay with ARQ would be excessive.

(2) The retransmission strategy is not conveniently implemented.

1.1 ORGANIZATION OF THESIS

Keeping in view requirements of communication channels in 3G wireless systems, need of reliable data communication, fast as well as accurate is the main consideration. So main work in this thesis is to develop an algorithm for a Viterbi Decoder to decode an encoded data. In Chapter two various channel coding algorithms have been discussed. Channel coding refers to the class of signal transformations designed to improve communications performance by enabling the transmitted signals to better withstand the effects of various channel impairments, such as noise, interference, and fading.

Chapter three gives brief introduction of Viterbi Algorithm. The Viterbi algorithm is widely used for the elimination of the potential noise in a data stream. It belongs to a large class of error correcting codes known as convolution codes.

Chapter four provides brief introduction to architecture of Field Programmable Gate Arrays. Programmable devices are a class of general-purpose chips that can be reconfigured for a wide variety of applications. The first programmable device that achieved widespread use was the PROM (Programmable Read-Only Memory). Another step was the development of PLD (Programmable Logic Devices). These devices were constructed to implement logic circuits. The PLD includes an array AND gates connect to an array of OR gates. The PAL (Programmable Array Logic) is a commonly used PLD consisting of a programmable AND-plane followed by a fixed OR-plane. The PAL was designed for small logic circuits.

The MPGA (Mask Programmable Gate Array) was developed to handle larger logic circuits. A common MPGA consists of rows of transistors that can be interconnected to implement desired logic circuits. User specified contents are available both within the rows and between the rows. This enabled implementation of basic logic gates and the ability to interconnect the gates. In 1985, Xilinx Inc. introduced the FPGA (Field Programmable Gate Array). The Interconnects between all the elements was designed to be user programmable.

Chapter five describes method of implementing Viterbi Algorithm on FPGA.

Results which we got are discussed in Chapter six.Viterbi test Bench is created using Xilinx Webpack.Code. To prove the correctness of our design, the verilog HDL description was simulated & tested using Model Sim Verilog Simulator. Afterwards Xilinx Webpack is used for design entry, synthesis, place & route and floor plan design.

Chapter seven concludes the thesis discussing successes and failure of this work and also suggesting some problems for future work.

CHAPTER-2

CHANNEL CODING ALGORITHMS

2.1 INTRODUCTION

The class of signal transformations designed to improve communications performance by enabling the transmitted signals to better withstand the effects of various channel impairments, such as noise, interference, and fading is referred to as channel coding.

The use of large-scale integrated circuits (LSI) and high-speed digital signal processing (DSP) techniques have made it possible to provide as much as 10 dB performance improvement through these methods, at much less cost than through the use of most other methods such as higher power transmitters or larger antennas.

2.2 WAVEFORM CODING

It is known that noise-immunity is one of the basic attributes of information transmission systems. Since errors are possible in communication channels during the data transmissions we must apply error-correcting codes to combat these errors . The purpose of forward error correction (FEC) is to improve the capacity of channel by adding some carefully designed redundant information to the data being transmitted through the channel. The process of adding this redundant information is known as channel coding.

Channel coding can be partitioned into two study areas, waveform (or signal design) coding and structured sequences (or structured redundancy) [1]. Waveform coding deals with transforming waveforms into "better waveforms," to make the detection process less subject to errors. Structured sequences deal with transforming data sequences into "better sequences," having structured redundancy (redundant bits). The redundant bits can then

be used for the detection and correction of errors. The encoding procedure provides the coded signal (whether waveforms or structured sequences) with better distance properties than those of their uncoded counterparts.

2.3 TYPES OF ERROR CONTROL

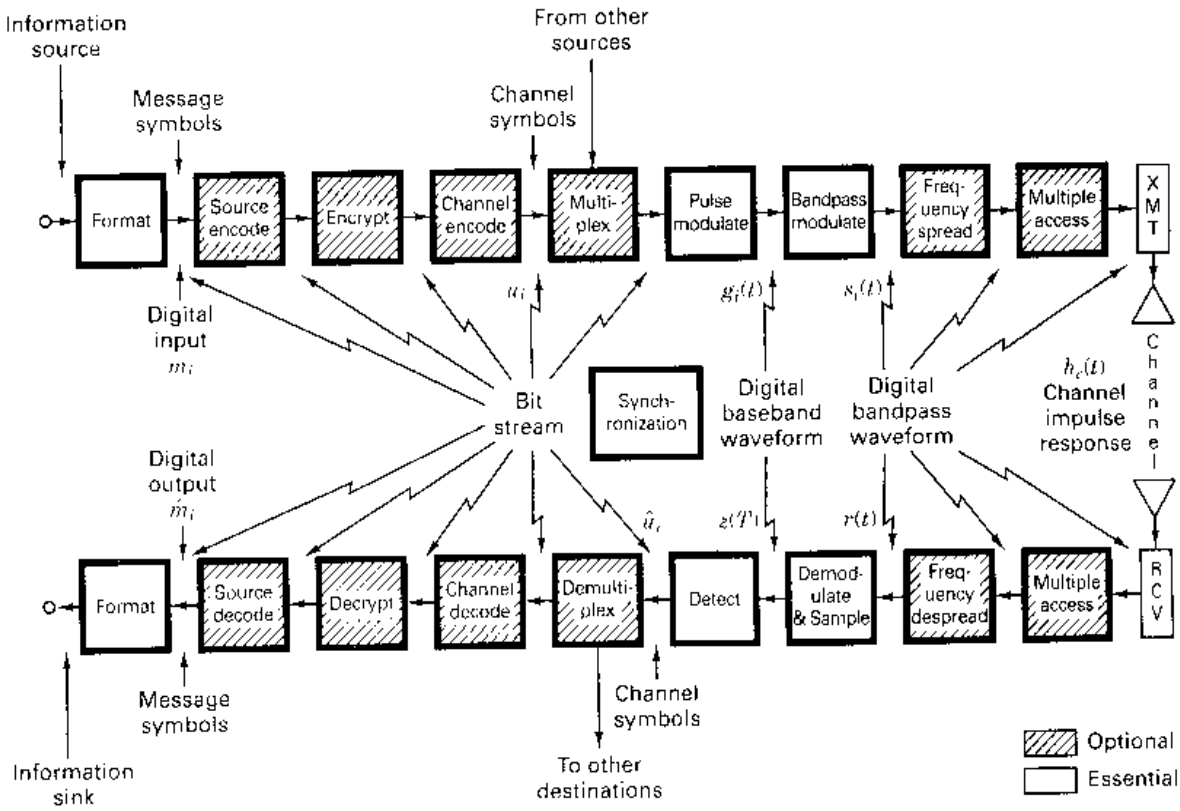


Figure 2.1 Block Diagram of a typical Digital Communication System

Two basic ways by which redundancy is used for controlling errors are:

The first, error detection and retransmission, utilizes parity bits (redundant bits added to the data) to detect that an error has been made. The receiving terminal does not attempt to correct the error; it simply requests that the transmitter retransmit the data. Notice that a two-way link is required for such dialogue between the transmitter and receiver. The second type of error control, forward error correction (FEC) requires a one-way link only, since in this case the parity bits are designed for both the detection and correction of errors.

2.4 FORWARD ERROR CORRECTION METHODS (FECs)

The availability of wireless technology has revolutionized the way communications is done in our world today [2]. Cellular and satellite technology makes it possible for people to be connected to the rest of the world from anywhere. With this increased availability comes increased dependence on the underlying systems to transmit information both quickly and accurately. Because the communications channels in wireless systems can be much more hostile than in "wired" systems, voice and data must use forward error correction coding to reduce the probability of channel effects corrupting the information being transmitted. A new type of coding, called Viterbi coding, can achieve a level of performance that comes closer to theoretical bounds than more conventional coding systems.

2.5 TRADITIONAL CODING SCHEMES

In any real communication system, transmission over a channel will cause the information received at the other end to differ from what was originally transmitted. This is because the channel injects noise into the original signal. In digital communication systems, if the power in the noise is large enough relative to the power in the original signal, transmitted bits can be corrupted to the point that the receiver makes incorrect decisions about the data that was transmitted. If too many of these errors occur (in data communications "too many" can often mean one), the communication system will not be usable. Therefore, techniques must be used to reduce the probability of an error occurring for a given signal-to-noise ratio (SNR)[3].In 1948, Claude Shannon proved that the probability of such an error occurring could be theoretically reduced to zero, given that the transmission rate does not exceed the capacity of the channel, *C*. In the case of transmission of a signal of average power *S* over a channel of bandwidth *B* that injects additive white Gaussian noise (AWGN) with power *N*, the capacity is given by

$$C=B \log_2 \left(1+\frac{S}{N} \right)$$

For a communication system to begin to approach this limit, it cannot transmit the data "as is" with no additional processing; a single instance of high noise could cause a bit error.

In computer networks, this problem is typically solved by an automatic repeat request (ARQ) scheme as already discussed above .In such a scheme, a checksum or other redundancy is added to a data frame to allow error detection at the receiver. If no error is detected, the receiver sends an acknowledgement (ACK) back to the sender. If there is a

frame error, then the receiver either sends a negative acknowledgement (NACK) to the sender, or it does nothing and lets the sender retransmit the data after it times out of the period in which it waits for an ACK. The ARQ technique is not typically used in wireless communications for a couple of reasons. One reason is that ARQ systems require duplex communication capabilities. While this is relatively cheap and easy to do on a wired network, it requires an increase in receiver complexity for wireless links that is often unacceptable. Also, in computer networks the probability of bit errors caused by noise is very low in many cases, so a frame retransmission is rarely necessary [1]. This is in great contrast to wireless networks, where bit errors are usually much more frequent. The number of frame retransmissions required to implement an ARQ scheme on a system with low SNR would slow throughput to a near-halt. Since the ARQ scheme is impractical for many communication systems, forward error correction (FEC) schemes are often used instead. FEC codes are designed to improve the decisions that the receiver makes by giving it enough information to correct some of the errors that the channel has introduced into the signal. This performance improvement is largely brought about by two techniques, *redundancy* and *noise averaging* By adding redundant bits to the digital message, the encoder accentuates the uniqueness of the transmitted message. This eases the decision burden of the receiver because limiting allowable sequences to a fraction of those possible means that multiple bits will have to be corrupted by noise for the message to be decoded incorrectly. In noise averaging, the code is designed so the bits of the message affect many bits of the encoder output, allowing the receiver to average out effects of the noise over a large number of received bits. While FEC schemes can provide exceptional improvements in performance for noisy channels, their benefits do not come

without costs. The use of coding requires additional processing at the transmitter and especially at the receiver, as can be seen in Figure 2.2. While encoders do not typically require large amounts of processing, the complexity of decoders can be significant. For more complicated coding techniques, the complexity of optimal decoders prohibits their implementation because they require large amounts of hardware and large amounts of processing time. In these cases, sub-optimal approximations that require less processing and provide inferior performance can be used. Also, coding reduces the data rate by a constant factor for a fixed bandwidth due to the transmission of redundant bits. The general challenge in coding system design is to provide enough randomness in the code to provide good performance in noise while providing enough structure in the code to make its decoding feasible [4].

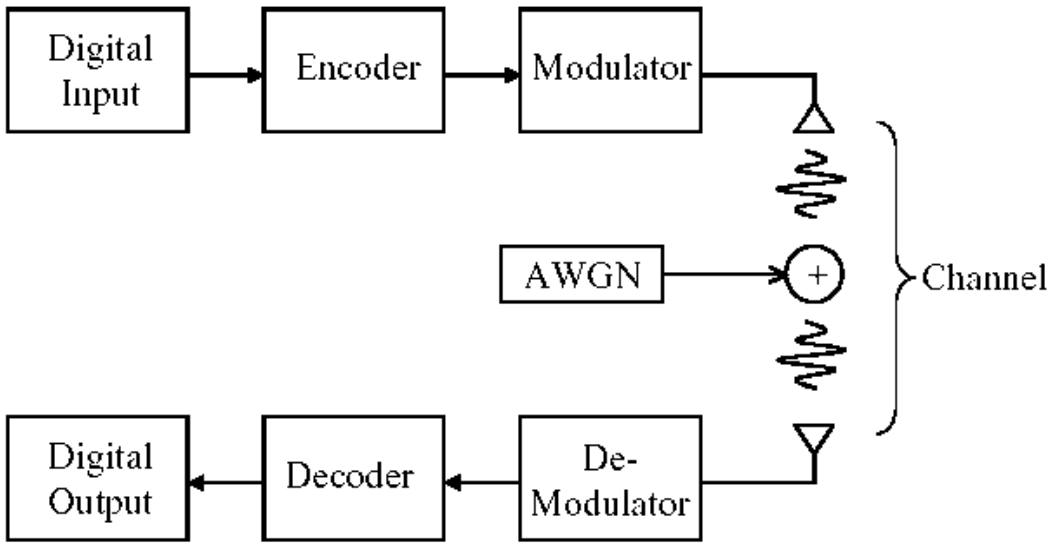


Figure 2.2 Diagram of a communications system with forward error correction (FEC).

Traditionally, coding schemes have been divided into two distinct categories: block coding and convolutional coding. Although the lines of this division have begun to blur as advanced coding schemes, like turbo coding, use elements of both types of codes, it is still useful to discuss each type of code separately to accentuate the differences between the schemes.

2.6 BLOCK CODES

As the name implies, block coding is performed by partitioning the data stream into fixed-size messages for encoding. The messages of *k* bits are mapped to code words of length *n* bits. The *code rate*, *R*, for an (*n*, *k*) block code is then given by

$$R=\frac{k}{n}$$

For some block codes, the codeword consists of the *k* original message bits with *n* – *k*

$$\mathbf{x}=[\mathbf{m} \quad \mathbf{p}]$$

parity bits appended to it. The codeword **x** is thus represented in matrix notation as where

$$\mathbf{m}=[m_1, m_2, \ldots m_k] \quad \text{and} \quad \mathbf{p}=[p_1, p_2, \ldots p_{n-k}]$$

are the message and parity bits, respectively. Because the message itself is part of the codeword, this code is referred to as *systematic*.

The encoder calculates **x** by a matrix multiplication of **m** with a generator matrix,

G.

$$\mathbf{x}=\mathbf{mG}$$

For the systematic code described above, **G** is a concatenation of a k × k identity matrix **I**, which generates **m**, and a k × (n-k) parity matrix **Z**, which generates **p**.

$$\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{Z} \end{bmatrix}$$

A code generated in this manner is called *linear*. The Hamming distance between any two codewords **x1** and **x2** is the number of bit positions in which their binary representations differ. This will be expressed as

$$d_h(\mathbf{x}_1, \mathbf{x}_2)$$

The Hamming weight, w_h, of a codeword **x** is the number of 1's in its binary representation or, alternatively,

$$w_h(\mathbf{x}) = d_h(\mathbf{x}, \mathbf{0}).$$

If the Hamming distance is computed for all possible codeword pairs, then the minimum Hamming distance, *dmin*, for the code is found by choosing the smallest calculated distance. For a linear code, *dmin* is equivalent to the minimum nonzero w_h. This is significant because the number of bit errors that can be corrected by the code, *t* is given by

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor.$$

Thus, by designing the code for a large d*min*, the error-correcting power of the code will be greater [5]. However, increasing d*min* requires an increase in n relative to k, which decreases the efficiency of the code, as defined by the code rate, R. By using large codewords, R can be brought up to acceptable levels for a given dmin, but encoder and decoder complexity scale up with n, so there is a limit to the amount that the code performance can be improved by increasing codeword length. Since block encoding is a memoryless one-to-one mapping, it can be viewed most simply as a look-up operation. The encoder uses the *k*-bit message as an index to a table that contains the *n*-bit codeword that it outputs, as shown in Figure 2.3. Such a look-up table would have 2*k* entries of *n* bits. For very large codewords, this can be prohibitively complex to implement for both encoding and (especially) decoding.

A subset of block codes called *cyclic codes* can provide a solution to the implementation woes of long block codes [6]. A sample systematic cyclic encoder is shown in Figure 2.4. By using a linear feedback shift register (LFSR), the parity bits are calculated as the message bits are being sent out of the encoder. Then, the switches are flipped, and the parity bits stream out.

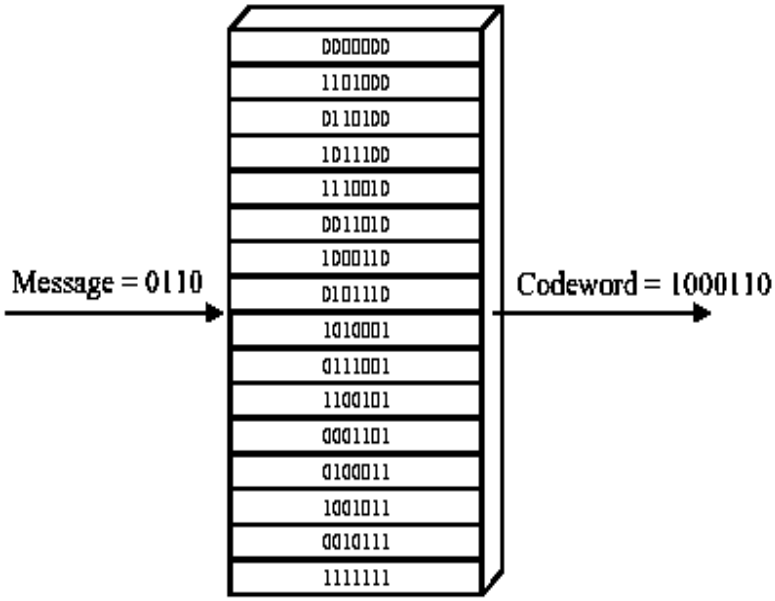


Figure 2.3 Lookup table representation of a (7,4) block encoder with *dmin* = 3.

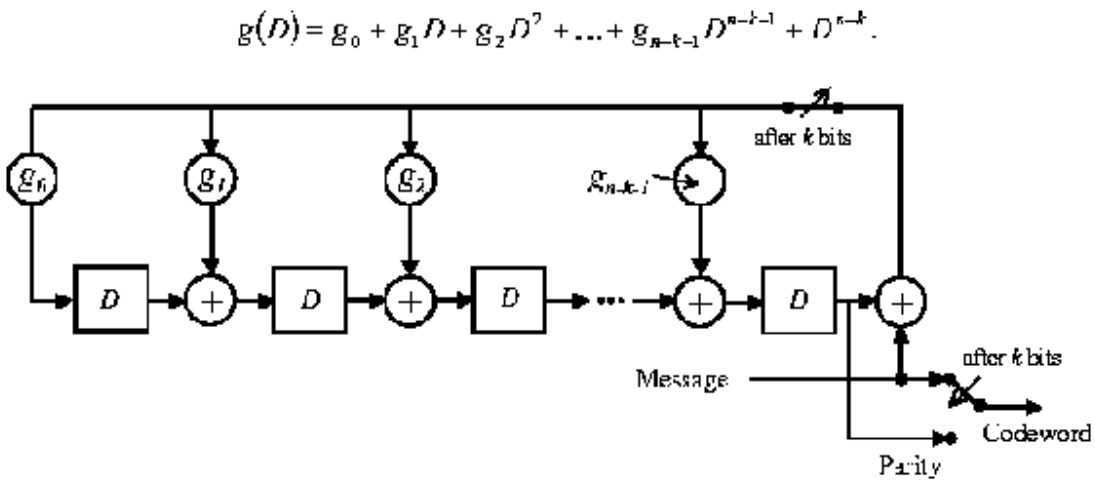


Figure 2.4: Block diagram of an encoder for a systematic cyclic code.

This implementation is much less complex than the general case for block codes, because it only requires n - k memory elements in the LFSR to implement the encoder. Cyclic codes also have the beneficial property that their encoder structure can be represented by a single generator polynomial.

Decoding block codes is a much more complicated operation than encoding due to the bit errors caused by the noisy channel. If the look-up table scheme discussed for the encoder is extended to decoding, there are two implementation options. The n-bit block could be used as an index to a massive $2 \times k$ bit look-up table, or the n-bit block could be compared with all of the entries of a $2k \times n$ table. Neither of these options, while providing optimal results, is computationally feasible for all but the shortest block codes [7]. One tractable, but sub-optimal alternative is to use an algebraic decoding strategy like the Berlekamp algorithm, which allows decoding of all combinations of errors up to the error correction capability, t. In the case of cyclic codes, their increased structure allows for much simpler decoding algorithms like the Meggitt decoder or the Berlekamp- Massey algorithm.

CHAPTER-3

CONVOLUTIONAL CODES AND VITERBI DECODING

3.1 CONVOLUTIONAL CODES

The convolutional encoder is basically a finite state machine. The k bit input is fed to the constraint length K shift register and the n outputs are calculated from the generator polynomials by the modulo-2 addition. The generator polynomial specifies the connections of the encoder to the modulo-2 adder. The 1 in the generator polynomial indicates the connections and zero indicates no connections between the stage and the modulo 2 adder. The figure below illustrates a simple convolutional coder with k=1, K=3, n=3, g1 (n) = (1 0 1), g2 (n) = (1 1 1), g3 (n) = (0 1 1) and R=1/2.

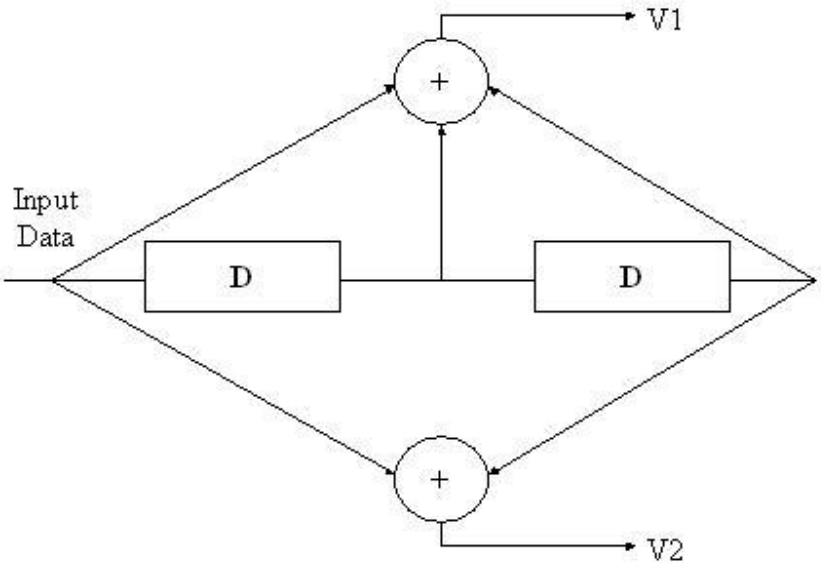


Figure 3.1: Convolutional Encoder

Convolutional encoder can be described in terms of state table, state diagram and trellis diagram. The State is defined as the contents of the shift register of the encoder. In state table output symbol can be described as a function of input symbol and the state. State

diagram shows the transition between different states. Trellis diagram is the description of state diagram of the encoder by a time line i.e. to represent each time unit with a separate state diagram [3].

Input u	Present state (S_1,S_0)	Next state (S_1,S_0)	Output (v_1,v_2)
0	0 0	0 0	0 0
1	0 0	0 1	1 1
0	0 1	1 0	1 1
1	0 1	1 1	0 0
0	1 0	0 0	1 0
1	1 0	0 1	0 1
0	1 1	1 0	0 1
1	1 1	1 1	1 0

Figure 3.2: State Table

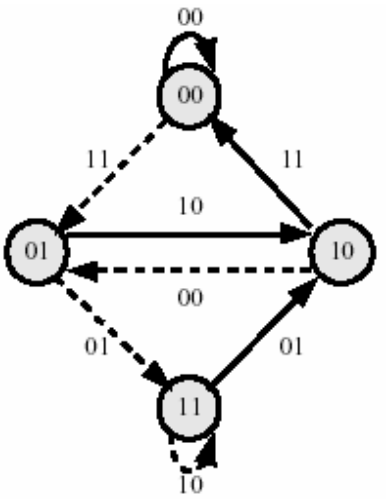


Figure 3.3: State Diagram

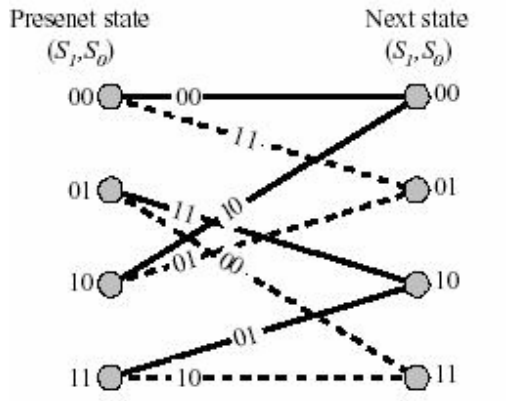


Figure 3.4: Trellis Diagram

The encoding for the sequence 0 1 1 0 1 0 0 and the output sequence is 00 11 00 01 01 11 10 is shown in the Figure 3.5.

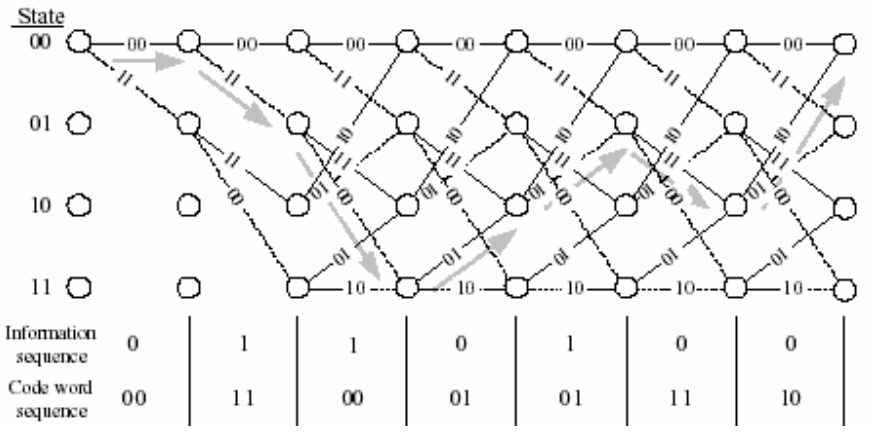


Figure 3.5: Trellis Diagram for Encoder

3.2 VITERBI ALGORITHM

Viterbi algorithm was introduced in 1967 by Viterbi. Viterbi algorithm is called as optimum algorithm because it minimizes the probability of error. The algorithm can be broken down into the following three steps.

1. Weigh the trellis; that is, calculate the branch metrics.

2. Recursively computes the shortest paths to time n, in terms of the shortest paths to time n-1. In this step, decisions are used to recursively update the survivor path of the signal. This is known as add-compare-select (ACS) recursion.

3. Recursively finds the shortest path leading to each trellis state using the decisions from Step 2. The shortest path is called the survivor path for that state and the process is referred to as survivor path decode. Finally, if all survivor paths are traced back in time, they merge into a unique path, which is the most likely signal path [8].

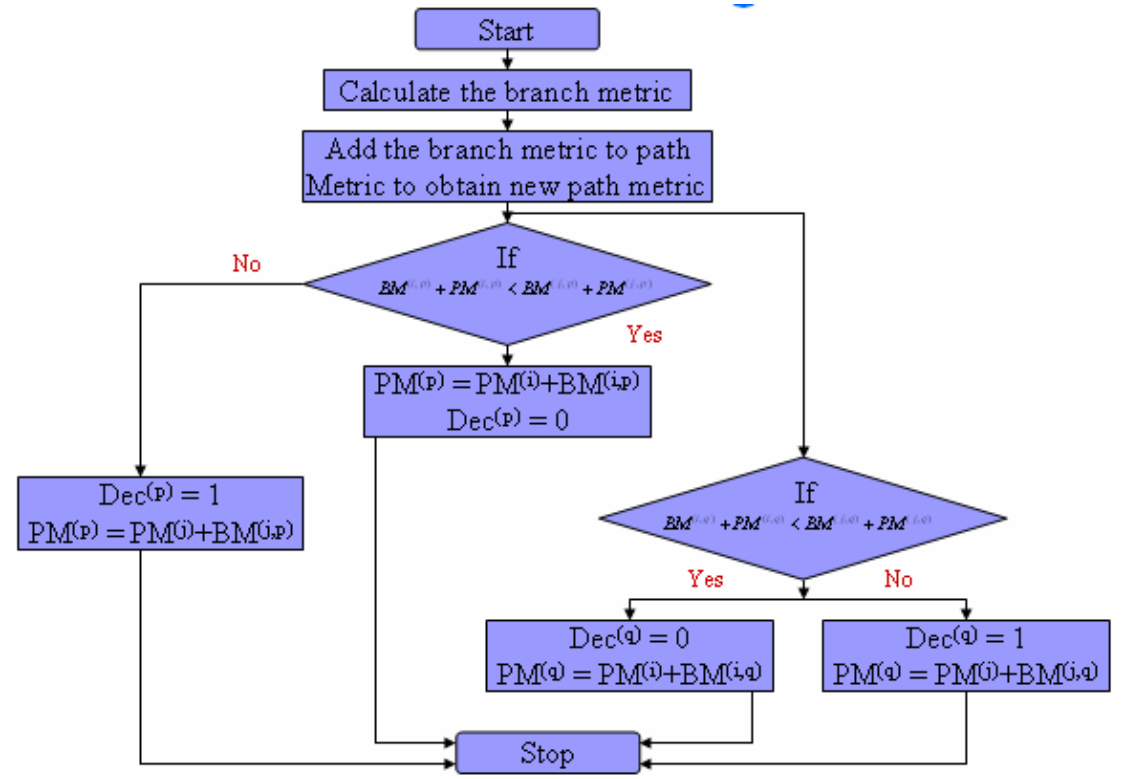


Figure 3.6: Flowchart for Viterbi Algorithm

3.3 VITERBI DECODING FOR ERROR FREE CHANNEL

The sequence of Figure 3.5 is assumed. Figure 3.7 shows the decoding sequence for two code words received. The hamming distance between the codewords being received and the output of encoder is calculated.

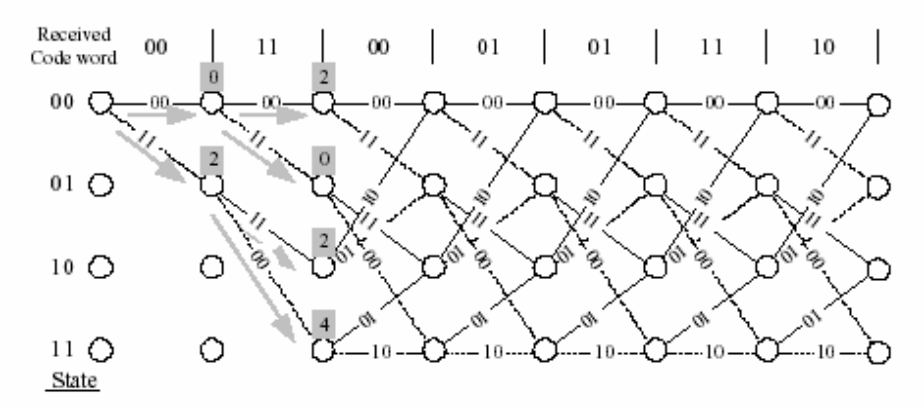


Figure 3.7: Decoding Example for Calculation of Hamming Distance

Consider the third code word being received. The path metrics are simply the addition of branch metric and previous path metric. For state (00), the partial path metric through the line marked 00 is 2, while the partial path metric through the line marked 10 is 3. The former is less than the later. So the survivor path of state (00) is through the solid line and path metric for the state is updated as 2. The same operations are done for each state.

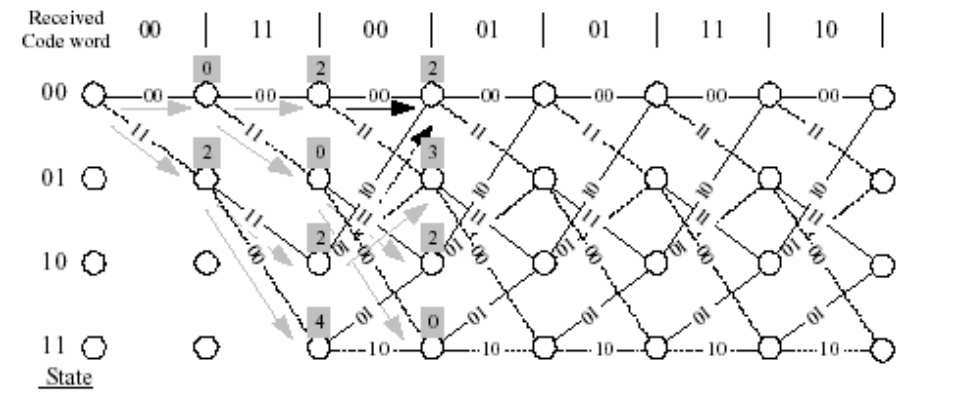


Figure 3.8: Decoding Example for Calculation of Path Metric

When the computation of path metric is done, the next step is to decode the most similar sequence. The decoding process is denoted as traceback because this process is like tracing the sequence back. We trace the best path from the state with the minimum path metric. The final path is highlighted in Figure 3.9.

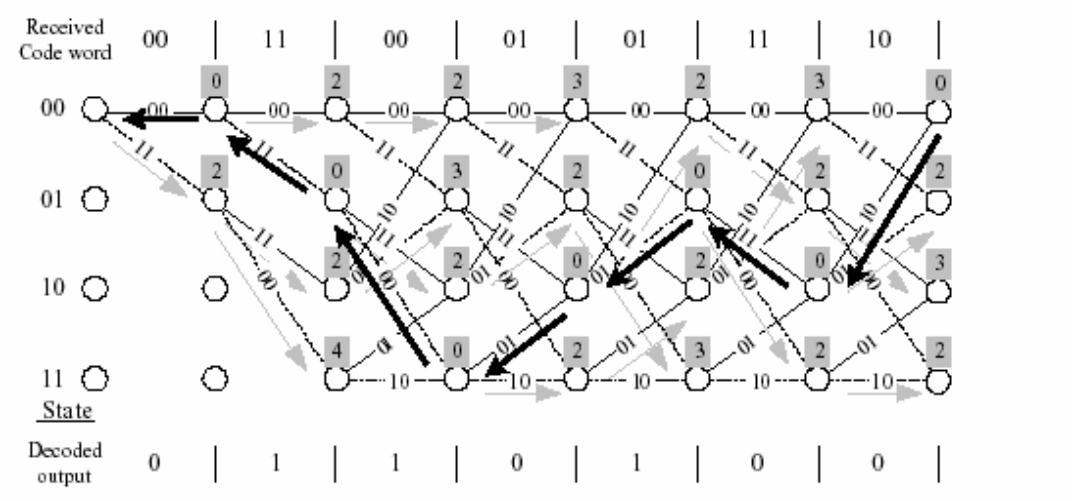


Figure 3.9: Decoding by Traceback Method

3.4 VITERBI DECODING FOR NOISY CHANNEL

Consider now the case in which an error is made on channel and one bit of the received symbol is incorrect. The fourth codeword has error. Figure 3.10 shows first four steps of decoding process.

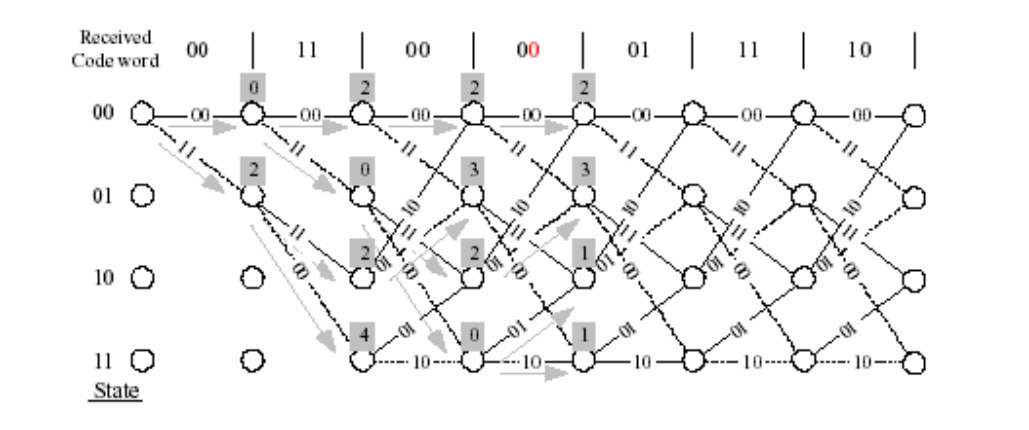


Figure 3.10: Decoding for Noisy Channel for Four Code Words

Similar steps are followed for all the codewords and complete diagram is as shown in Figure 3.11.

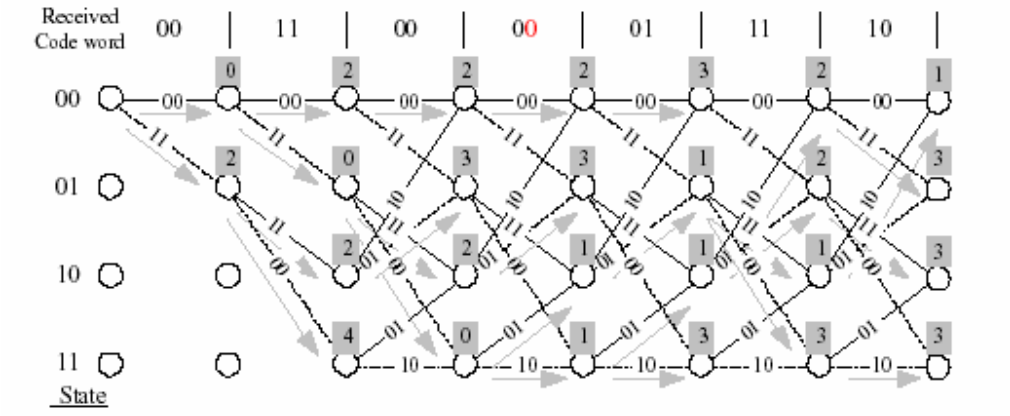


Figure 3.11: Complete Decoding Diagram

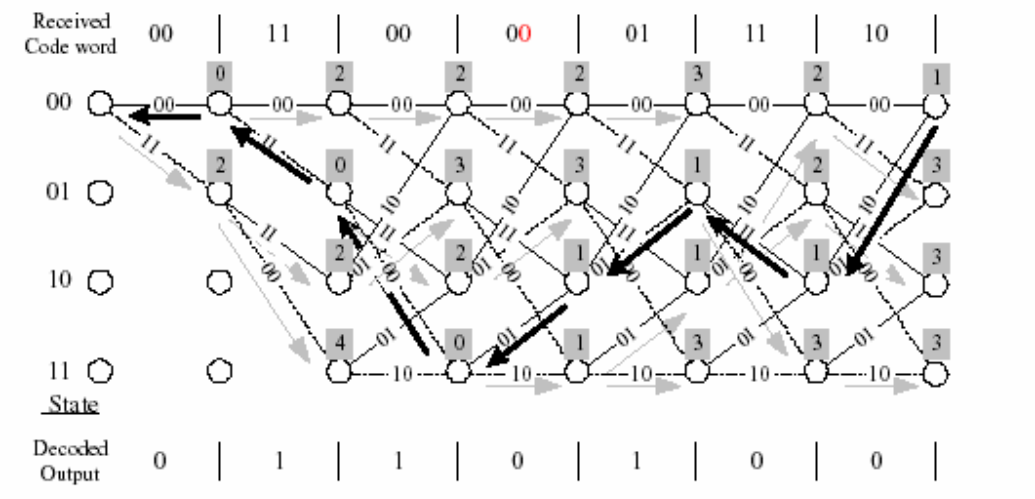


Figure3.12: Decoding for Noisy Channel Using Traceback Method

The Figure 3.12 shows that when the final path is traced back even with one bit of error,output is correct.

3.5 VITERBI DECODER

The basic units of Viterbi decoder are branch metric unit, add compare and select unit and survivor memory management unit [9, 18].

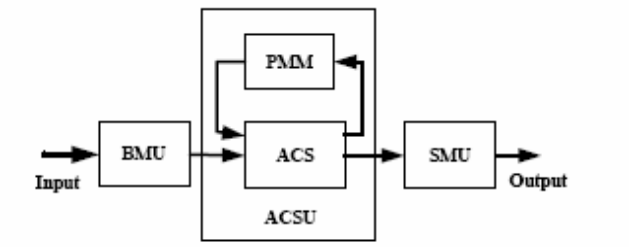


Figure 3.13: Block Diagram of Viterbi decoder

3.5.1 BRANCH METRIC UNIT

The first unit is called branch metric unit. Here the received data symbols are compared to the ideal outputs of the encoder from the transmitter and branch metric is calculated. Hamming distance or the Euclidean distance is used for branch metric computation.

3.5.2 PATH METRIC UNIT

The second unit, called path metric computation unit, calculates the path metrics of a stage by adding the branch metrics, associated with a received symbol, to the path metrics from the previous stage of the trellis.

3.5.3 SURVIVOUR MEMORY MANAGEMENT UNIT

The final unit is the trace-back process or register exchange method, where the survivor path and the output data are identified. The trace-back (TB) and the register-exchange (RE) methods are the two major techniques used for the path history management in the chip designs of Viterbi decoders.

The TB method takes up less area but requires much more time as compared to RE method because it needs to search or trace the survivor path back sequentially. Also, extra hardware is required to reverse the decoded bits. The major disadvantage of the RE approach is that its routing cost is very high especially in the case of long-constraint lengths and it requires much more resources.

3.5.3.1 TRACEBACK METHOD

In the TB method, the storage can be implemented as RAM and is called the path memory. Comparisons in the ACS unit and not the actual survivors are stored. After at least L branches have been processed, the trellis connections are recalled in the reverse order and the path is traced back through the trellis diagram

The TB method extracts the decoded bits; beginning from the state with the minimum PM. Beginning at this state and tracing backward in time by following the survivor path, which originally contributed to the current PM, a unique path is identified. While tracing back through the trellis, the decoded output sequence, corresponding to the traced branches, is generated in the reverse order. Trace back architecture has a limited memory bandwidth in nature, and thus limits the decoding speed.

3.5.3.2 REGISTER EXCHANGE METHOD

The register exchange (RE) method is the simplest conceptually and a commonly used technique. Because of the large power consumption and large area required in VLSI implementations of the RE method, the trace back method (TB) method is the preferred method in the design of large constraint length, high performance Viterbi decoders. In the register exchange, a register assigned to each state contains information bits for the survivor path from the initial state to the current state. In fact, the register keeps the partially decoded output sequence along the path, as illustrated in Figure 3.14. The register of state S1 at t=3 contains '101'. This is the decoded output sequence along the hold path from the initial state.

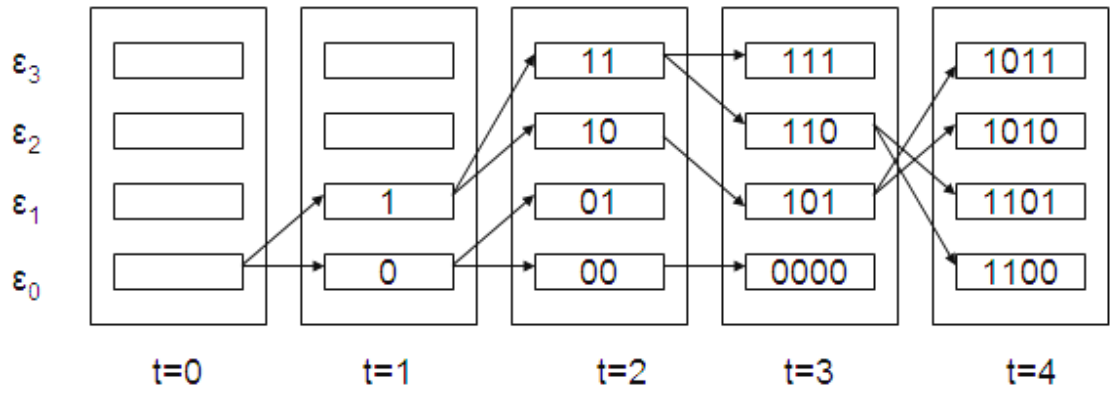


Figure 3.14: Register Exchange Method

The register-exchange method eliminates the need to trace back since the register of the final state contains the decoded output sequence. However, this method results in complex hardware due to the need to copy the contents of all the registers in a stage to the next stage. The survivor path information is applied to the least significant bit of each register, and all the registers perform a shift left operation at each stage to make room for the next bits. Hence, each register fills in the survivor path information from the least significant bit toward the most significant bit. The scheme is called shift update. The shift update method is simple in implementation but causes high switching activity due to the shift operation and, hence, results in high power dissipation.

3.6 TYPES OF VITERBI DECODING

In order to realize a certain coding scheme a suitable measure of similarity or distance metric between two code words is vital. The two important metrics used to measure the distance between two code words are the Hamming distance and Euclidian distance adopted by the decoder depending on the code scheme, required accuracy, channel characteristics and demodulator type.

3.6.1 HARD DECISION VITERBI DECODING

In the hard-decision decoding, the path through the trellis is determined using the Hamming distance measure. Thus, the most optimal path through the trellis is the path with the minimum Hamming distance. The Hamming distance can be defined as a number of bits that are different between the observed symbol at the decoder and the sent symbol from the encoder. Furthermore, the hard decision decoding applies one bit quantization on the received bits.

3.6.2 SOFT DECISION VITERBI DECODING

Soft-decision decoding is applied for the maximum likelihood decoding, when the data is transmitted over the Gaussian channel. On the contrary to the hard decision decoding, the soft-decision decoding uses multi-bit quantization for the received bits, and Euclidean distance as a distance measure instead of the hamming distance. The demodulator input is now an analog waveform and is usually quantized into different levels in order to help the decoder decide more easily. A 3-bit quantization results in an 8-ary output [11, 14].

3.7 LIMITATION OF VITERBI ALGORITHM

When a binary convolutional code with k=1 and constraint length K is decoded by means of VA, there are 2K−1 states. Convolutional code in which k bits are shifted at a time into the shift register with K stages generates a trellis that has (1) 2 k K− states. Consequently, the decoding of such a code by means of VA requires keeping track of (1) 2 k K− surviving paths and (1) 2 k K− metrics. At each stage of the trellis, there are k 2

paths that merge at each node. Since each path that converges at common node requires the computation of a metric, there are k 2 metrics computed for each node. Of the k 2 paths that merge at each node, only one survives and this is the minimum distance path. Thus the number of computations in decoding performed at each stage increases exponentially with k and K. The exponential increase in computations and storage required to implement make it impractical for convolutional codes with large constraint length [2].

3.8 OTHER CODING TECHNIQUES

There are other structures and coding techniques that form excellent channel coding schemes in many practical applications. These structures include punctured convolutional codes, which effectively increase the rate R of a code by deleting periodically a code bit and concatenated codes, which combine two different code structures usually an RS outer code with an inner binary convolutional code to produce a powerful code. Also, another very important method, which was partly developed by Ramsey, is the convolutional interleaving scheme that finds many applications especially in channels where the errors occur in bursts. This technique uses an interleaver fed straight from the encoder, which is essentially a bank of registers and effectively separates each code bit from the other by changing their order in time before they are sent over the channel. In the receiver end, a synchronized de interleaver is used which performs the inverse operation and reassembles the original code sequence before fed to the decoder.

For the special case of k = 1, the codes of rates 1/2, 1/3, 1/4, 1/5, 1/7 are sometimes called mother codes. We can combine these single bit input codes to produce punctured codes, which give us code rates other than 1/n.

By using two rate 1/2 codes together, and then just not transmitting one of the output bits we can convert this rate 1/2 implementation into a 2/3 rate code. 2 bits come and 3 go out. This concept is called puncturing. On the receiver side, dummy bits that do not affect the decoding metric are inserted in the appropriate places before decoding. This technique allows producing codes of many different rates using just a one simple hardware [1].

3.9 OTHER DECODING ALGORITHMS

Prior to the discovery of the VA, a number of other algorithms had been proposed for decoding convolutional codes. These were sequential decoding algorithm proposed by Wozen craft and subsequently modified by Fano. The Fano sequential algorithm searches for most probable path through the trellis by examining one path at a time. In this algorithm an additional negative constant is added to the each branch metric. The value of this constant is selected such that the metric of the correct path will increase on the average, while the metric of incorrect path will decrease on the average. By comparing the metric of a candidate path with an increasing threshold, Fano's algorithm detects and discards incorrect path. Its error performance is comparable to that of Viterbi decoding. In comparison with Viterbi decoding algorithm, sequential decoding has larger decoding delay. On the positive side, sequential decoding requires less storage than Viterbi decoding and hence it is more attractive for convolutional codes with large constraint length. Another type of sequential decoding algorithm is called a stack algorithm. In contrast to VA, this keeps track of (1) 2 k K− paths and corresponding metrics, the stack sequential decoding algorithm deals with fewer paths and their corresponding metrics. In

a stack algorithm, more probable paths are ordered according to their metrics, with the path at the top of the stack having largest metric. At each step of the algorithm, only the path at the top of the stack is extended by one branch. In comparison to VA, the stack algorithm requires fewer metric computations, but this computational saving is offset to a large extent by the computations involved in reordering the stack after taking into account of every iteration. A third alternative is feedback decoding. In feedback decoding, decoding delay is significantly smaller than decoding delay in Viterbi decoder.

3.10 APPLICATIONS OF CONVOLUTIONAL CODES

1. Satellite Applications

Applications in satellite communications can be more challenging in terms of high performance and high data rate requirements since both power and bandwidth impose significant limitations. The fixed INTELSAT and the mobile INMARSAT satellite systems employ mainly R = ½ and R = ¾, K = 7 convolutional codes of different data rates within 0.6-64 Kbit/s.

2. Digital Mobile Applications

Convolutional codes are extensively used in digital mobile communication systems such as GSM, TIA IS 136 (telecommunication industry association interim standard 136) and MPT (Ministry of posts and telecommunications) in cellular telephony of Europe, North America and Japan respectively. In GSM a combination of different block (cyclic) and the (2, 1, 5) convolutional codes together with interleaving techniques and puncturing are typically used, operating at data rates of 2.4-9.6 Kbits/s for data traffic channels and 5.6-13 Kbit/s for speech traffic channels. Convolutional codes have been used in many NASA (National Aeronautics and Space Administration) missions since the l960. Table 1

summarizes the characteristics of the codes used in Pioneer 9-11, Voyager and Galileo projects launched for solar orbit, Jupiter and Saturn missions and exploration of other planets. In Voyager and Galileo missions the convolutional codes shown in the table were also combined with an outer RS code in concatenation in order to improve their performance. The RS (255, 223) code was concatenated with the inner (3, 1, 7) convolutional code when Voyager was transmitting images of Pluto and Neptune to earth. The same RS code was used in concatenation with the (4, 1, 15) convolutional code employed by Galileo spacecraft.

3. Code division multiple access

CDMA being interference based use forward error correction schemes like convolutional coding to increase cell capacity. Viterbi algorithm is main building block of CDMA. CDMA uses Viterbi decoder with constraint length 9 and data rate 1/3. The generator polynomials are (557,663,711). A three bit soft decision is used [13, 14].

4. Ultra Wide Band Applications

Convolutional codes are widely used for ultra wide band applications (UWB). Viterbi decoder is used for multiple orthogonal frequency division multiplexing. UWB communication systems have widely been used for communication usage. Multiple systems have been proposed for high speed wireless personal area networks. In these systems, convolutional codes are widely selected as channel codes to correct transmission errors. MB-OFDM based system widely uses puncture convolutional codes to provide different error correcting capability and data rates. In MB-OFDM system, the input to decoder is soft information from frequency domain analyzer. To achieve different data rates and operate on 33 different distance, 1/3 convolutional codes with k=7 is punctured

to provide rate 1/2, 5/8 and 3/4 codes. Due to large variation in data rates and code rate, it is desirable to design a VD to support the highest data rate with high efficiency and reduce power consumption in lower data rates [15].

5. Voice-band data application

Convolutional codes also find applications in voice-band data communication systems such as modems used in the general switched telephone network (GSTN). Under bandwidth limited conditions (300-3400 Hz) multilevel coded modulation (MCM) techniques have been used to achieve the required performance without lowering the data rate. During the decade 1984-1994 the international telephone and telegraph consultative committee (ITTCC), later called International Telecommunication Union Telecommunication Standardization Sector (ITU-T) produced three major standards for voice-band data modems. They all featured trellis coded modulation (TCM) combined with non-linear convolutional coding schemes at transmission rates of 9.6, 14.4 and 28.8 Kbit/s respectively.

CHAPTER-4

ARCHITECTURE OF FPGAs

4.1 INTRODUCTION TO FPGAs:

The Field Programmable Gate Array or FPGA as it is more widely called is a type of programmable device. Programmable devices are a class of general-purpose chips that can be reconfigured for a wide veriety of applications. The first programmable device, which achieved a widespread use, was the PROM (Programmable Read-Only Memory).

Another step was the development of PLD (Programmable Logic Devices). These devices were constructed to implement logic circuits. The PLD includes an array AND gates connect to an array of OR gates. The PAL (Programmable Array Logic) is a commonly used PLD consisting of a programmable AND-plane followed by a fixed OR-plane. The PAL was designed for small logic circuits.

The MPGA (Mask Programmable Gate Array) was developed to handle larger logic circuits. A common MPGA consists of rows of transistors that can be interconnected to implement desired logic circuits. User specified contents are available both within the rows and between the rows. This enabled implementation of basic logic gates and the ability to interconnect the gates. In 1985, Xilinx Inc. introduced the FPGA (Field Programmable Gate Array). The Interconnects between all the elements was designed to be user programmable.

4.2 TYPES OF FPGAs

There are main four categories commercially available:

-Symmetrical

Row-Based

-Heirarchical

Sea-Of-Gates

In all these, interconnections and how they are programmed vary.

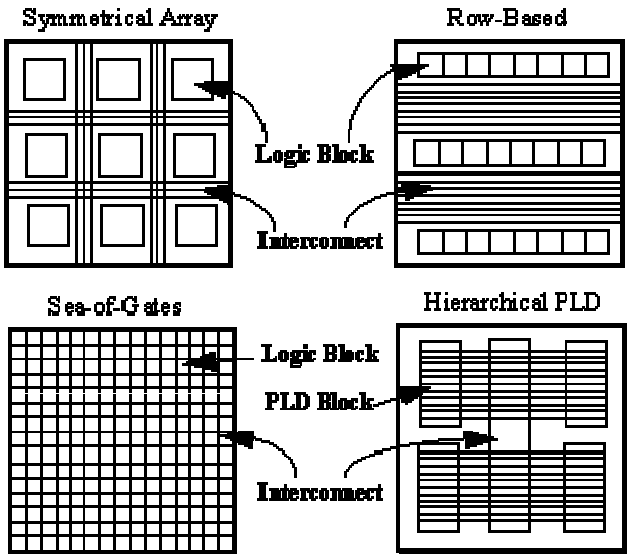


Figure 4.1 Four main categories available of FPGAs

Currently, there are four technologies used for FPGAs [17]. These are:

4.2.1 STATIC RAM TECHNOLOGY –

In the Static RAM FPGA programmable connections are made using pass-transistors, transmission gates, or multiplexers that are controlled by SRAM cells. The advantage of this technology is that it allows fast in-circuit reconfiguration. The major disadvantage is the size of the chip required by the RAM technology.

4.2.2 ANTI-FUSE TECHNOLOGY –

An anti-fuse resides in a high-impedance state; and can be programmed into low impedance or "fused" state. A less expensive than the RAM technology, this device is a Program once device

4.2.3 EPROM / EEPROM TECHNOLOGY –

This method is the same as used in the EPROM memories. One advantage of this technology is that it can be reprogrammed without external storage of configuration; though the EPROM transistors cannot be re-programmed in-circuit.

The Field-Programmable Gate Arrays (FPGAs) provide the benefits of custom CMOS VLSI, while avoiding the intial cost, time delay, and inherent risk of a conventional masked gate array. Loading configuration data into the internal memory cells customizes the FPGAs. The FPGA can either actively read its configuration data out of external serial or byte-parallel PROM (master mode), or the configuration data can be written into the FPGA (slave and peripheral mode). The FPGA can be programmed an

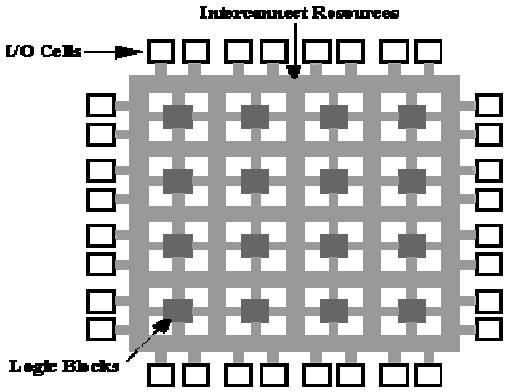


Figure 4.2: THE FPGA

unlimited number of times and supports system clock rates of up to 50 MHz.The FPGA has three major configurable elements: configurable logic blocks (CLBs), input/output blocks, and interconnects. The CLBs provide the functional elements for constructing user's logic (figure 4.2). The IOBs provide the interface between the package pins and internal signal lines. The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLBs and IOBs onto the appropriate networks. Customized configuration is established by programming internal static memory cells that determine the logic functions and internal connections implemented in the FPGA.

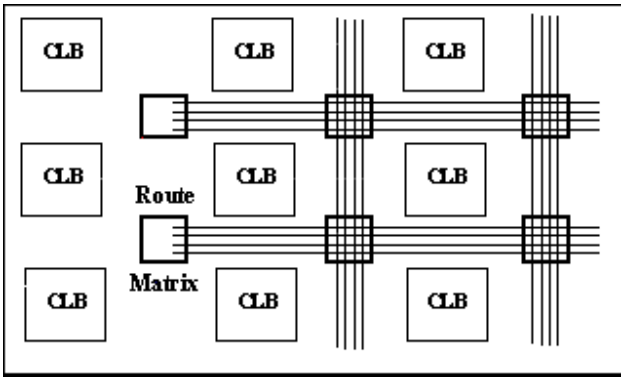


Figure 4.3: CLBs INTERCONNECTS

Figure 4.3 depicts a FPGA with a two-dimensional array of logic blocks that can be interconnected by interconnect wires. All internal connections are composed of metal segments with programmable switching points to implement the desired routing. An abundance of different routing resources is provided to acheive efficient automated routing. There are four main types of interconnect, the relative length of their segments distinguishes three: single-length lines, double-length lines and Longlines. (NOTE: The

number of routing channels shown in the figure are for illustration purposes only; the actual number of routing channels varies with the array size.) In addition, eight global buffers drive fast, low-skew nets most often used for clocks or global control signals.

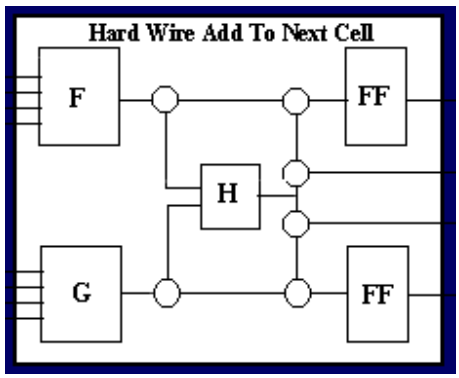


Figure 4.4: CONFIGURABLE LOGIC BLOCK

The principle CLB (Configurable Logic Block) elements are shown in Figure 4.4. Each CLB contains a pair of flip-flops and two independent 4-input function generators. These function generators have a good deal of flexabilty as most combinatorial logic functions need less than four inputs. Configurable Logic Blocks implement most of the logic in an FPGA. The flexabilty and symmetry of the CLB architecture facilitates the placement and routing of a given application.

4.3 VIRTEX-II FPGA: The Platform FPGA for Programmable Logic - is the highest density and highest performance FPGA in the world. The Virtex-II series of FPGAs are the latest FPGAs manufactured by Xilinx.

The system gate density ranges from 40K(XC2V40) to 8M(XC2V8000) gates. This gate density is unrivaled by any other manufacturer. The Virtex-II is a programmable device comprised of input/output blocks (IOBs) and internal configurable logic blocks (CLBs). Virtex-II devices are user-programmable gate arrays with various configurable elements. The Virtex-II architecture is optimized for high-density and high-performance logic designs. As shown in Figure 4.1, the programmable device is comprised of input/output blocks (IOBs) and internal configurable logic blocks (CLBs).

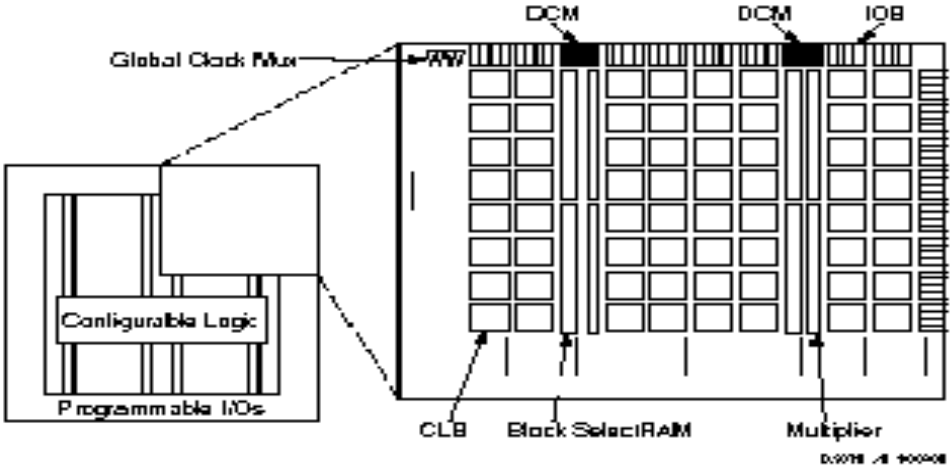


Figure 4.5 : Virtex-11 Architecture an overview

The internal configurable logic includes four major elements organized in a regular array.

• Configurable Logic Blocks (CLBs) provide functional elements for combinatorial and synchronous logic, including basic storage elements. BUFTs (3-state buffers) associated with each CLB element drive dedicated segmentable horizontal routing resources.

• Block SelectRAM memory modules provide large 18 Kbit storage elements of dual-port RAM.

• Multiplier blocks are 18-bit x 18-bit dedicated multipliers.

• DCM (Digital Clock Manager) blocks provide self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division, coarse- and fine-grained clock phase shifting.

A new generation of programmable routing resources called Active Interconnect Technology interconnects all of these elements. The general routing matrix (GRM) is an array of routing switches. Each programmable element is tied to a switch matrix, allowing multiple connections to the general routing matrix. The overall programmable interconnection is hierarchical and designed to support high-speed designs.

All programmable elements, including the routing resources, are controlled by values stored in static memory cells. These values are loaded in the memory cells during configuration and can be reloaded to change the functions of the programmable elements.

4.4 VIRTEX-II FEATURES

This section briefly describes Virtex-II features [17].

4.4.1 INPUT/OUTPUT BLOCKS (IOBS)

IOBs are programmable and can be categorized as follows:

• Input block with an optional single-data-rate or double-data-rate (DDR) register

• Output block with an optional single-data-rate or DDR register, and an optional 3-state buffer, to be driven directly or through a single or DDR register

• Bidirectional block (any combination of input and output configurations). These registers are either edge-triggered D-type flip-flops or level-sensitive latches. IOBs support the following single-ended I/O standards:

• LVTTL, LVCMOS (3.3V, 2.5V, 1.8V, and 1.5V)

• PCI-X compatible (133 MHz and 66 MHz) at 3.3V

• PCI compliant (66 MHz and 33 MHz) at 3.3V

• CardBus compliant (33 MHz) at 3.3V

• GTL and GTLP

• HSTL (Class I, II, III, and IV)

• SSTL (3.3V and 2.5V, Class I and II)

• AGP-2X

The digitally controlled impedance (DCI) I/O feature automatically provides on-chip termination for each I/O element. The IOB elements also support the following differential signaling I/O standards:

• LVDS

• BLVDS (Bus LVDS)

• ULVDS

• LDT

• LVPECL

Two adjacent pads are used for each differential pair. Two or four IOB blocks connect to one switch matrix to access the routing resources.

4.4.2 CONFIGURABLE LOGIC BLOCKS (CLBS)

CLB resources include four slices and two 3-state buffers. Each slice is equivalent and contains:

• Two function generators (F & G)

• Two storage elements

• Arithmetic logic gates

• Large multiplexers

• Wide function capability

• Fast carry look-ahead chain

• Horizontal cascade chain (OR gate)

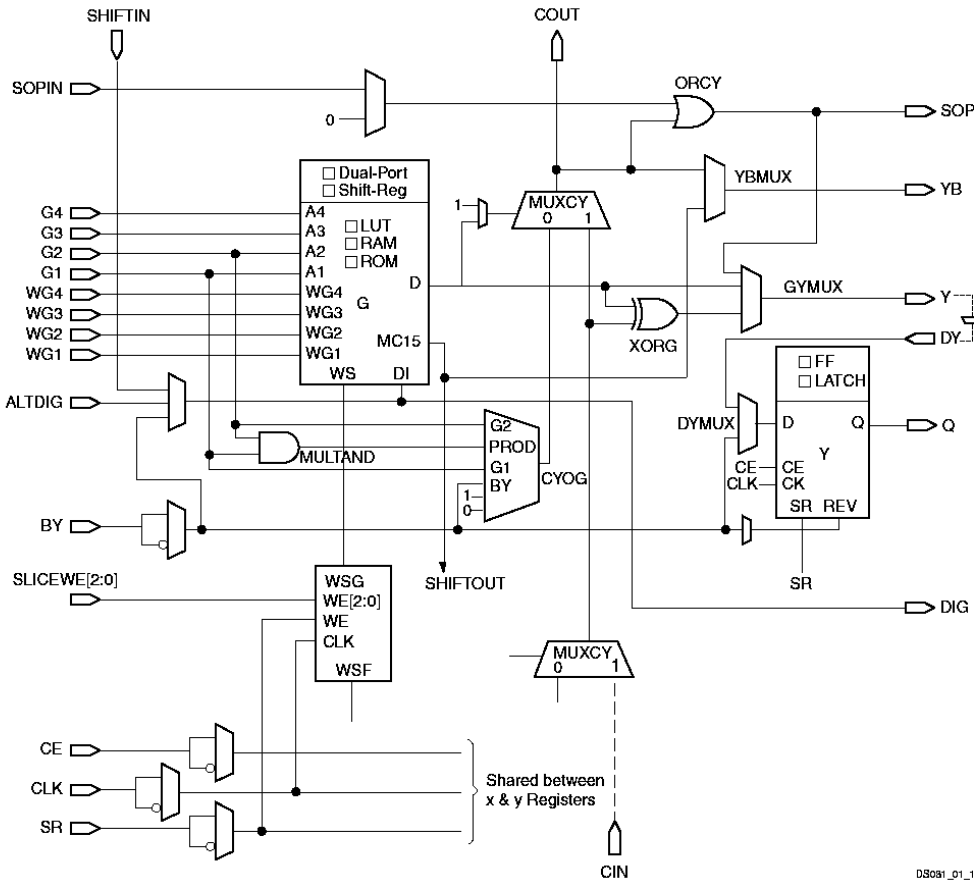


Figure 4.6: Virtex-II Pro Slice (Top Half)

The function generators F & G are configurable as 4-input look-up tables (LUTs), as 16-

bit shift registers, or as 16-bit distributed Select RAM memory. In addition, the two storage elements are either edge-triggered D-type flip-flops or level-sensitive latches. Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources.

4.4.3 BLOCK SELECT RAM MEMORY

The block Select RAM memory resources are 18 Kb of dual-port RAM, programmable from 16K x 1 bit to 512 x 36 bits, in various depth and width configurations. Each port is totally synchronous and independent, offering three "read-during-write" modes. Block Select RAM memory is cascadable to implement large embedded storage blocks.

A multiplier block is associated with each Select RAM memory block. The multiplier block is a dedicated 18 x 18-bit multiplier and is optimized for operations based on the block Select RAM content on one port. The 18 x 18 multiplier can be used independently of the block SelectRAM resource. Read/multiply/accumulate operations and DSP filter structures are extremely efficient. Both the Select RAM memory and the multiplier resource are connected to four switch matrices to access the general routing resources.

4.4.4 GLOBAL CLOCKING

The DCM and global clock multiplexer buffers provide a complete solution for designing high-speed clocking schemes. Up to 12 DCM blocks are available. To generate de-skewed internal or external clocks, each DCM can be used to eliminate clock distribution delay. The DCM also provides 90-, 180-, and 270-degree phase-shifted versions of its output clocks. Fine-grained phase shifting offers high-resolution phase adjustments in increments of 1/256 of the clock period. Very flexible frequency synthesis provides a

clock output frequency equal to any M/D ratio of the input clock frequency, where M and D are two integers. Virtex-II devices have 16 global clock MUX buffers, with up to eight clock nets per quadrant. Each global clock MUX buffer can select one of the two clock inputs and switch glitch-free from one clock to the other. Each DCM block is able to drive up to four of the 16 global clock MUX buffers.

4.4.5 ROUTING RESOURCES

The IOB, CLB, block SelectRAM, multiplier, and DCM elements all use the same interconnect scheme and the same access to the global routing matrix. Timing models are shared, greatly improving the predictability of the performance of high-speed designs. There are a total of 16 global clock lines, with eight available per quadrant. In addition, 24 vertical and horizontal long lines per row or column as well as massive secondary and local routing resources provide fast interconnect. Virtex-II buffered interconnects are relatively unaffected by net fanout and the interconnect layout is designed to minimize crosstalk.

Horizontal and vertical routing resources for each row or column include:

• 24 long lines

• 120 hex lines

• 40 double lines

• 16 direct connect lines (total in all four directions)

4.4.6 BOUNDARY SCAN

Boundary scan instructions and associated data registers support a standard methodology for accessing and config-uring Virtex-II devices that complies with IEEE standards

1149.1 - 1993 and 1532. A system mode and a test mode are implemented. In system mode, a Virtex-II device per-forms its intended mission even while executing non-test boundary-scan instructions. In test mode, boundary-scan test instructions control the I/O pins for testing purposes. The Virtex-II Test Access Port (TAP) supports BYPASS, PRELOAD, SAMPLE, IDCODE, and USERCODE non-test instructions. The EXTEST, INTEST, and HIGHZ test instruc-tions are also supported.

4.4.7 CONFIGURATION

Virtex-II devices are configured by loading data into internal configuration memory, using the following five modes:

- Slave-serial mode
- Master-serial mode
- Slave SelectMAP mode
- Master SelectMAP mode
- Boundary-Scan mode (IEEE 1532)

A Data Encryption Standard (DES) decryptor is available on-chip to secure the bitstreams. One or two triple-DES key sets can be used to optionally encrypt the configuration information.

4.4.8 READBACK AND INTEGRATED LOGIC ANALYZER

Configuration data stored in Virtex-II configuration memory can be read back for verification. Along with the configuration data, the contents of all flip-flops/latches, distributed SelectRAM, and block SelectRAM memory resources can be read back. This capability is useful for real-time debugging. The Integrated Logic Analyzer (ILA) core

and software[18] provides a complete solution for accessing and verifying Virtex-II devices.

Contains two major programmable parts:

-Configurable Logic Blocks (CLBs)

-Input/Output Blocks (IOBs)

CLBs interconnect through a general routing matrix (GRM). The GRM comprises an array of routing switches located at the intersections of horizontal and vertical routing channels. Each CLB nests into a VersaBlock™ that also provides local routing resources to connect the CLB to the GRM.

Figure 4.7 describes with flow chart the complete FPGA design cycle.

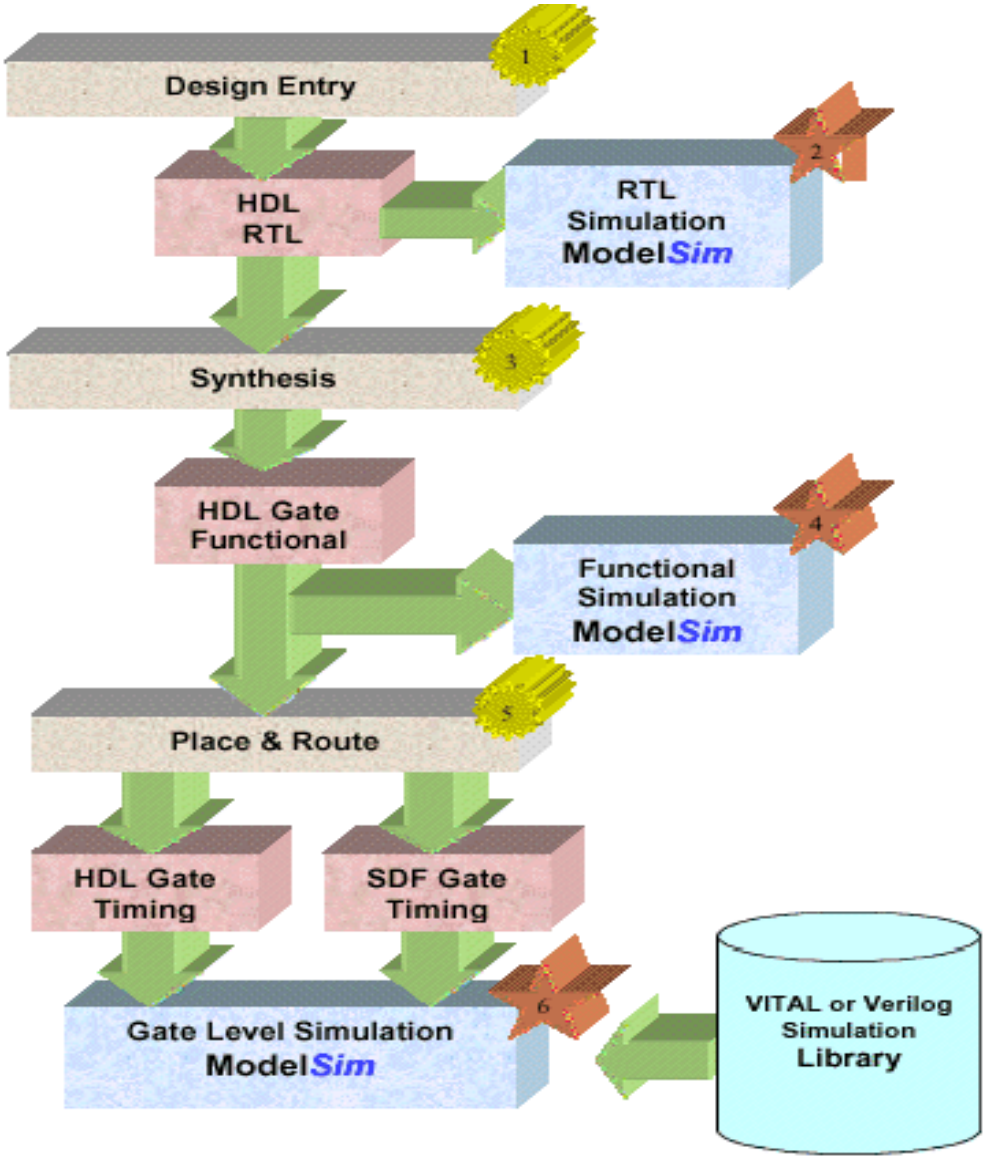


Figure 4.7 :Flow chart for FPGA Design cycle

CHAPTER-5

IMPLEMENTATION OF VITERBI ALGORITHM ON FPGA

5.1 DECODER IMPLEMENTATION

In the context of the trellis diagram of Figure 3(k), transitions during anyone time interval can be grouped into 2^{v-1} disjoint cells, each cell depicting four possible transitions, where v = K -1 is called the encoder memory. For the K = 3 example, v = 2 and $2^{v-1} = 2$ cells. These cells are shown in Figure 5.1, where a, b, c, and d refer to the states at time t_i, and a', b', c', and d' refer to the states at time t_{i+1} Shown on each transition is the branch metric δ_{xy}, where the subscript indicates that the metric corresponds to the transition from state x to state y. These cells and the associated logic

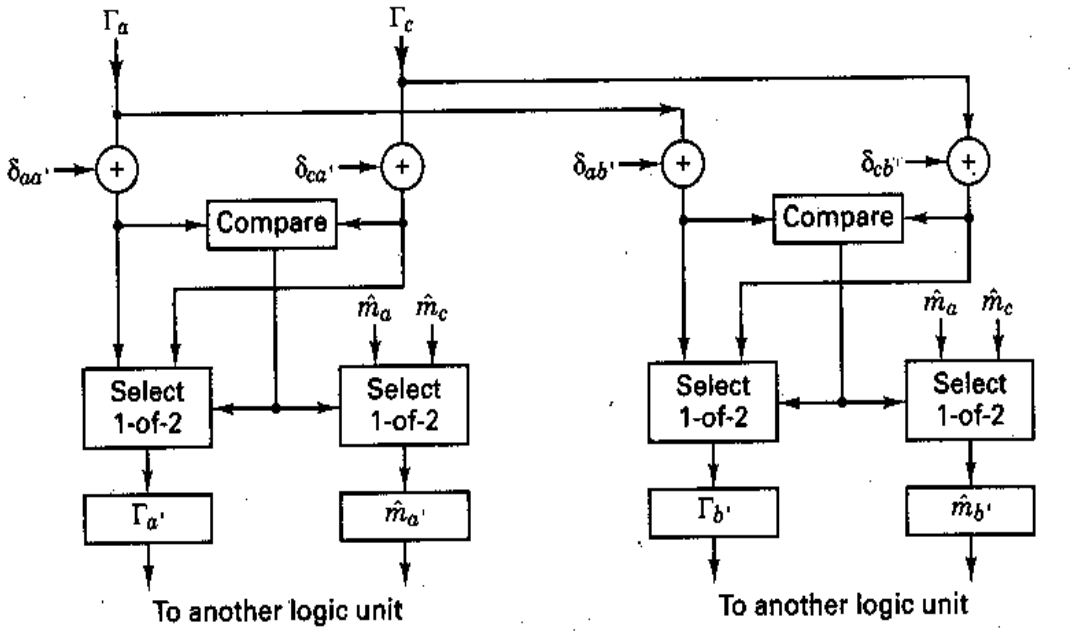


Figure 5.1 Logic unit that implements the add-compare-select functions corresponding to cell # 1

units that update the state metrics Γ_x, where x designates a particular state, represent the basic building blocks of the decoder.

5.1.1 ADD-COMPARE-SELECT COMPUTATION

Continuing with the K = 3, 2-cell example, Figure 5.1 illustrates the logic unit that corresponds to cell 1. The logic executes the special purpose computation called add-compare-select (ACS). The state metric $\Gamma_{a'}$ is calculated by adding the previous-time state metric of state a, Γ_a to the branch metric $\delta_{aa'}$ and the previous time state metric of state c, r c, to the branch metric $\delta_{ca'}$. This results in two possible path metrics as candidates for the new state metric $\Gamma_{a'}$. The two candidates are compared in the logic unit of Figure 5.1. The largest likelihood (smallest distance) of the two path metrics is stored as the new state metric $\Gamma_{a'}$ for state a. Also stored is the new path history $\hat{m}_a$ for state a, where $\hat{m}_a$ is the message-path history of the state augmented by the data of the winning path. Also shown in Figure 5.1 is the cell-l ACS logic that yields the new state metric $\Gamma_{b'}$ and the new path history $\hat{m}_{b'}$ This ACS operation is similarly performed for the paths in other cells. The oldest bit on the path with the smallest state metric forms the decoder output.

5.1.2 ADD-COMPARE-SELECT AS SEEN ON THE TRELLIS

The message sequence was m = 11 0 11, the codeword sequence was U = 11 01 0100 01, and the received sequence was b= 11 01 011001. Figure 5.2 depicts a decoding trellis diagram similar to Figure 3.11. A branch metric that labels each branch is the Hamming distance between the received code symbols and the corresponding branch word from the encoder trellis. Additionally, the Figure 5.2 trellis indicates a value at each state x, and for each time from time t_2 to t_6, which is a state metric Γ_x .We perform the add-compare-select (ACS) operation when there are two transitions entering a state, as there are for times t_4 and later. For example at time t_4, the value of the state metric for state a is obtained by incrementing the state metric Γ a = 3 at time t_3 with the branch metric $\delta_{aa'}$ = 1 yielding a candidate value of 4. Simultaneously, the state metric Γ_c = 2 at time t_3 is incremented with the branch metric $\delta_{ca'}$ == 1 yielding a candidate value of 3. The select operation of the ACS process selects the largest-likelihood (minimum distance) path metric as the new state metric; hence, for state a at time t_4, the new state metric is $\Gamma_{a'}$ = 3.

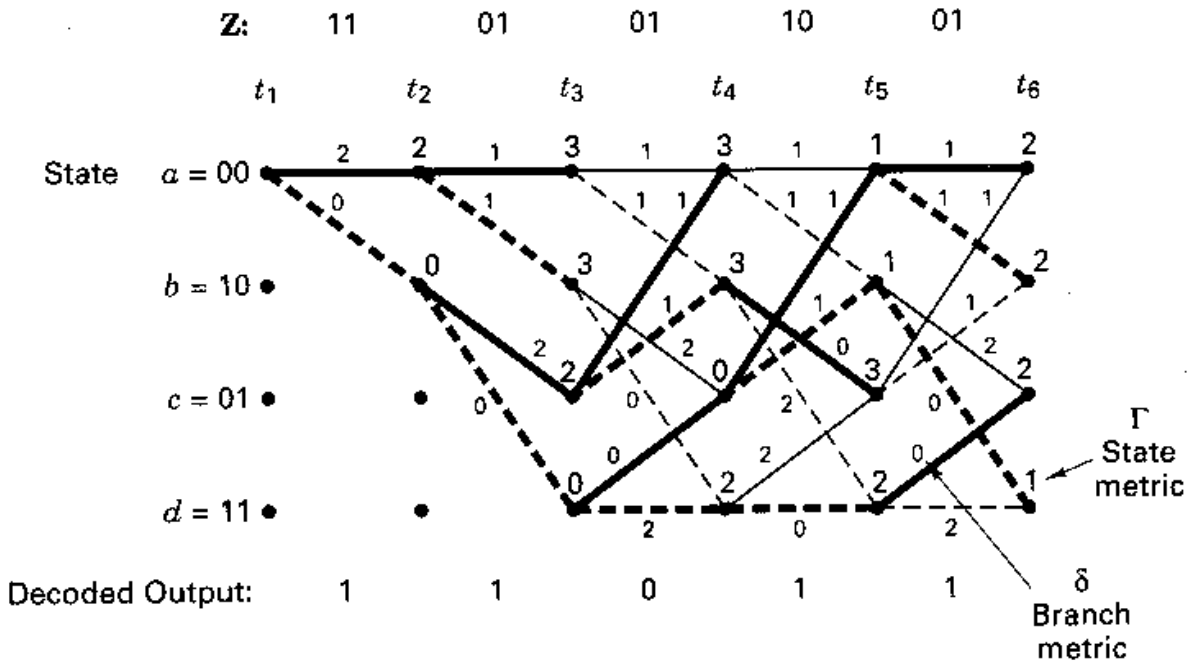


Figure 5.2 Add-compare-select computations in Viterbi decoding

The winning path is shown with a heavy line and the path that has been dropped is shown with a lighter line. On the trellis of Figure 5.2, observe the state metrics from left to right. Verify that at each time, the value of each state metric is obtained by incrementing the connected state metric from the previous time along the winning path (heavy line) with the branch metric between them. At some point in the trellis (after a time interval of 4 or 5 times the constraint length), the oldest bits can be decoded. As an example, looking at time t_6 in Figure 5.2, we see that the minimum-distance state metric has a value of 1. From this state d, the winning path can be traced back to time t_1, and one can verify that the decoded message is the same as the original message, by the convention that dashed and solid lines represent binary ones and zeros respectively

5.2 PATH MEMORY.AND SYNCHRONIZATION

The storage requirements of the Viterbi decoder grow exponentially with constraint length K. For a code with rate l/n, the decoder retains a set of 2^{K-1} paths after each decoding step. With high probability, these paths will not be mutually disjoint very far

back from the present decoding depth .All of the 2^{K-1} paths tend to have a common stem which eventually branches to the various states. Thus if the decoder stores enough of the history of the 2^{K-1} paths, the oldest bits on all paths will be the same. A simple decoder implementation, then, contains a fixed amount of path history and outputs the oldest bit on an arbitrary path each time it steps one level deeper into the trellis. The amount of path storage required is

$$u = h\, 2^{K-1} \qquad (5.1)$$

where h is the length of the information bit path history per state. A refinement, which minimizes the value of h, uses the oldest bit on the most likely path as the decoder output, instead of the oldest bit on an arbitrary path. It has been demonstrated that a value of h of 4 or 5 times the code constraint length is sufficient for near-optimum decoder performance. The storage requirement u is the basic limitation on the implementation of Viterbi decoders. Commercial decoders are limited to a constraint length of about K = 10. Efforts to increase coding gain by further increasing constraint length are met by the exponential increase in memory requirements (and complexity) that follows from Equation (5.1). Branch word synchronization is the process of determining the beginning of a branch word in the received sequence. Such synchronization can take place without new information being added to the transmitted symbol stream because the received data appear to have an excessive error rate when not synchronized. Therefore, a simple way of accomplishing synchronization is to monitor some concomitant indication of this large error rate, that is, the rate at which the state metrics are increasing or the rate at which the surviving paths in the trellis merge. The monitored parameters are compared to a threshold, and synchronization is then adjusted accordingly.

5.3 VERILOG HDL CODE FOR DIFFERENT VITERBI DECODER MODULES

```
module acs_enable (clk,res,res_3);

  input clk,res;
  output res_3;

  wire res_1,res_2;

  dff dff_0(res,res_1,clk,res);
  dff dff_1(res_1,res_2,clk,res);
  dff dff_2(res_2,res_3,clk,res);

endmodule

module back(state,survival_data,out);

  input [1:0] state;
  input [3:0] survival_data;

  output [1:0] out;

  assign out [0] = survival_data [state];
  assign out [1] = state [0];

endmodule

module compare_select
       (p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3,rst_3,
        out0,out1,out2,out3,
        ACS);

  input [3:0] p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3;
  input rst_3;
  output [2:0] out0,out1,out2,out3;
  output [3:0] ACS;
```

```
  function [2:0] find_min_metric;
   input [3:0] a,b;
   input enabled;
    begin
      if (enabled == 0) find_min_metric=a;
      else if (a<=b) find_min_metric=a;
          else find_min_metric=b;

  end
  endfunction

  function set_control;
    input [2:0] a,b;
    input enabled;
    begin
      if (enabled == 0) set_control=0;
      else if (a<=b) set_control=0;
          else set_control=1;
    end
  endfunction

  assign out0=find_min_metric(p0_0,p1_0,rst_3);
  assign out1=find_min_metric(p2_1,p3_1,rst_3);
  assign out2=find_min_metric(p0_2,p1_2,rst_3);
  assign out3=find_min_metric(p2_3,p3_3,rst_3);

  assign ACS[0] = set_control (p0_0,p1_0,rst_3);
  assign ACS[1] = set_control (p2_1,p3_1,rst_3);
  assign ACS[2] = set_control (p0_2,p1_2,rst_3);
  assign ACS[3] = set_control (p2_3,p3_3,rst_3);

endmodule

module compute_metric
       (m_out0,m_out1,m_out2,m_out3,
        s0,s1,s2,s3,
        p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3);

    input [2:0] m_out0,m_out1,m_out2,m_out3;
    input [1:0] s0,s1,s2,s3;

    output [3:0] p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3;


     assign p0_0 = m_out0 + s0,
          p0_2 = m_out0 + s3,
```

```
            p2_3 = m_out2 + s1,
            p2_1 = m_out2 + s2,

            p1_0 = m_out1 + s3,
            p1_2 = m_out1 + s0,


            p3_1 = m_out3 + s1,
            p3_3 = m_out3 + s2;

endmodule


module dff(D,Q,Clock,Reset); // N.B. reset is active-low
  output Q;
  input D,Clock,Reset;

  parameter CARDINALITY = 1;
  reg [CARDINALITY-1:0] Q;
  wire [CARDINALITY-1:0] D;

    always @(posedge Clock) if (Reset!==0) #1 Q=D;
endmodule


module metric (m_in0,m_in1,m_in2,m_in3,
            m_out0,m_out1,m_out2,m_out3,
            clk,reset);

  input [2:0] m_in0,m_in1,m_in2,m_in3;
  output [2:0] m_out0,m_out1,m_out2,m_out3;

  input clk,reset;

     dff #(3) metric3(m_in3, m_out3, clk, reset);
     dff #(3) metric2(m_in2, m_out2, clk, reset);
     dff #(3) metric1(m_in1, m_out1, clk, reset);
     dff #(3) metric0(m_in0, m_out0, clk, reset);

endmodule


module path(in,out,clk,reset);

  input [3:0] in;
```

```
  input clk,reset;
  output [3:0] out;

  wire [3:0] p_in;

  dff #(4) path0(p_in,out,clk,reset);


assign p_in = in;

endmodule

module path_memory (ACS,control,clk,reset,
                    decode_out);

  input [3:0] ACS;
  input [1:0] control;
  input clk,reset;
  output decode_out;

  wire [3:0] out1,out2,out3,out4,out5,out6,out7,out8,out9,out10,out11;
  wire [1:0] state1,state2,state3,state4,state5,state6,state7,state8,state9,state10,state11;
  wire [1:0] last_state;

     path x1 (ACS , out1 ,clk,reset),
         x2 (out1, out2 ,clk,reset),
         x3 (out2, out3 ,clk,reset),
         x4 (out3, out4 ,clk,reset),
         x5 (out4, out5 ,clk,reset),
         x6 (out5, out6 ,clk,reset),
         x7 (out6, out7 ,clk,reset),
         x8 (out7, out8 ,clk,reset),
         x9 (out8, out9 ,clk,reset),
         x10(out9, out10,clk,reset),
         x11(out10,out11,clk,reset);


     back b11 (control,out1 ,state1),
         b10 (state1 ,out2 ,state2),
         b9  (state2 ,out3 ,state3),
         b8  (state3 ,out4 ,state4),
         b7  (state4 ,out5 ,state5),
         b6  (state5 ,out6 ,state6),
         b5  (state6 ,out7 ,state7),
         b4  (state7 ,out8 ,state8),
         b3  (state8 ,out9 ,state9),
```

```
        b2  (state9 ,out10,state10),
        b1  (state10,out11,state11);

   dff #(2) buff (state11,last_state,clk,reset);

   assign decode_out = last_state;


endmodule

module reduce
     (input0,input1,input2,input3,
     m_in0,m_in1,m_in2,m_in3,
     control);

  input [2:0] input0,input1,input2,input3;
  output [2:0] m_in0,m_in1,m_in2,m_in3;
  output [1:0] control;

  wire [2:0] smallest;
  reg [1:0] control;

  function [2:0] find_smallest;
    input [2:0] input0,input1,input2,input3;
    reg [2:0] a,b;
      begin
        if(input0<=input1) a=input0; else a=input1;
        if(input2<=input3) b=input2; else b=input3;
        if(a<=b) find_smallest = a;
        else find_smallest = b;
      end
  endfunction
  always @(smallest)
   begin
    case (smallest)
      input0 : control = 0;
      input1 : control = 1;
      input2 : control = 2;
      input3 : control = 3;
    endcase
   end

  assign smallest = find_smallest(input0,input1,input2,input3);

  assign m_in0 = input0 - smallest;
  assign m_in1 = input1 - smallest;
```

```
  assign m_in2 = input2 - smallest;
  assign m_in3 = input3 - smallest;

endmodule

module viterbi (in0,in1,in2,in3,decode_out,clk,reset);


  input [1:0] in0,in1,in2,in3;
  output decode_out;
  input clk,reset;

  wire [2:0] m_in0,m_in1,m_in2,m_in3;
  wire [2:0] m_out0,m_out1,m_out2,m_out3;
  wire [3:0] p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3;
  wire [3:0] ACS;
  wire [2:0] out0,out1,out2,out3;
  wire [1:0] control;

  wire res_3;

    compute_metric u1(m_out0,m_out1,m_out2,m_out3,in0,in1,in2,in3,
              p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3);

    metric u2(m_in0,m_in1,m_in2,m_in3,m_out0,m_out1,m_out2,m_out3,clk,reset);

    acs_enable u3 (clk, reset, res_3);

    compare_select u4(p0_0,p1_0,p2_1,p3_1,p0_2,p1_2,p2_3,p3_3,res_3,
                out0,out1,out2,out3,ACS);

    reduce u5(out0,out1,out2,out3,m_in0,m_in1,m_in2,m_in3,control);

    path_memory u6(ACS,control,clk,reset,
              decode_out);

endmodule

module viterbi_distances  (Y_in_1,Y_in_0,in0,in1,in2,in3);

  input Y_in_1,Y_in_0;
  output [1:0] in0,in1,in2,in3;

  reg [7:0] in;

  always @(Y_in_1 or Y_in_0)
```

```
    begin
     case (Y_in_1)
       0 : case (Y_in_0)
            0 : in = 'b00010110;
            1 : in = 'b01001001;
           endcase


            1 : case (Y_in_0)
            0 : in = 'b01100001;
            1 : in = 'b10010100;
           endcase
      endcase
   end

   assign in0 = in [7:6];
   assign in1 = in [5:4];
   assign in2 = in [3:2];
   assign in3 = in [1:0];

endmodule

module viterbi_testbench;

   wire decode_out;              // decoder out
   reg [1:0] Y;                  // received signal

   reg Clk, Res;

   wire [1:0] in0,in1,in2,in3;

   initial
     begin
       #4000 $finish;
     end

   initial
     begin
       Clk=0;
       #70 Res=0;
       #10 Res=1;
     end                         // hit reset after inputs are stable

   always #50 Clk=~Clk;

// 1. What is inputted to the test bench come from the Viterbi Encoder for
```

```
//    K = 3, rate = 1/2 (111,101)
// 2. For the example below, the original data is : 010111001010001 + 00 (2 tail bits)
//    The encoder output is : 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 + 10 11
// 3. We may manually introduce error(s) into decoder input by altering the data(s) below.



  initial
    begin

//          input      |     input
//      with error (s) | without error(s)
//   ------------------|------------------

      #51 Y = 0;            // 0
      #100 Y = 2;           // 3 e
      #100 Y = 2;           // 2
      #100 Y = 0;           // 0
      #100 Y = 3;           // 1 e
      #100 Y = 2;           // 2
      #100 Y = 1;           // 1
      #100 Y = 3;           // 3
      #100 Y = 1;           // 3 e
      #100 Y = 2;           // 2
      #100 Y = 0;           // 0
      #100 Y = 3;           // 2 e
      #100 Y = 3;           // 3
      #100 Y = 0;           // 0
      #100 Y = 1;           // 3 e
      #100 Y = 2;           // 2
      #100 Y = 3;           // 3
      #100 Y = 0;
    end


  viterbi_distances v_1 (Y[1],Y[0],in0,in1,in2,in3);
  viterbi v_2 (in0,in1,in2,in3,decode_out,Clk,Res);

endmodule
```

CHAPTER-6

RESULTS AND DISCUSSIONS

Viterbi test Bench is created using Xilinx Webpack. Code is written in Verilog HDL. There are different Modules for the code and that are Viterbi test bench, Viterbi, Viterbi distance, reduce, path, path memory, compute metric, dff, acs enable, back and compare select. Verilog Code for each module is given in chapter 5^{th}. Each module performs function as discribed by example in chapter 3^{rd}.

To prove the correctness of our design, the verilog HDL description was simulated & tested using Model Sim Verilog Simulator. Afterwards Xilinx Webpack is used for design entry, sythesis, place & route and floor plan design.

What is inputted to the test bench come from the Viterbi Encoder for K=3, rate= ½ (111,101). For this Example the original data is: 010111001010001 + 00 (2 tail bits). The Encoder output is: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 + 10 11.

We may manually introduce error (s) into decoder input by altering the data (s).

Device selected is 2v40fg256-6.

Number of Slices:	86	out	of	256	33%
Number of Slice Flip Flops:	62	out	of	512	12%
Number of 4 input LUT's :	131	out	of	512	25%
Number of bonded IOB's:	10	out	of	88	11%
Number of GCLK's:	1	out	of	16	6%

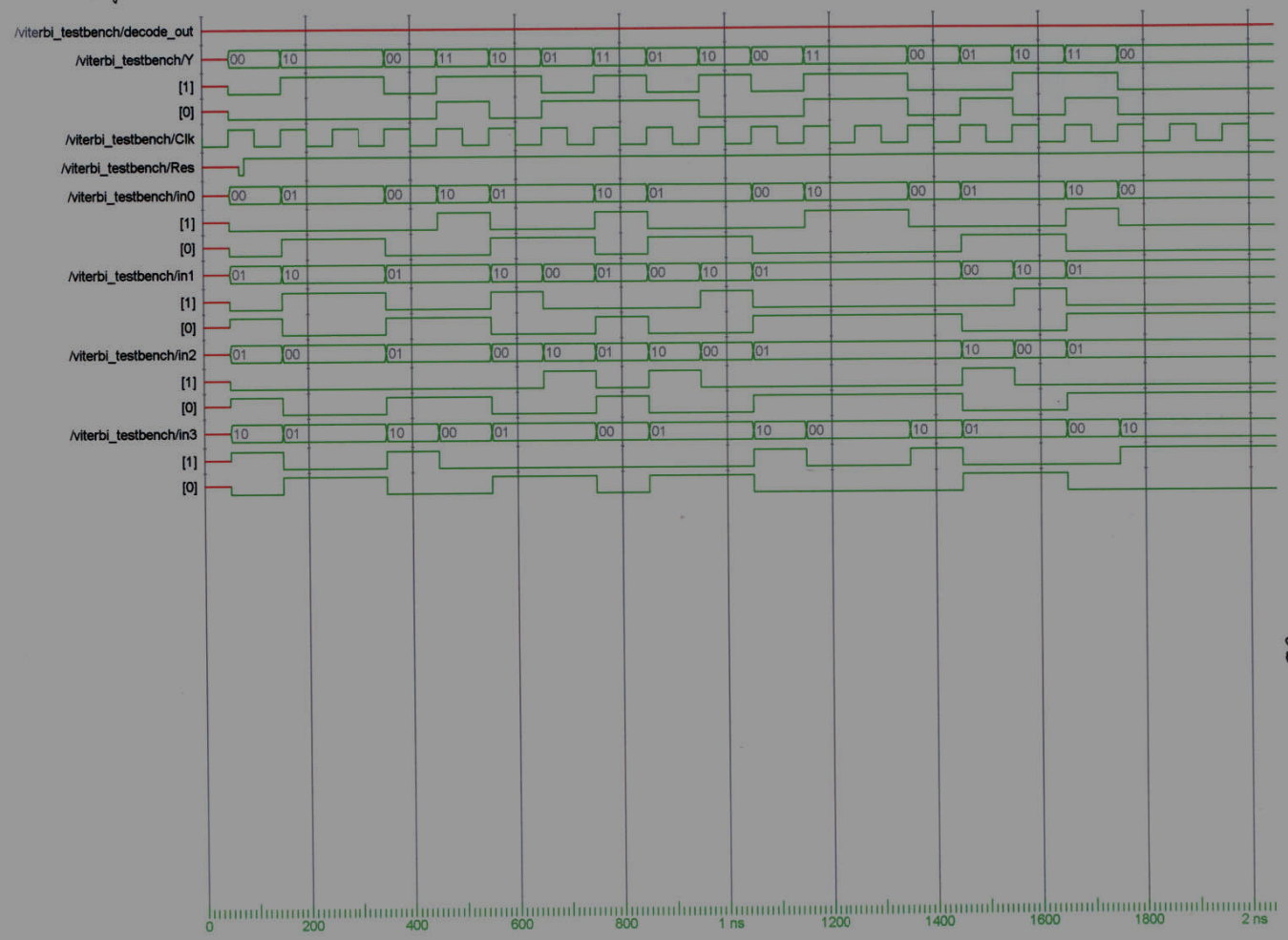
Floor plan of the synthesized FPGA XCV chip is shown in figure 6.1

Figure 6.2 describes simulation waveforms of viterbi decoder.In this all values displayed are in nano seconds (ns). In this Timing numbers are only a synthesis estimate and trace report generated after place & route gives accurate timing information.

Figure 6.3 gives Symbol of a Viterbi Decoder.From the figure it is clear that in this viterbi decoder there are six inputs & one output. The inputs are in0, in1, in2, in3, clk & reset & the output is decode out.

Figure 6.4 gives RTL Schematic of Viterbi Decoder.

Fig. 6(b) Simulation Waveforms of Viterbi Decoder



89

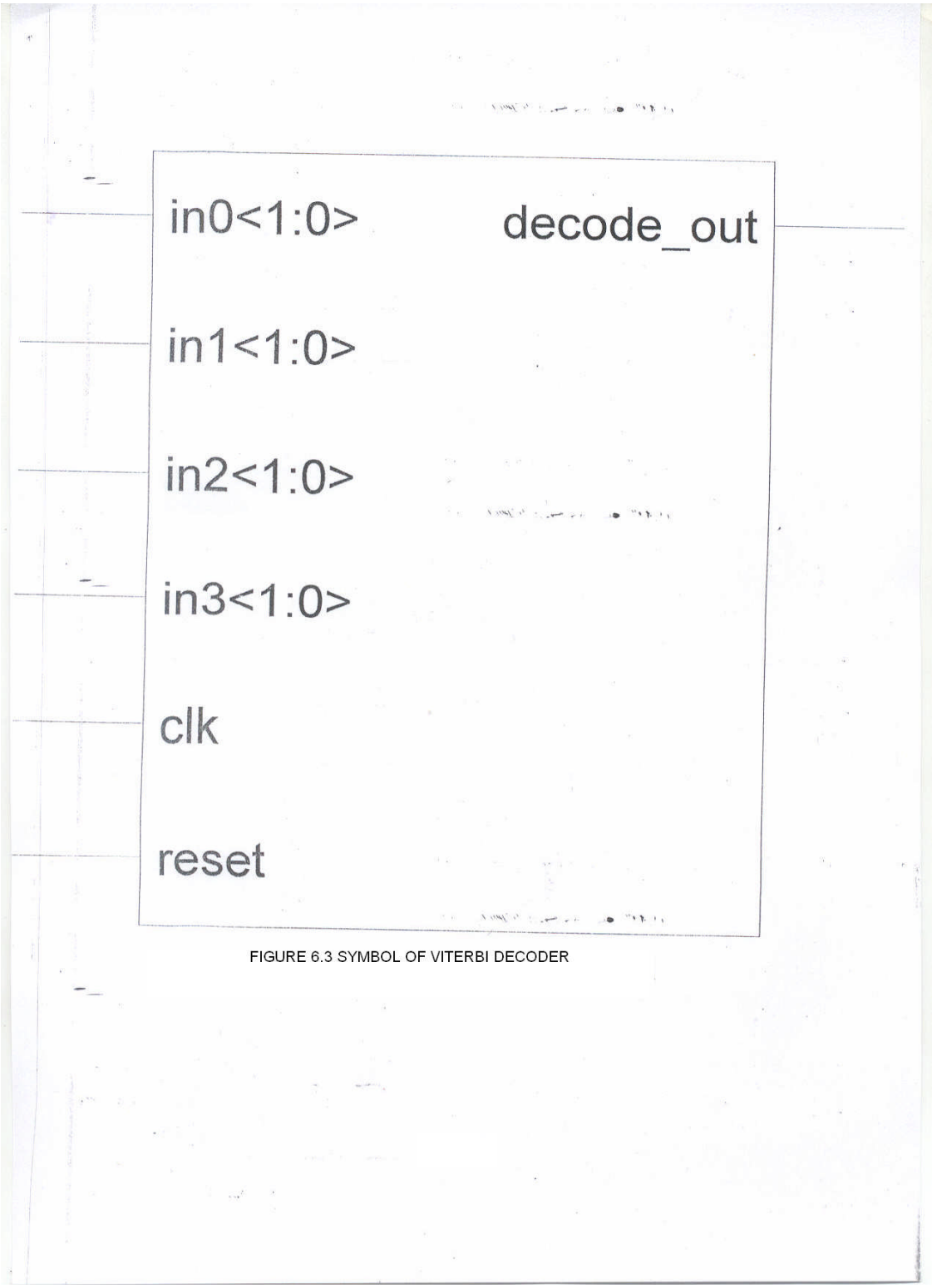


FIGURE 6.3 SYMBOL OF VITERBI DECODER

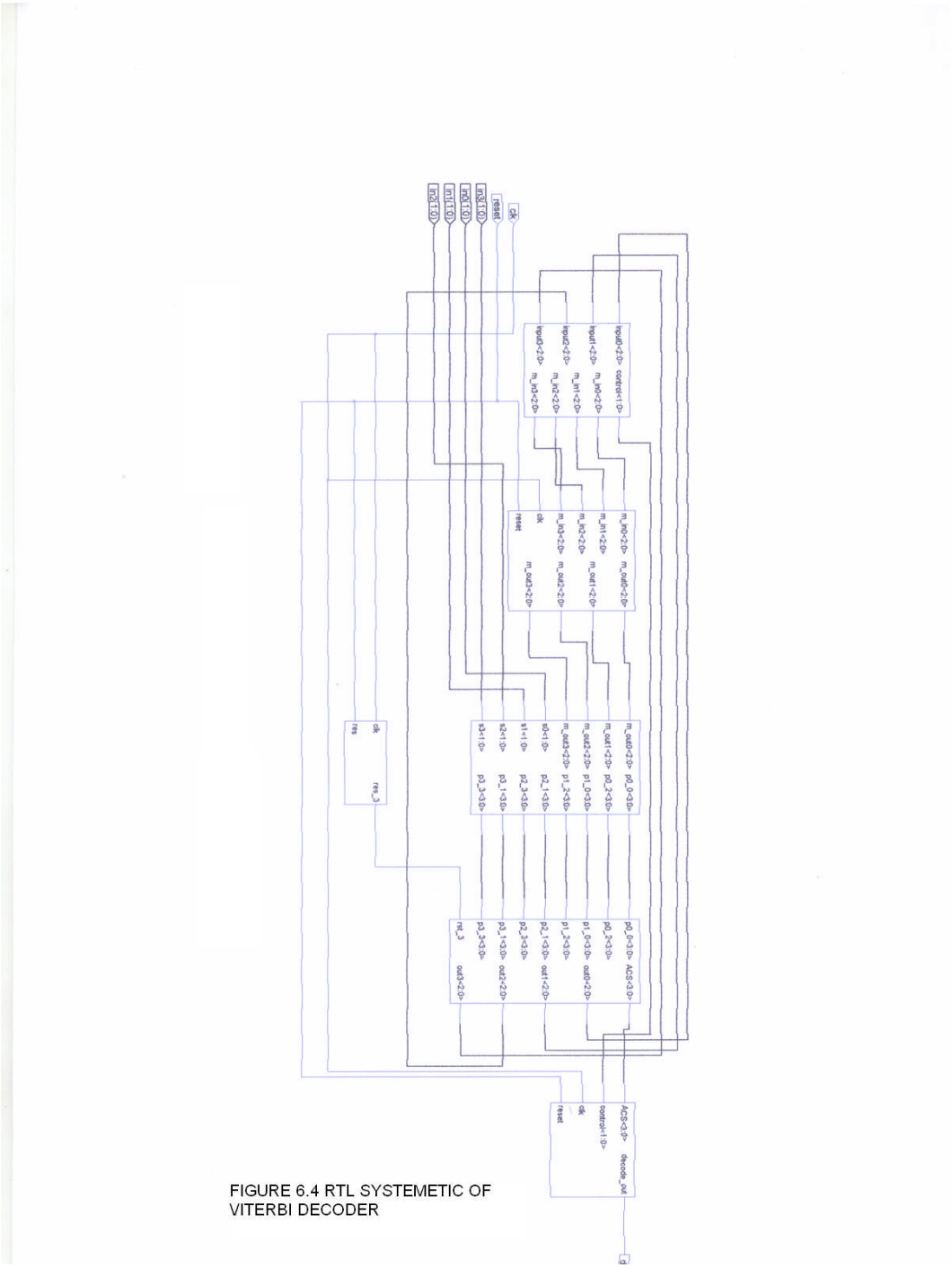


FIGURE 6.4 RTL SYSTEMETIC OF VITERBI DECODER

CHIP UTILIZATION REPORT

* Final Report *

==========

Final Results

RTL Top Level Output File Name : viterbi.ngr

Top Level Output File Name : viterbi

Output Format : NGC

Optimization Criterion : Speed

Keep Hierarchy : NO

Macro Generator : macro+

Design Statistics

IOs : 11

Macro Statistics :

Registers : 19

1-bit register : 3

2-bit register : 1

3-bit register : 4

4-bit register : 11

Multiplexers : 11

1-bit 4-to-1 multiplexer : 11

Adders/Subtractors : 12

3-bit adder carry out : 8

3-bit subtractor : 4

Comparators : 15

3-bit comparator equal : 4

3-bit comparator lessequal : 7

4-bit comparator lessequal : 4

Cell Usage :

BELS : 206

GND : 1

LUT1 : 1

LUT1_L : 8

LUT2 : 9

LUT2_D : 8

LUT2_L : 12

LUT3 : 38

LUT3_D : 8

LUT3_L : 3

LUT4 : 37

LUT4_D : 1

LUT4_L : 6

MUXCY : 32

MUXF5 : 13

```
#      VCC                  : 1
#      XORCY                : 28
# FlipFlops/Latches         : 62
#      FDE                  : 60
#      LDCP                 : 2
# Clock Buffers             : 1
#      BUFGP                : 1
# IO Buffers                : 10
#      IBUF                 : 9
#      OBUF                 : 1
=========================================================================
Device utilization summary:
Selected Device : 2v40fg256-6
 Number of Slices:                  86  out of    256    33%
 Number of Slice Flip Flops:        62  out of    512    12%
 Number of 4 input LUTs:           131  out of    512    25%
 Number of bonded IOBs:             10  out of     88    11%
 Number of GCLKs:                    1  out of     16     6%

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.

Clock Information:

```
------------------

-----------------------------------+------------------------+-------+

Clock Signal                       | Clock buffer(FF name)  | Load  |

-----------------------------------+------------------------+-------+

clk                                | BUFGP                  | 60    |

u5_Mcompar__n0000_AEB:O            | NONE(*)(u5_control_1_0)| 2     |

-----------------------------------+------------------------+-------+
```

(*) This 1 clock signal(s) are generated by combinatorial logic,

and XST is not able to identify which are the primary clock signals.

Please use the CLOCK_SIGNAL constraint to specify the clock signal(s) generated by combinatorial logic.

Timing Summary:

Speed Grade: -6

Minimum period: 14.693ns (Maximum Frequency: 68.060MHz)

Minimum input arrival time before clock: 13.218ns

Maximum output required time after clock: 5.880ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

Timing constraint: Default period analysis for Clock 'clk'

Delay: 14.693ns (Levels of Logic = 13)

Source: u6_x1_path0_Q_0

Destination: u6_buff_Q_0

Source Clock: clk rising

Destination Clock: clk rising

Data Path: u6_x1_path0_Q_0 to u6_buff_Q_0

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDE:C->Q	2	0.449	0.605	u6_x1_path0_Q_0 (u6_x1_path0_Q_0)
LUT3:I1->O	1	0.347	0.000	u6_b11_Mmux__COND_3_inst_mux_f5_0111_F (N11090)
MUXF5:I0->O	3	0.345	0.750	u6_b11_Mmux__COND_3_inst_mux_f5_0111 (u6_state1<0>)
MUXF5:S->O	3	0.553	0.750	u6_b10_Mmux__COND_3_inst_mux_f5_0111 (u6_state2<0>)
MUXF5:S->O	3	0.553	0.750	u6_b9_Mmux__COND_3_inst_mux_f5_0111 (u6_state3<0>)

MUXF5:S->O 3 0.553 0.750 u6_b8_Mmux__COND_3_inst_mux_f5_0111 (u6_state4<0>)

MUXF5:S->O 3 0.553 0.750 u6_b7_Mmux__COND_3_inst_mux_f5_0111 (u6_state5<0>)

MUXF5:S->O 2 0.553 0.605 u6_b6_Mmux__COND_3_inst_mux_f5_0111 (u6_state6<0>)

MUXF5:S->O 4 0.553 0.818 u6_b5_Mmux__COND_3_inst_mux_f5_0111 (u6_state7<0>)

LUT3:I0->O 1 0.347 0.000 u6_b4_Mmux__COND_3_inst_mux_f5_0111_F (N11085)

MUXF5:I0->O 3 0.345 0.750 u6_b4_Mmux__COND_3_inst_mux_f5_0111 (u6_state8<0>)

MUXF5:S->O 3 0.553 0.750 u6_b3_Mmux__COND_3_inst_mux_f5_0111 (u6_state9<0>)

MUXF5:S->O 1 0.553 0.312 u6_b2_Mmux__COND_3_inst_mux_f5_0111 (u6_state10<0>)

MUXF5:S->O 1 0.553 0.000 u6_b1_Mmux__COND_3_inst_mux_f5_0111 (u6_state11<0>)

FDE:D 0.293 u6_buff_Q_0

Total 14.693ns (7.103ns logic, 7.590ns route)

(48.3% logic, 51.7% route)

Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'

Offset: 11.999ns (Levels of Logic = 16)

Source: in1<0>

Destination: u2_metric3_Q_2

Destination Clock: clk rising

Data Path: in1<0> to u2_metric3_Q_2

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
IBUF:I->O	2	0.653	0.605	in1_0_IBUF (in1_0_IBUF)
LUT2_D:I0->LO	1	0.347	0.000	u1_Madd__n0001_inst_lut2_01 (N11185)
MUXCY:S->O	1	0.235	0.000	u1_Madd__n0001_inst_cy_0 (u1_Madd__n0001_inst_cy_0)
MUXCY:CI->O	1	0.042	0.000	u1_Madd__n0001_inst_cy_1 (u1_Madd__n0001_inst_cy_1)
XORCY:CI->O	4	1.007	0.818	u1_Madd__n0001_inst_sum_2 (p3_1<2>)
LUT4:I2->O	1	0.347	0.312	u4_Ker367832 (CHOICE1001)
LUT3:I2->O	3	0.347	0.750	u4_Ker367879 (u4_N3680)
LUT3_D:I0->O	4	0.347	0.818	

u5_Mmux__old_find_smallest_1_a_1_I1_Result1_SW0 (out1<1>)

LUT2:I1->O 1 0.347 0.312

u5_Mmux__old_find_smallest_1_a_1_I2_Result41_G_SW0 (N11010)

LUT4:I3->O 1 0.347 0.000

u5_Mmux__old_find_smallest_1_a_1_I2_Result411_F (N11075)

MUXF5:I0->O 6 0.345 0.894

u5_Mmux__old_find_smallest_1_a_1_I2_Result411 (u5__n0010<3>)

LUT4:I3->O 5 0.347 0.855

u5_Mmux_find_smallest_1_find_smallest_I2_Result90 (CHOICE1040)

LUT4_L:I2->LO 1 0.347 0.000 u5_Msub_m_in3_inst_lut2_31 (u5_Msub_m_in3_inst_lut2_3)

MUXCY:S->O 1 0.235 0.000 u5_Msub_m_in3_inst_cy_3 (u5_Msub_m_in3_inst_cy_3)

MUXCY:CI->O 0 0.042 0.000 u5_Msub_m_in3_inst_cy_4 (u5_Msub_m_in3_inst_cy_4)

XORCY:CI->O 1 1.007 0.000 u5_Msub_m_in3_inst_sum_5 (m_in3<2>)

FDE:D 0.293 u2_metric3_Q_2

Total 11.999ns (6.635ns logic, 5.364ns route)

(55.3% logic, 44.7% route)

Timing constraint: Default OFFSET IN BEFORE for Clock 'u5_Mcompar__n0000_AEB:O'

Offset: 13.218ns (Levels of Logic = 15)

Source: in1<0>

Destination: u5_control_1_0

Destination Clock: u5_Mcompar__n0000_AEB:O falling

Data Path: in1<0> to u5_control_1_0

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
IBUF:I->O	2	0.653	0.605	in1_0_IBUF (in1_0_IBUF)
LUT2_D:I0->LO	1	0.347	0.000	u1_Madd__n0001_inst_lut2_01 (N11185)
MUXCY:S->O	1	0.235	0.000	u1_Madd__n0001_inst_cy_0 (u1_Madd__n0001_inst_cy_0)
MUXCY:CI->O	1	0.042	0.000	u1_Madd__n0001_inst_cy_1 (u1_Madd__n0001_inst_cy_1)
XORCY:CI->O	4	1.007	0.818	u1_Madd__n0001_inst_sum_2 (p3_1<2>)
LUT4:I2->O	1	0.347	0.312	u4_Ker367832 (CHOICE1001)
LUT3:I2->O	3	0.347	0.750	u4_Ker367879 (u4_N3680)
LUT3_D:I0->O	4	0.347	0.818	u5_Mmux__old_find_smallest_1_a_1_I1_Result1_SW0 (out1<1>)
LUT2:I1->O	1	0.347	0.312	u5_Mmux__old_find_smallest_1_a_1_I2_Result41_G_SW0 (N11010)

LUT4:I3->O 1 0.347 0.000 u5_Mmux__old_find_smallest_1_a_1_I2_Result411_F (N11075)

MUXF5:I0->O 6 0.345 0.894 u5_Mmux__old_find_smallest_1_a_1_I2_Result411 (u5__n0010<3>)

LUT4:I3->O 5 0.347 0.855 u5_Mmux_find_smallest_1_find_smallest_I2_Result90 (CHOICE1040)

LUT3:I0->O 4 0.347 0.818 u5_Mmux_find_smallest_1_find_smallest_I2_Result101 (u5_find_smallest_1_find_smallest<0>)

LUT3:I1->O 3 0.347 0.750 u5_Mcompar__n0002_AEB (u5__n0002)

LUT2:I0->O 1 0.347 0.312 u5_control_1__n00001 (u5_control_1__n0000)

LDCP:CLR 0.222 u5_control_1_0

Total 13.218ns (5.974ns logic, 7.244ns route)

(45.2% logic, 54.8% route)

Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'

Offset: 5.880ns (Levels of Logic = 1)

Source: u6_buff_Q_0

Destination: decode_out

Source Clock: clk rising

Data Path: u6_buff_Q_0 to decode_out

```
Gate     Net
   Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
   ----------------------------------------  ------------
    FDE:C->Q             1   0.449   0.312  u6_buff_Q_0 (u6_buff_Q_0)
    OBUF:I->O                    5.119          decode_out_OBUF (decode_out)
   ----------------------------------------
   Total                     5.880ns (5.568ns logic, 0.312ns route)
                                 (94.7% logic, 5.3% route)
CPU : 11.19 / 15.09 s | Elapsed : 11.00 / 14.00 s
 Total memory usage is 74716 kilobytes
```

CHAPTER-7

CONCLUSION & FUTURE SCOPE

Viterbi Algorithm is widely used for the elimination of the potential noise in a data stream. Encoding is such that the Viterbi Decoder can remove potential noise in the incoming stream by decoding it. The charecteristics of the decoder are its effectiveness in noise elimination, speed of decoding and cost (hardware utilisation).

This thesis has presented the design and Implementation of the Viterbi Decoder.Its streamed input-output, regular architecture and parallel execution favor on an FPGA Implementation.

FPGAs open a wide range of opportunities in the solution space that can result in high performance and economic solutions to a DSP problem because they do nat map well to software programmable DSP architectures. The algorithm will have an ASIC solution but this may not be an option for reasons of schedule, economics of scale and flexibility. Applications such as data communications and image processing require more processing power but when the fastest DSP processor is not fast enough, the only alternatives are to add multiple DSP processors or to design custom hardware devices. Multiple DSP processors are expensive, require many components and consume too much power. The performance gain that comes with each additional processor is small when compared to the increase in cost, board space, power consumption, and development time.

Custom devices deliver the performance but sacrifice flexibility and require a large engineering investment with no chance to recover from mistakes. FPGAs are the new solution used by many engineers to implement computationally intensive algorithms.

FPGAs offer the best of all the worlds, the flexibility of a programmable solution, the performance of a custom solution, and lowering overall cost.

So keeping in view of the advantages of using FPGAs as compared to other customised devices this work can further be increased to the designing & implementation of a generic viterbi decoder. The genericity of the design facilitates not only the rapid prototyping of Viterbi decoders with different specifications but moreover, it explores the performance of different implementations in order to obtain the most suitable solution for a particular communication system. Some of the generic parameters are basic decoder specifications, metric size, trellis window length, number of surviving paths and pipeline depth. A new Viterbi decoder with new specifications can be realized by only re-synthesizing the code.

REFERENCES AND BIBILOGARPHY

1. Bernard Sklar, Digital Communication, .pearson education second edition ,2001.

2. S.Haykan, Communication Systems , Wiley, 1994.

3. L.R.Bahl, J.Cocke,F.Jelinek, and J.Raviv,"Optimal decoding of linear block codes for minimizing symbol error rate," IEEE Trans. Inform. Theory, vol.IT-20,pp.284-287,1974.

4. J.LMassey ,"foundation and methods of channel encoding ,"in Proc.Intl.Conf. Inform. Theory Syst.,NTG- fachderichte,Berlin 1978.

5. J.K.Wolf ,"Efficient maximum likelihood decoding on linear bock codes using a trellis,"IEEE Trans. Inform. Theory, Vol.IT-24,pp.76-80, Jan.1978.

6. Y.Berger and Y.Be'ry,," Bounds on the Trellis size of linerar block codes,"IEEE Trans, Inform.. Theory, Vol. 39.1993.

7. D.J . Muder ," Minimal trellises for block codes ," IEEE Trans.. Inform. Theory, Vol34,pp. 1049-1053, Se 1998.

8. Odenwalder, J.P.," Optimal decoding of Convolutional codes",Ph.D.dissertation , university of California, Los Angeles ,1971.

9. Gallager,R.G.,"Information teory and Reliable Communication , John Wiley & Sons, Inc New York, 1968.

10. Fano,R.M.," A heruisitic Discussionn of Probabilistic Decoding ", IRE Trans. Inf. Theory ,Vol.IT9,No2,1963,pp.64-74.

11. Larson ,K.J,"Short Convolutional Codes with maximal free Distance for rates ½,1/3and ¼", IEEE Trans , Inf. Theory Vol.IT19,No.3,1973,pp.371-372.

12. Viterbi ,A.” Convolutional codes and their performance in communication systems “’IEEE Trans . Commun. Technol,VOl.Com19,no,5,Oct.1971,pp.715-772.
13. Forney, G.D. Jr. and Bower E.K., “ A high speed Sequential decoder : Prototype design and Tests”, IEEE Trans .Communs .Technol., vol.COM 19, no.5 Oct1971.pp.821-835.
14. Jelinek, F.,Fast Sequential Decoding Algorithm using a stack,”IBM J.Res.Dev., Vol.13,Nov.1969,pp 675-685
15. Xilinx Webpack software manual 2002.
16. Samir Palnitkar , Verilog HDL Pearsonn education 3rd edition 2000.
17. J.Bhaskar , Verilog HDL Synthesis A practical primer 2nd edition 1999.