



SCHOOL OF ELECTRONICS ENGINEERING

VIT - CHENNAI CAMPUS

CONVOLUTIONAL CODING AND VITERBI DECODER IMPLEMENTATION ON FPGA

ECE307 – INFORMATION THEORY AND CODING

Project Submitted by :

Shraddha Agrawal – 13BEC1138

Prithvish V N – 13BEC1100

Priyamvada Kundu – 13BEC1113

Guided by :

Prof. Ralph Samuel Thangaraj

SENSE – VIT Chennai

INDEX

S. No.	Content	Page No.
1	Introduction	3
2	Convolutional Encoding	4-5
3	Viterbi Decoding	6-7
4	MATLAB implementation	8
5	Procedure for implementation on FPGA	9-10
6	Results	11-12

INTRODUCTION

This project implements convolutional encoding and decoding using Viterbi algorithm on FPGA (Field Programmable Gate Array). These are popular techniques for channel encoding and decoding. The purpose of channel encoding is to find codes which transmit quickly, contain many valid code words and can correct or at least detect many errors. Channel encoding adds extra data bits (redundancy) to make the transmission of data more robust to disturbances present on the transmission channel. Channel decoder attempts to reconstruct the original information sequence from the knowledge of the channel encoding algorithm.

Convolutional coding is a type of error correcting codes that generates parity symbols via the sliding application of a Boolean polynomial function to a data stream. Viterbi algorithm aims at finding the most likely transmitted message sequence minimizing the BER. Though it is resource consuming but it does the maximum likelihood decoding. It can be a hard decision decoding (using Hamming distance) or a soft decision decoding (using Euclidean distance). These are the techniques commonly used in popular wireless standards and satellite communication. This technique is used in various space missions such as Mars Pathfinder, Mars Exploration Rover, Cassini probe to Saturn.

In the project, a convolutional code with code rate $\frac{1}{2}$ and constraint length 3 is implemented on FPGA (SPARTAN 3E) and the same is decoded using hard decision Viterbi decoder.

CONVOLUTIONAL ENCODING

Convolutional coding is a type of error correcting codes that generates parity symbols via the sliding application of a Boolean polynomial function to a data stream.

The encoder uses a sliding window to calculate $r > 1$ parity bits by combining various subsets of bits in the window. The combining is a simple addition i.e., modulo 2 addition, or equivalently, an exclusive-or operation.

Convolutional codes are described using two parameters :

- Code rate
- Constraint length

The code rate, k/n , is expressed as a ratio of the number of bits into the convolutional encoder (k) to the number of channel symbols output by the convolutional encoder (n) in a given encoder cycle. If a convolutional code produces r parity bits per window and slides the window forward by one bit at a time, its rate is $1/r$. The constraint length parameter, K , denotes the "length" of the convolutional encoder, i.e. how many k -bit stages are available to feed the combinatorial logic that produces the output symbols. The size of the window, in bits, is called the coder's constraint length.

An example of a convolutional code with code rate $1/2$ and constraint length 3.

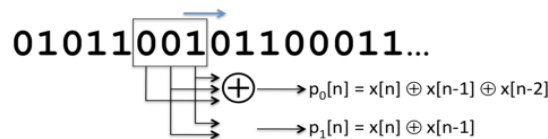


Fig. 1 Sliding window and encoding process

Encoding process :

- The encoder looks at K bits at a time and produces r parity bits according to carefully chosen functions that operate over various subsets of the K bits.
- The encoder spits out r bits, which are sent sequentially, slides the window by 1 to the right, and then repeats the process.
- By convention, we will assume that each message has $K - 1$ "0" bits padded in front, so that the initial conditions work out properly

One can view each parity equation as being produced by composing the message bits, X , and a **generator polynomial, g** . In the above, the generator polynomial coefficients are $(1, 1, 1)$ and $(1, 1, 0)$.

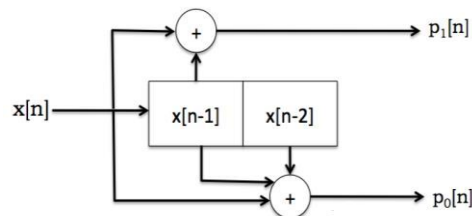


Fig 2 Block diagram view using shift registers

State machine corresponds to a view of the encoder as a set of states with well-defined transitions between them.

Input $x[n]$	Present State $x[n-1]$ $x[n-2]$		Output $p0[n]$ $p1[n]$		Next State $x[n-1]$ $x[n-2]$	
0	0	0	0	0	0	0
1	0	0	1	1	1	0
0	0	1	1	0	0	0
1	0	1	0	1	1	0
0	1	0	1	1	0	1
1	1	0	0	0	1	1
0	1	1	0	1	0	1
1	1	1	1	0	1	1

Table 1 : Transition table

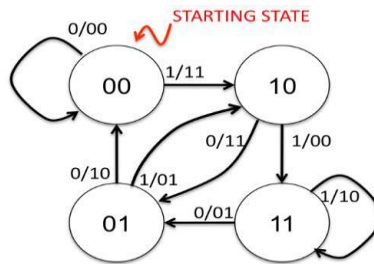


Fig 3. State diagram

This state machine view is an elegant way to explain what the transmitter does, and also what the receiver ought to do to decode the message. The transmitter begins in the initial state (labeled “STARTING STATE”) and processes the message one bit at a time. For each message bit, it makes the state transition from the current state to the new one depending on the value of the input bit, and sends the parity bits that are on the corresponding arc. The receiver, of course, does not have direct knowledge of the transmitter’s state transitions. It only sees the received sequence of parity bits, with possible corruptions. Its task is to determine the **best possible sequence of transmitter states that could have produced the parity bit sequence**.

The trellis is a structure derived from the state machine that will allow us to develop an efficient way to decode convolutional codes. The state machine view shows what happens at each instant when the sender has a message bit to process, but doesn’t show how the system evolves in time. The trellis is a structure that makes the time evolution explicit.

- Each column of the trellis has the set of states
- Each state in a column is connected to two states in the next column—the same two states in the state diagram.
- The top link from each state in a column of the trellis shows what gets transmitted on a “0”, while the bottom shows what gets transmitted on a “1”.

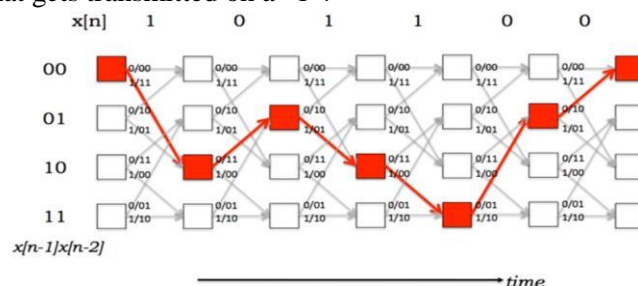


Fig.4 Trellis for convolutional encoder

VITERBI DECODING

The receiver should determine the “best possible” sequence of transmitter states. A decoder that is able to infer the most likely sequence (i.e., message bits) that must have been traversed (sent) by the transmitter is also called a maximum likelihood decoder.

Consider a simple numerical example. Suppose that bit errors are independent and identically distributed with a BER of 0.001, and that the receiver digitizes a sequence of analog samples into the bits 1101001. Is the sender more likely to have sent 1100111 or 1100001? The first has a Hamming distance of 3, and the probability of receiving that sequence is $(0.999)^4 (0.001)^3 = 9.9 \times 10^{-10}$. The second choice has a Hamming distance of 1 and a probability of $(0.999)^6 (0.001)^1 = 9.9 \times 10^{-4}$, which is six orders of magnitude higher and is overwhelmingly more likely.

Thus, the most likely sequence of parity bits that was transmitted must be the one with the smallest Hamming distance from the sequence of parity bits received. Given a choice of possible transmitted messages, the decoder should pick the one with the smallest such Hamming distance.

The trellis is a structure derived from the state machine that will allow us to develop an efficient way to decode convolutional codes. The Viterbi decoder finds a maximum likelihood path through the Trellis. One needs a way to capture any errors that occur in going through the states of the trellis, and a way to navigate the trellis without actually materializing the entire trellis (i.e., without enumerating all possible paths through it and then finding the one with smallest accumulated error). The Viterbi decoder solves these problems.

The decoding algorithm uses two metrics:

- the branch metric (BM)
- the path metric (PM).

The **branch metric** is a measure of the “distance” between what was transmitted and what was received, and is defined for each arc in the trellis. In hard decision decoding, where we are given a sequence of digitized parity bits, the branch metric is the Hamming distance between the expected parity bits and the received ones. Whereas in soft decision decoding, it is the Euclidean distance. (Hamming distance is the number by which the bits differ in each symbol.) The **path metric** is a value associated with a state in the trellis (i.e., a value associated with each node). For hard decision decoding, it corresponds to the Hamming distance over the most likely path from the initial state to the current state in the trellis. By “most likely”, we mean the path with smallest Hamming distance between the initial state and the current state, measured over all possible paths between the two states.

The key insight in the Viterbi algorithm is that the receiver can compute the path metric for a (state, time) pair incrementally using the path metrics of previously computed states and the branch metrics.

Suppose the receiver has computed the path metric $PM[s, i]$ for each state ‘s’ at time step i. The value of $PM[s, i]$ is the total number of bit errors detected when comparing the received parity bits to the most likely transmitted message, considering all messages that could have been sent by the transmitter until time step i (starting from state “00”, which we will take by convention to be the starting state always).

Among all the possible states at time step i , the most likely state is the one with the smallest path metric. If there is more than one such state, they are all equally good possibilities. Any message sequence that leaves the transmitter in state s at time $i + 1$ must have left the transmitter in state α or state β at time i .

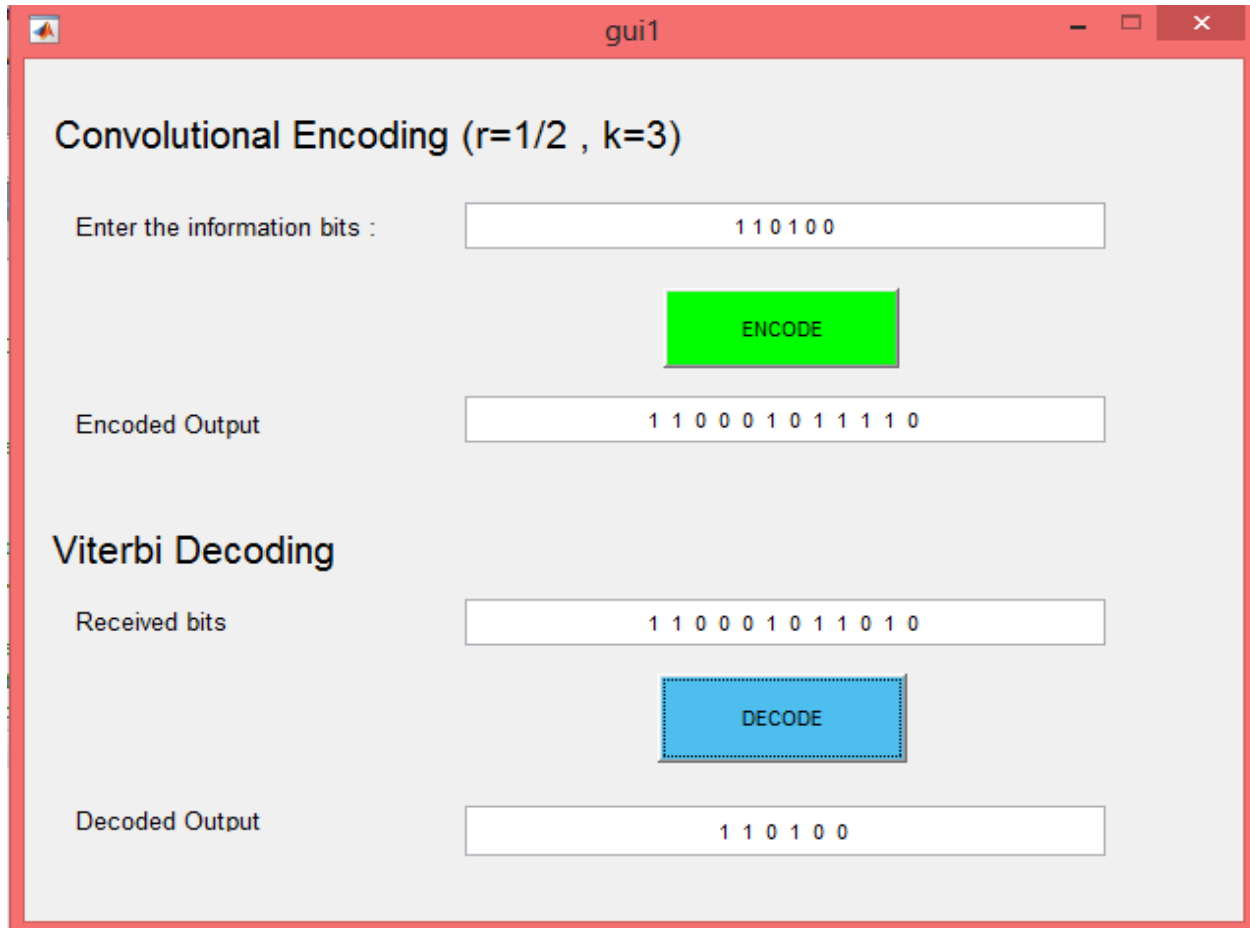
$$PM[s, i + 1] = \min(PM[\alpha, i] + BM[\alpha \rightarrow s], PM[\beta, i] + BM[\beta \rightarrow s]),$$

where α and β are the two predecessor states.

Initially, state '00' has a cost of 0 and the other 2^{k-1} states have a cost of ∞ . The main loop of the algorithm consists of two main steps: calculating the branch metric for the next set of parity bits, and computing the path metric for the next column. The path metric computation may be thought of as an add-compare-select procedure:

1. Add the branch metric to the path metric for the old state.
2. Compare the sums for paths arriving at the new state (there are only two such paths to compare at each new state because there are only two incoming arcs from the previous column).
3. Select the path with the smallest value, breaking ties arbitrarily. This path corresponds to the one with fewest errors.

MATLAB IMPLEMENTATION



The image shows a MATLAB GUI window titled 'gui1'. It is divided into two main sections: 'Convolutional Encoding (r=1/2 , k=3)' and 'Viterbi Decoding'. In the encoding section, there is a text input field labeled 'Enter the information bits :' containing the binary sequence '1 1 0 1 0 0'. Below this is a green button labeled 'ENCODE'. The 'Encoded Output' is displayed in a text field as '1 1 0 0 0 1 0 1 1 1 0'. The Viterbi Decoding section has a text input field labeled 'Received bits' containing '1 1 0 0 0 1 0 1 1 0 1 0'. Below this is a blue button labeled 'DECODE'. The 'Decoded Output' is shown in a text field as '1 1 0 1 0 0'.

Convolutional Encoding ($r=1/2$, $k=3$)

Enter the information bits : 1 1 0 1 0 0

ENCODE

Encoded Output 1 1 0 0 0 1 0 1 1 1 0

Viterbi Decoding

Received bits 1 1 0 0 0 1 0 1 1 0 1 0

DECODE

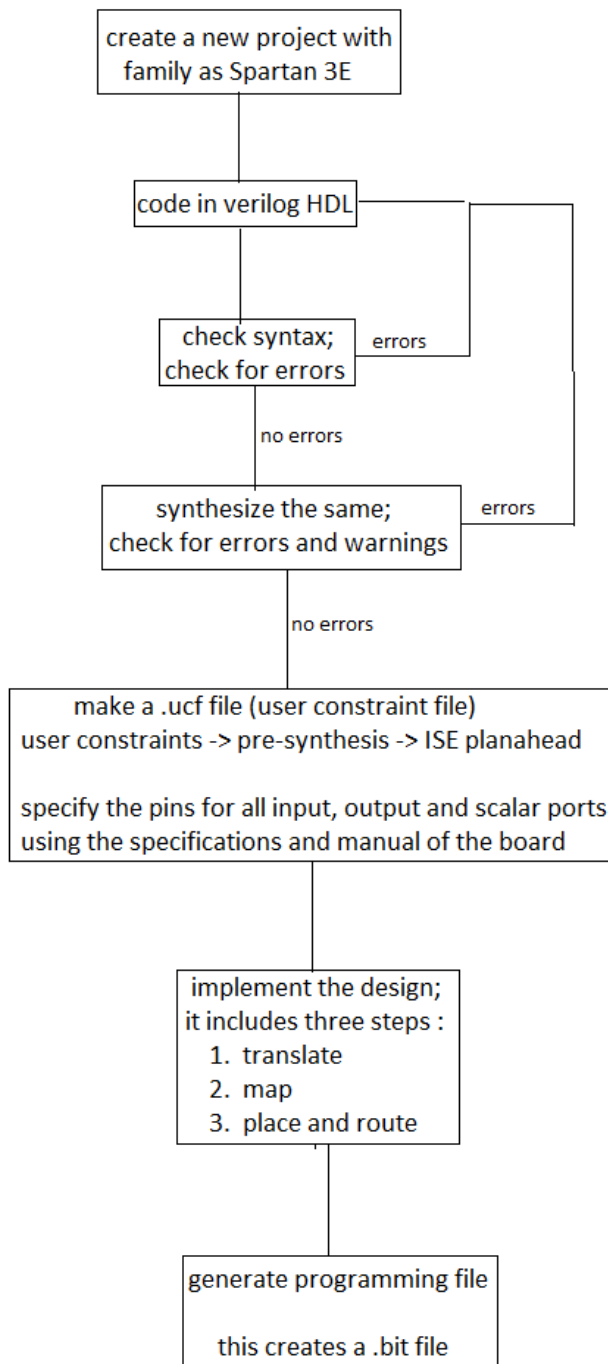
Decoded Output 1 1 0 1 0 0

Fig. 5 Result from MATLAB GUI implementation

The codes for convolutional encoder and Viterbi decoder were made using the concepts explained above and these were combined and put into a GUI which can be easily made using MATLAB GUI tools.

PROCEDURE FOR IMPLEMENTATION USING FPGA

FPGA used : Xilinx Spartan 3E box



configure target devices
a new iMPACT file is created

ISE iMPACT :
file -> new project
->select 'enter a boundaryscan chain manually'
-> add Xilinx device
-> choose the above created .bit file
-> output file type - XSVF file - create XSVF file -
save it in desired location
-> Program - pulse PROG
-> outfile file type - XSVF file - stop writing file
-> close and save the impact file

Open Topview Programmer - Xilinx FPGA
browse - select the above created XSVF file
configure



Fig. 6 FPGA board (XC320E)

RESULT

Encoder :

1. input = 1000
output = 11101100

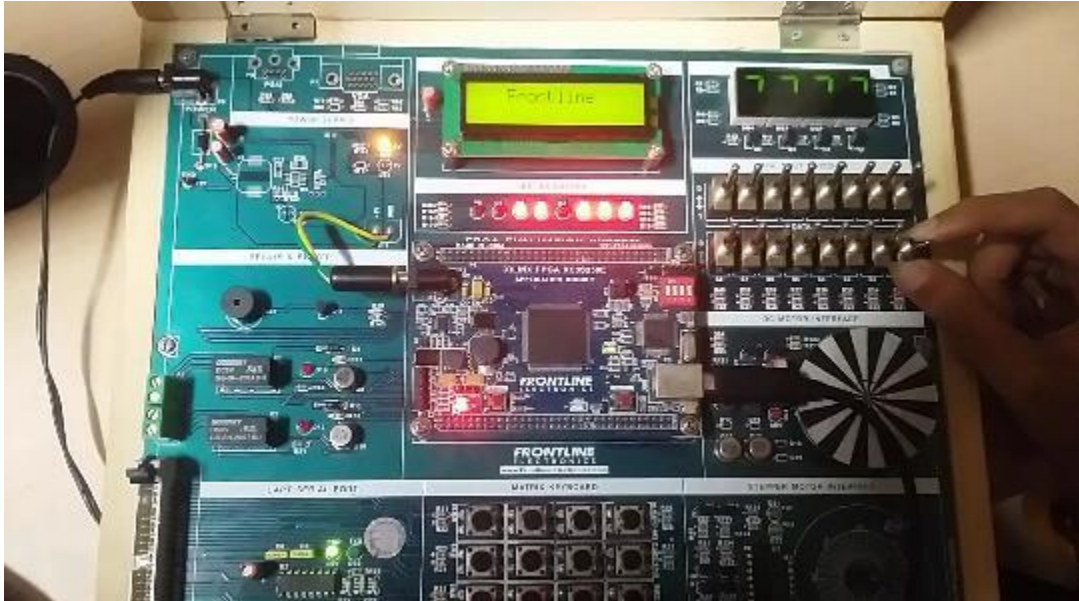


Fig. 7 given input – 1000

2. input = 1100
output = 11010111

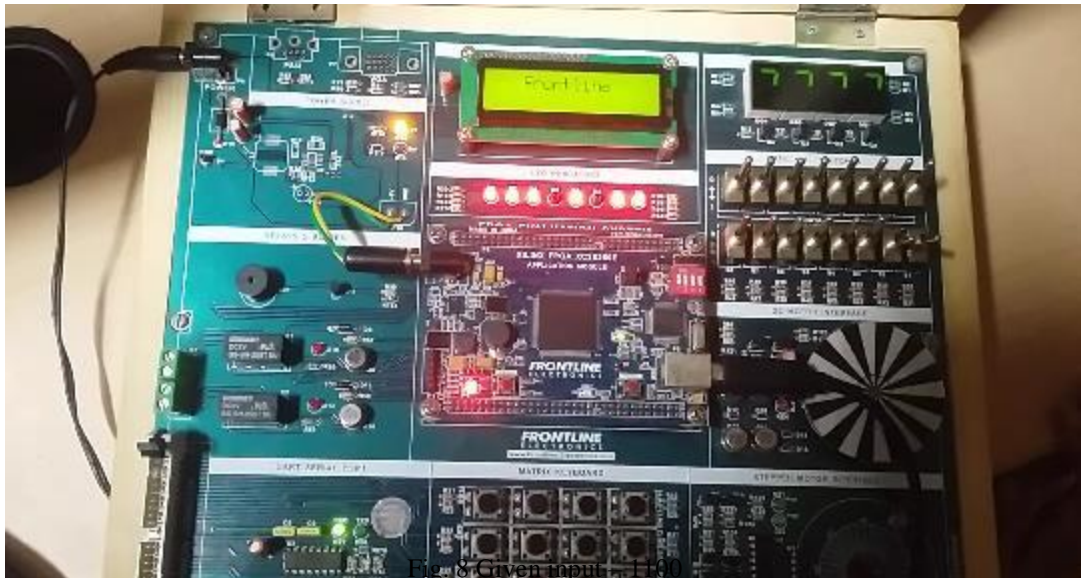


Fig. 8 Given input – 1100

3. input = 1101
output = 11010100

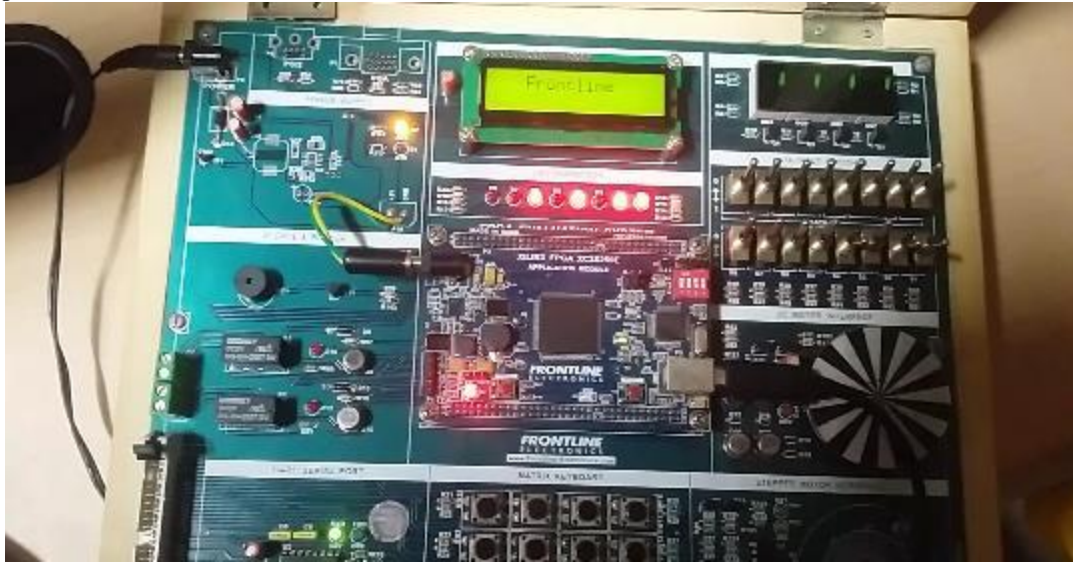


Fig. 9 Given input - 1101