IMPLEMENTATION OF TWO-STAGE COMMUNICATION SYSTEM USING FPGA

PROJECT REPORT

submitted by,

ADHEENA S.S (TKM20EC003)

FATHIMA SUNIL (TKM20EC047)

RAMPRASAD K THAMBAN (TKM20EC101)

RAHULVINAYAK J.R (LTKM20EC139

to

APJ Abdul Kalam Technological University

in partial fulfilment of the requirements for the award of the

Degree of *Bachelor of Technology in Electronics &Communication*

under the guidance of

Prof. AMAL MOLE S

(Assistant Professor, Dept of ECE, TKM College of Engineering, Kollam)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
TKM COLLEGE OF ENGINEERING, KOLLAM 691005

MAY 2024

DECLARATION

We, hereby declare that the project report **"IMPLEMENTATION OF TWO-STAGE COMMUNICATION SYSTEM USING FPGA"** submitted for partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Electronics and Communication Engineering of APJ Abdul Kalam Technological University, Kerala, is a bonafide work done by us under the supervision of Prof. Amal Mole S (Assistant Professor of the Dept of ECE, TKM College of Engineering), Prof. Vishnu Damodaran (Associate Professor of the Dept. of ECE, TKM College of Engineering, Kollam). This submission represents our ideas in our own words, and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained.

Place: KOLLAM

Date: 08-05-24

ADHEENA SS (TKM20EC003)

FATHIMA SUNIL(TKM20EC047)

RAHULVINAYAK J.R(LTKM20C139)

RAMPRASAD K THAMBAN(TKM20EC101)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TKM COLLEGE OF ENGINEERING,

KOLLAM 691005

CERTIFICATE

This is to certify that the project report entitled **"IMPLEMENTATION OF TWO-STAGE COMMUNICATION SYSTEM USING FPGA"** is a bonafide record of the work presented by **ADHEENA S.S (TKM20EC002), FATHIMA SUNIL (TKM20EC047), RAMPRASAD K THAMBAN (TKM20EC101), RAHULVINAYAK J.R (LTKM20EC139)** to APJ Abdul Kalam Technological University in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Electronics and Communication Engineering during the academic year 2023-2024 under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Prof. Amal Mole	**Prof. Vishnu Damodaran**	**Dr. Nishanth N**
Project Guide	Project Coordinator	Head of the Department

ACKNOWLEDGMENT

We take this opportunity to convey our deep sense of gratitude to all those who have been kind enough to offer their advice and assistance when needed which has led to the successful completion of this project. First of all, we thank God Almighty for all His blessings throughout this endeavour without which it would not have been possible.

It is our privilege and pleasure to express our profound sense of respect, gratitude, and indebtedness to **Dr. Nishanth N**, Head of the Department of Electronics and Communication Engineering for guiding and providing facilities for the successful completion of the preliminary phase of our project work. We sincerely thank **Prof. Vishnu Damodaran**, Project Coordinator, Professor, Department of Electronics and Communication Engineering for her guidance, valuable support, and constant encouragement given to us during this work.

We take this opportunity to express our sincere gratitude to our guide, **Prof. Amal Mole S**, Assistant Professor, Department of Electronics and Communication Engineering.

We are also thankful to all the faculty members of the Department of Electronics and Communication Engineering and mentors for their guidance and wholehearted support. Their effort and sincerity will always be remembered in our hearts.

Last, but not the least, we wish to acknowledge our parents and friends for giving us moral strength and encouragement throughout this period.

ADHEENA S.S (TKM20EC003)

FATHIMA SUNIL (TKM20EC047)

RAHULVINAYAK J.R(LTKM20EC139)

RAMPRASAD K THAMBAN (TKM20EC101)

ABSTRACT

Efficient signal encoding and decoding are vital components of modern communication systems, particularly in compact handheld devices where space and power constraints are paramount. This paper presents a novel approach to address this challenge by proposing dedicated hardware designed exclusively for signal encoding and decoding, aimed at ensuring error-free transmission.

The proposed system consists of two stages: the source encoder and the channel encoder. Initially, the speech signal is digitized using an Analog-to-Digital Converter (ADC) and processed through a Pulse Code Modulation (PCM) source encoder. Subsequently, a channel encoder employing convolutional encoding techniques is utilized to further enhance the signal integrity. On the receiver side, the transmitted signal is

decoded using a Viterbi decoder, PCM decoder, and a Digital-to-Analog Converter (DAC), all integrated within the same Field-Programmable Gate Array (FPGA) for compactness and efficiency.

While traditional approaches may involve multiple integrated chips, resulting in increased power consumption, reduced efficiency, and larger spatial footprint, our proposed solution aims to mitigate these drawbacks. By consolidating the entire encoding and decoding process into a single chip, our approach improves coding efficiency, lowers power consumption, and reduces size requirements, thus meeting the demands of power-efficient handheld devices.

Through extensive experimentation and analysis, we demonstrate the effectiveness of our dedicated hardware solution in achieving error-free transmission, thereby offering a promising avenue for the development of robust communication systems for compact handheld devices. The implementation is done using Xilinx ISE 14.7 on to Spartan 6 FPGA board.

CONTENTS

LIST OF ABBREVIATIONS

FPGA	Field Programmable Gate Array
LDPC	Low-Density Parity-Check
AWGN	Additive White Gaussian Noise
FEC	Forward Error Correction

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 GENERAL BACKGROUND

A two-stage communication system with Field-Programmable Gate Arrays (FPGAs) efficiently encodes speech signals, reducing data size and enhancing transmission. Comprising source and channel encoders, these systems optimize signals at various stages. Initially, the Analog-to-Digital Converter (ADC) digitizes the analog speech signal. Pulse Code Modulation (PCM) encoding then efficiently represents the digitized signal for transmission. The channel encoder further refines the encoded signal, employing strategies like convolutional encoding for error resilience.

FPGAs' adaptability makes them ideal for both transmitter and receiver stages. Transmitter-side modules for PCM source and channel encoding efficiently process and encode speech signals using programmable logic blocks. The receiver side, also within the FPGA, decodes the received signal through processes like Viterbi and PCM decoding. Additionally, a Digital-to-Analog Converter (DAC) converts the decoded digital signal back to analog for output.Despite their advantages, FPGA-based systems pose challenges in design complexity and resource optimization, requiring expertise in hardware design and FPGA programming. Rigorous verification and testing ensure reliability. Nonetheless, the flexibility and integration capabilities of FPGAs make them compelling for efficient speech signal encoding in data transmission applications.

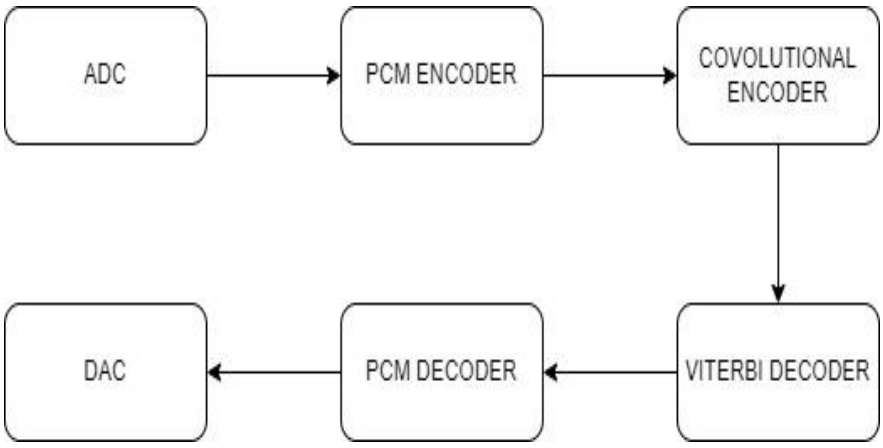


Figure 1: Block diagram of two stage communication sytem

1.2 ORGANIZATION OF CHAPTERS

The upcoming chapters highlight the following contents:

I. Chapter 2 is the Literature Review

II. Chapter 3 gives us an overview of Convolutional Encoder

III. Chapter 4 -PCM Encoder

IV. Chapter 5- PCM Decoder

V. Chapter 6- Viterbi Decoder

VI. Chapter 7-Methodology

VII. Chapter 8- Results and Discussions

VIII. Chapter 9- Conclusion and Future Scope

IX. References

CHAPTER 2

LITERATURE REVIEW

[1] Dr. Dhafir A. Alneema " FPGA Based Implementation of Convolutional Encoder- Viterbi Decoder Using Multiple Booting Technique"

It focuses on designing and implementing a Convolutional encoder and Viterbi decoder on a Spartan 3E FPGA Starter kit. The FPGA adapts its function based on the program loaded from Intel flash memory, enabling it to function as either an encoder or a decoder without the need for additional hardware. Multiple Booting Technique: This technique allows for the implementation of both the Convolutional encoder and Viterbi decoder on the same FPGA. Expandability to Different Codes. The multiple booting technique can be extended to support other types of encoders with different parameters and corresponding decoders. Compatibility and Flexibility: The Viterbi decoder, with slight modifications, can be used in systems with different constraint lengths, frame sizes, and code rates.

[2] V.Kavinilavu1 , S. Salivahanan, V. S. Kanchana Bhaaskaran2 , Samiappa Sakthikumaran, B. Brindha and C. Vinoth "Implementation of Convolutional Encoder and Viterbi Decoder using Verilog HDL"

Focuses on the design and implementation of the Convolutional encoder and Viterbi decoder. This design has been simulated in MODELSIM 10.0e and synthesized using XILINX-ISE 12.4i Continuous Data Stream Capability: Unlike block codes, convolutional coding can be applied to both blocks of data and continuous data streams. Forward Error Correction (FEC): Convolutional codes with Viterbi decoding are widely used in communication systems to ensure reliable data transmission over noisy channels. Hardware Description Language (HDL) Implementation: The use of Verilog HDL signifies the practicality of the implementation in real-world hardware.

Akash Thakur and Manju K Chattopadhyay 2017" Design and Implementation of Viterbi Decoder Using VHDL"

This paper presents a digital design conversion of a Viterbi decoder for a ½ rate convolutional encoder with a constraint length of k = 3. The design is implemented in VHDL, simulated, and synthesized using XILINX

ISE 14.7.The synthesis results indicate a maximum operating frequency of 100.725 MHz, with reduced

memory requirements compared to conventional methods. Instead of storing the path metric of all four states for a single time instant, a decision is made by the survivor unit and reduce the memory requirement for a complete trellis length. Relevance in Communication and Data Storage Industries: Encoders and decoders are fundamental in these industries. The design addresses the need for reliable systems with minimal error probability and highspeed operation.

[3] K.Santhosh Kumar , M.V.H. Bhaskara Murthy" FPGA Implementation of Viterbi Algorithm for Decoding of Convolution Codes"

The Convolutional coding has an advantage over the block codes in that it can be applied to a continuous data stream as well as to blocks of data. Viterbi decoder employed in digital wireless communication plays a rife role in the overall power consumption of trellis coded modulation decoder. Power reduction in Viterbi decoder could be achieved by reducing the number of states. A pre-computation architecture with T-algorithm was implemented for this purpose, and when we compare this result with full Trellis Viterbi decoder, this approach significantly reduces power consumption without degrading decoding speed. Convolutional encoding with Viterbi decoding is a powerful FEC technique that is particularly suited to a channel in which the transmitted signal is corrupted mainly by Additive White Gaussian Noise (AWGN) .

[4] Mr.J.Anuj Sai , Mr.P.Kiran Kumar ,Mr. V.Vamshi Krishna Reddy ,Mr.L.Shiva Nagender Rao"Efficient Convolutional Adaptive Viterbi Encoder and Decoder Using RTL Design"2018

Convolution encoding with Viterbi decoding is a powerful method for forward error correction. This FEC scheme is an essential component of wireless communication systems. Viterbi decoder can perform efficiently when the correcting and detecting bit is only, when the bits increases the efficiency decreases.

[5] Rohan M. Pednekar, Dayanand B M" Design and Implementation of Convolution Encoder with Viterbi Decoder"

The paper introduces the importance of error detection and correction in communication systems, particularly in the presence of noise. It highlights the use of convolutional coding with Viterbi decoding for forward error correction. The paper explains the fundamentals of convolutional codes, including parameters like code rate and constraint length. It presents the structure of a convolutional encoder with generator polynomials and describes its operation through state, tree, and trellis diagrams. The Viterbi decoding algorithm is introduced as a maximum likelihood decoding technique. This paper mainly discusses about the results based on MATLAB simulations of the scenario.

[6] V. S. Sreekanth1, Y. Rama Krishna2, A. V. V. Prasad3 "Simulation and Implementation of Convolution Encoder and Viterbi Decoder"

The paper introduces the importance of error detection and correction in communication systems, particularly in the presence of noise. It highlights the use of convolutional coding with Viterbi decoding for forward error correction. The paper explains the fundamentals of convolutional codes, including parameters like code rate and constraint length. It presents the structure of a convolutional encoder with generator polynomials and describes its operation through state, tree, and trellis diagrams. The Viterbi decoding algorithm is introduced as a maximum likelihood decoding technique. This paper mainly discusses about the results based on MATLAB simulations of the scenario.

CHAPTER-3

CONVOLUTIONAL ENCODER

3.1 AN OVERVIEW OF CONVOLUTIONAL ENCODER

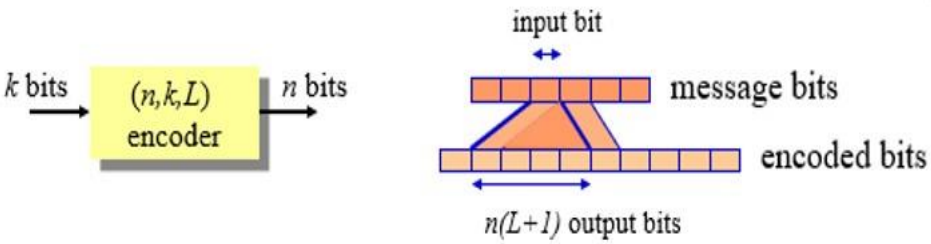


Figure 2: Block diagram Convolution Encoder

Convolutional codes represent a vital component in communication systems requiring reliable data transfer with low implementation complexity. Unlike block codes, convolutional codes operate on continuous streams of data, making them particularly adept for applications demanding real-time processing. These codes employ memory, indicated by the parameter L in (n, k, L), allowing them to utilize previous bits to influence the encoding or decoding of subsequent bits. The constraint length, defined as C = n(L + 1), is a crucial factor in the performance of convolutional codes, illustrating the number of encoded bits influenced by a single message bit.

As the memory depth expands, convolutional codes can achieve higher performance levels.

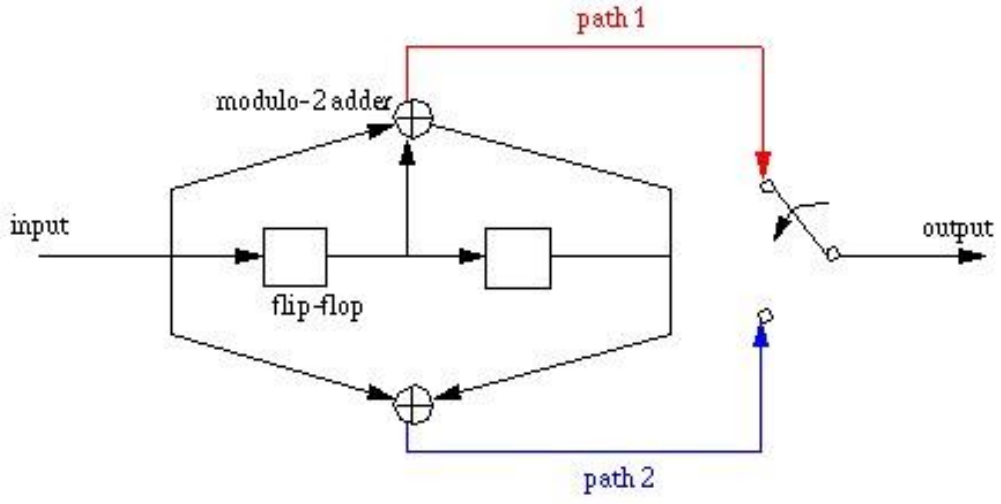


Figure 3: Block diagram of Convolution Encoder with k=1,n=2,r=1/2

Implemented through convolutional encoders, these codes transform m-bit information symbols into n-bit symbols. The code rate, denoted by R = k/n, is a fundamental metric influencing the efficiency of the code. Convolutional codes find widespread application across digital video transmission, radio communication, mobile networks, and satellite communication. Notably, they are often integrated with hard-decision codes, particularly Reed–Solomon codes, to enhance error correction capabilities.

Before the advent of turbo codes, convolutional codes stood out as highly efficient error-correcting codes, coming close to the limits defined by Shannon's theorem. The construction of convolutional codes involves elements like a shift register, where the constraint length (K) determines how many codewords one information bit affects in the encoder output. Various representations, including state diagrams, tree diagrams, and trellis diagrams, provide insights into the operational intricacies of convolutional encoders. In summary, convolutional codes play a pivotal role in achieving reliable data transfer, offering a balance between performance and implementation complexity across a spectrum of contemporary communication technologies.

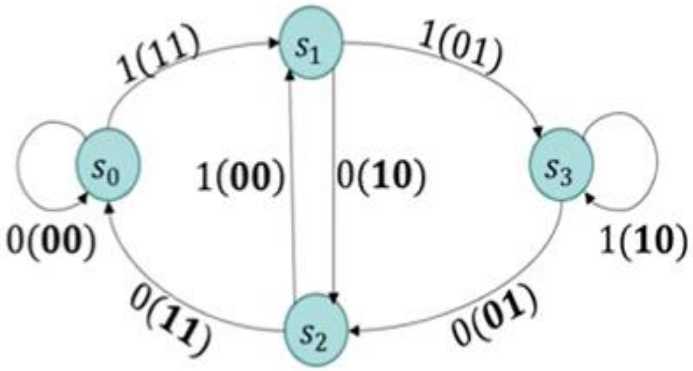


Figure 4: State Diagram

3.2 APPLICATIONS OF A CONVOLUTIONAL ENCODER

Convolutional encoders represent a transformative force in the realm of medical image analysis, where their intricate architecture proves instrumental in decoding the complexities of diagnostic imaging. In the context of radiology, these neural networks emerge as invaluable tools for tasks ranging from tumor detection to organ segmentation. The convolutional layers within the encoder excel at discerning nuanced patterns and hierarchical features in medical images, enabling a more accurate and efficient diagnosis. For instance, when applied to magnetic resonance imaging (MRI) data, convolutional encoders exhibit a remarkable ability to identify subtle variations in tissue structures, aiding in the early detection of anomalies. The spatial awareness inherent in convolutional layers allows the network to recognize the spatial relationships between pixels, crucial for understanding the intricate details present in medical imagery.

In tumor detection, convolutional encoders contribute significantly by isolating irregularities in tissues, helping clinicians identify potential malignancies with enhanced precision. The hierarchical feature extraction capabilities of these networks further enable the differentiation of diverse tissue types, facilitating comprehensive analyses of organ structures. Additionally, convolutional encoders play a pivotal role in medical image segmentation, delineating specific regions of interest within images for more targeted examinations. This proves particularly beneficial in treatment planning, where precise delineation of organs and tumors is crucial.

The impact of convolutional encoders extends beyond traditional medical imaging into the realm of functional neuroimaging. In tasks like functional MRI (fMRI) analysis, these encoders aid in mapping brain activity by capturing spatial and temporal patterns. This is pivotal for understanding neurological conditions and advancing research in fields like cognitive neuroscience. The ability of convolutional encoders to unravel intricate spatial dependencies and hierarchical features positions them as indispensable tools in the quest for more effective, accurate, and timely medical diagnoses, ultimately contributing to improved patient outcomes and the evolution of personalized medicine.

CHAPTER 4

PCM ENCODER

4.1 AN OVERVIEW OF PCM ENCODER

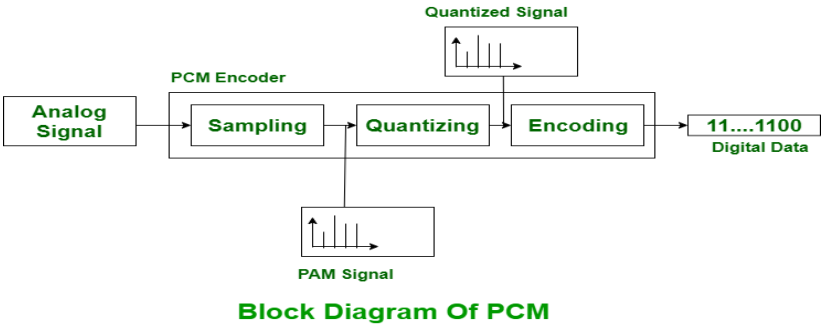


Figure 5: Block diagram of PCM Encoder

Pulse Code Modulation (PCM) encoding is a widely used method for converting analog signals, such as audio, into digital signals for transmission or storage. In PCM encoding, the analog signal is sampled at regular intervals, and each sample's amplitude is quantized into a digital value. These digital values are then transmitted or stored as binary data. PCM offers a straightforward and efficient way to digitize analog signals while preserving their essential characteristics.

A-Law and μ-Law companding are techniques commonly employed in PCM encoding to improve dynamic range and signal-to-noise ratio during transmission. Companding stands for "compressing" and "expanding" and involves nonlinear transformations of the signal amplitude before and after quantization, respectively. A-Law and μ-Law companding are logarithmic compression algorithms that map the input signal's amplitude to a nonlinear scale before quantization. This nonlinearity allows for more precise quantization of lower-amplitude signals, effectively improving the dynamic range and reducing quantization noise.

In A-Law companding, the input signal amplitude is compressed logarithmically using a specific algorithm, resulting in a more uniform distribution of quantization levels across the signal's dynamic range. Similarly, μ-Law companding employs a slightly different logarithmic compression algorithm to achieve similar benefits.

Quantization Endpoints by Segment Code								Quantization Code	
000	001	010	011	100	101	110	111		
0	32	64	128	256	512	1024	2048	0000	0
2	34	68	136	272	544	1088	2176	0001	1
4	36	72	144	288	576	1152	2304	0010	2
6	38	76	152	304	608	1216	2432	0011	3
8	40	80	160	320	640	1280	2560	0100	4
10	42	84	168	336	672	1344	2688	6101	5
12	44	88	176	352	704	1408	2816	0110	6
14	46	92	184	368	736	1472	2944	0111	7
16	48	96	192	384	768	1536	3072	1000	8
18	50	100	200	400	800	1600	3200	1001	9
20	52	104	208	416	832	1664	3328	1010	10
22	54	108	216	432	864	1728	3456	1011	11
24	56	112	224	448	896	1792	3584	1100	12
26	58	116	232	464	928	1856	3712	1101	13
28	60	120	240	480	960	1920	3840	1110	14
30	62	124	248	496	992	1984	3968	1111	15
32	64	128	256	512	1024	2048	4096		

Figure 6: A- law companding table

Both A-Law and μ-Law companding are widely used in telecommunications systems, particularly in digital voice transmission over networks like PSTN (Public Switched Telephone Network), where efficient use of bandwidth and high-quality audio reproduction are essential. We implement PCM Encoder and PCM Decoder using A-law companding table.

4.2 APPLICATIONS OF A PCM ENCODER

Pulse Code Modulation (PCM) encoders are widely used in digital communication systems to convert analog signals into digital format for transmission and storage. Common applications include telecommunications, digital audio recording, and medical imaging. For example, in telecommunications, PCM encoders are used in voice-over-IP (VoIP) systems to digitize analog voice signals for transmission over the internet

Telecommunications: PCM encoders are extensively used in telecommunications for converting analog voice signals into digital format. This digitization process allows for more efficient transmission and processing of voice data over communication networks. In traditional telephone systems, PCM encoders are used in both the subscriber's end (to convert voice signals into digital data) and the central office (to multiplex and transmit multiple voice channels over a single communication link).

Digital Audio Recording: PCM encoders play a crucial role in digital audio recording systems, such as CD players, digital audio recorders, and computer sound cards. Analog audio signals captured by microphones or other audio input devices are converted into digital PCM format for storage and processing. This digital representation allows for accurate reproduction of audio signals and facilitates various audio processing tasks like editing, mixing, and playback.

Medical Imaging: In medical imaging applications like MRI (Magnetic Resonance Imaging) and CT (Computed Tomography) scans, PCM encoders are used to convert analog signals from sensors (e.g., MRI coils, X-ray detectors) into digital format. This digital representation of medical data allows for efficient storage, transmission, and processing of imaging data, enabling healthcare professionals to analyze and diagnose medical conditions more effectively.

Data Transmission and Storage: PCM encoding is also utilized in various data transmission and storage systems where analog signals need to be converted into digital format for reliable transmission and storage. This includes applications like digital television broadcasting, satellite communications, and data logging systems where analog signals are converted into PCM format before transmission or storage to ensure data integrity and fidelity

CHAPTER 5

PCM DECODER

5.1 AN OVERVIEW OF PCM DECODER

The Pulse Code Modulation (PCM) decoder is the component of a communication system that reverses the encoding process, converting digital PCM signals back into their original analog form. The decoder is essential for reconstructing the original analog signal after transmission or storage in digital format.

The PCM decoding process involves several steps. First, the digital PCM signal is received and processed to recover the quantized digital values. These digital values represent the amplitude of the original analog signal at each sampling point.Next, the quantized digital values are typically expanded using the inverse of the companding algorithm used during encoding. For example, if A-Law or μ-Law companding was applied during encoding, the decoder applies the inverse companding function to expand the quantized values back to their original linear scale.

Once the quantized values are expanded, they are converted back into analog form using a Digital-to-Analog Converter (DAC). The DAC generates a continuous analog signal based on the discrete digital values, effectively reconstructing the original analog waveform. Finally, any additional processing or filtering may be applied to the reconstructed analog signal to remove noise or artifacts introduced during transmission or encoding.

Overall, the PCM decoder plays a crucial role in communication systems, ensuring that digital PCM signals are accurately converted back into their original analog form for playback or further processing. We use the A-Law companding table to implement PCM decoder which is shown in fig.4.

5.2 APPLICATIONS OF PCM DECODER

The Pulse Code Modulation (PCM) decoder serves a pivotal role across diverse applications where the conversion of digital PCM signals into their original analog form is essential. Telecommunication systems, encompassing both traditional and Voice over Internet Protocol (VoIP) networks, heavily rely on PCM decoding to transform digital voice signals back into analog audio for playback through speakers. Similarly, in the realm of audio recording and playback, devices like digital audio recorders, CD players, and music players utilize PCM decoding to render digital audio files stored in PCM format into analog signals, ensuring faithful reproduction of the original sound. Broadcasting systems also leverage PCM decoders to convert digitally encoded audio signals into analog form for transmission over radio or television, ensuring high-quality audio delivery to listeners or viewers.

Moreover, PCM decoding finds crucial applications in medical imaging technologies such as MRI and CT scanners, where digital signals representing medical images or patient data must be decoded into analog signals for display or analysis by medical professionals. Industrial control systems also utilize PCM decoding for converting digital control signals into analog form to manage various processes and equipment, aiding automation, robotics, and process control tasks. Additionally, in the domain of Digital Signal Processing (DSP), PCM decoding plays a fundamental role, particularly in applications such as audio processing, speech recognition, and image processing, where accurate conversion of digital signals back into analog form is paramount for effective analysis and manipulation. Overall, PCM decoding serves as a fundamental component across a broad spectrum of applications, facilitating the seamless conversion of digital PCM signals into their original analog representations for playback, transmission, analysis, or control purposes.

CHAPTER 6

VITERBI DECODER

6.1 AN OVERVIEW OF VITERBI DECODER

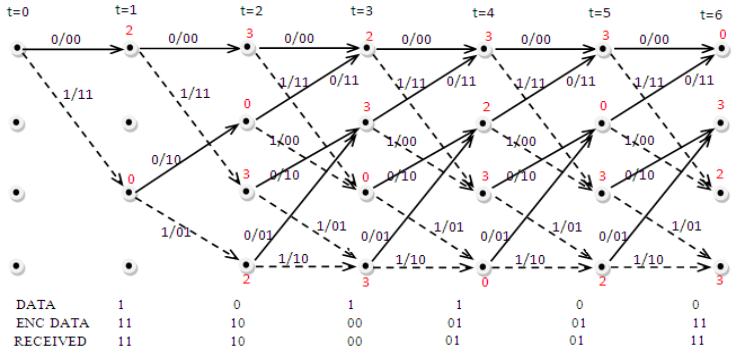


Figure 7: Trellis diagram

The Viterbi decoder, based on Maximum Likelihood Decoding, is a key component in communication systems for decoding convolutionally encoded signals. It employs the Viterbi algorithm to determine the most likely sequence of transmitted symbols given the received signal, using a trellis diagram representation. This algorithm effectively performs maximum likelihood sequence estimation, which aims to find the most probable transmitted sequence among all possible sequences.

In brief, the Viterbi decoder processes the received signal by comparing it with all possible transmitted sequences represented in the trellis diagram. It computes the likelihood of each possible sequence by evaluating the compatibility between the received signal and the expected signal for each transmitted symbol. This compatibility is typically measured using metrics such as Hamming distance or Euclidean distance.

As the decoder traverses through the trellis diagram, it accumulates the likelihood metrics for each possible path. At each step, the decoder selects the most likely path based on the accumulated metrics, effectively performing a dynamic programming approach to find the optimal path through the trellis.

Once the traversal is complete, the Viterbi decoder outputs the decoded sequence corresponding to the most likely path. This decoded sequence represents the estimated transmitted symbols, which can then be further processed or utilized in subsequent stages of the communication system.

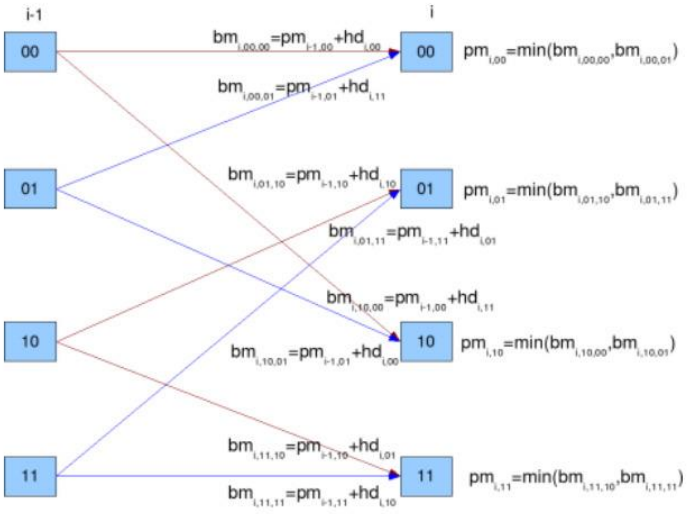


Figure 8: Maximum likelihood decoding

Overall, the Viterbi decoder using Maximum Likelihood Decoding is a powerful technique for efficiently decoding convolutionally encoded signals, enabling reliable communication in various systems such as wireless communication, digital broadcasting, and satellite communication. Its ability to perform maximum likelihood sequence estimation makes it particularly effective in mitigating errors introduced during transmission, thereby enhancing the overall reliability and performance of communication systems.

6.2 APPLICATIONS OF VITERBI DECODER

The Viterbi decoder finds widespread application across various communication systems and digital data storage devices, where error-correction coding techniques like convolutional encoding are crucial for enhancing data reliability in noisy environments. In wireless communication systems such as GSM, WiMAX, LTE, and satellite communication, Viterbi decoders play a pivotal role in decoding convolutionally encoded signals, ensuring reliable data transmission over wireless channels despite channel impairments and noise. Similarly, in digital broadcasting standards like DVB and ATSC, Viterbi decoders are instrumental in decoding digital audio and video signals, ensuring high-quality reception even in the presence of interference. Moreover, in data storage systems such as hard disk drives, optical discs, and flash memory devices, Viterbi decoders contribute to achieving high data reliability and storage density by correcting errors introduced during storage and retrieval processes. Additionally, in satellite communication applications like remote sensing and GPS, Viterbi decoders enable reliable communication over long distances and in harsh environmental conditions. Furthermore, in DSL modems, Viterbi decoders facilitate robust data transmission rates over copper telephone lines, enabling high-speed internet access for residential and commercial users. Overall, Viterbi decoders play a critical role in ensuring reliable and efficient transmission of digital data across a wide range of communication and data storage applications, mitigating the impact of noise, interference, and channel impairments.

CHAPTER 7

METHODOLOGY

To implement two-stage communication system , follow these general steps:

1. Selection of Desired Encoding Schemes:

- Conduct a thorough review of various encoding schemes, considering options such as Pulse Code Modulation (PCM) and Convolutional Encoding.

- Evaluate each scheme's advantages and drawbacks in terms of efficiency, error correction capabilities, and suitability for real-time processing.

- Based on the evaluation, select the most appropriate encoding scheme for the project.

2. Research:

- Dive into detailed research on the chosen encoding scheme, understanding its fundamental principles and how it operates.

- Examine existing literature, standards, and implementations to gather insights into best practices and potential challenges.

- Develop a clear understanding of block diagrams associated with the chosen encoding scheme, identifying key components and their interconnections.

3. System Design:

- Utilize the gathered information to create a comprehensive system design, incorporating block diagrams and algorithms.

- Clearly define the functions of each block in the system, emphasizing the flow of data and control between different components.

- Identify parameters critical to the system's efficiency, ensuring that the design addresses all essential aspects of the chosen encoding scheme.

4. Verilog Coding:

- Translate the designed system into Verilog code, adopting a modular approach to represent each block in the block diagram with corresponding Verilog modules.

- Establish clear interfaces between modules, facilitating efficient communication and data flow.

- Implement coding practices that ensure readability, maintainability, and adherence to the requirements outlined in the system design.

5. FPGA Implementation:

- Utilize tools such as Xilinx ISE 14.7 to synthesize the Verilog code and implement it onto an FPGA development board.

- Consider the specific features and constraints of the chosen FPGA platform during the implementation process.

- Verify that the synthesized hardware accurately represents the intended functionality of the system.

6. System Testing:

- Conduct comprehensive testing post-implementation to validate the performance of the hardware system.

- Evaluate error correction capabilities by introducing controlled errors and assessing the system's ability to detect and correct them.

- Test data transmission integrity, ensuring that the encoded data is transmitted accurately.

- Verify the accuracy of analog-to-digital and digital-to-analog conversions, comparing the input and output signals.

7. Documentation:

- Document the entire methodology, including the rationale behind the selection of encoding schemes, research findings, system design details, Verilog code explanations, FPGA implementation steps, and testing procedures.

- Provide clear and concise documentation to facilitate future understanding, troubleshooting, and potential further development of the project.

7.1 VERILOG CODE

(a) CONVOLUTIONAL ENCODER

```
module convolution(x,clk,reset,z);
input x,clk,reset;
output reg [0:1]z;
parameter [0:1]s0=2'b00;
parameter [0:1]s1=2'b10;
parameter[0:1]s2=2'b01;
parameter[0:1]s3=2'b11;
reg [0:1] ps,ns;
always @(posedge clk or posedge reset)
begin
if (reset)
begin
z=2'b00;
ps=s0;
ns=s0;

end
else
ps=ns;
case (ps)
s0: if (x==1'b1) begin
ns=s1;
z = 2'b11;
end
else begin
z=2'b00;
ns=s0;
end
s1: if (x==1'b1) begin
z=2'b01;
ns=s3;
end
else begin
z=2'b10;
ns=s2;
end
s2: if (x==1'b1) begin
ns=s1;
z=2'b00;
end
else begin
z=2'b11;
ns=s0;
end
s3: if (x==1'b1) begin
ns=s3;
```

```
z=2'b10;
end
else begin
ns=s2;
z=2'b01;
end
endcase
end
endmodule
```

(b) PCM ENCODER

```
module pcmencoder(pcmeip,clk,pcmeop);
   input [11:0]pcmeip;
   input clk;
   output reg[7:0]pcmeop;

always @(posedge clk)
   begin

      if (pcmeip>=12'd0 && pcmeip<=12'd2)
         pcmeop=8'b00000000;
      else if (pcmeip<=12'd4)
         pcmeop=8'b00000001;
      else if (pcmeip<=12'd6)
         pcmeop=8'b00000010;
      else if (pcmeip<=12'd8)
         pcmeop=8'b00000011;
      else if (pcmeip<=12'd10)

pcmeop=8'b000001001;
      else if (pcmeip<=12'd12)
         pcmeop=8'b00000101;
      else if (pcmeip<=12'd14)
         pcmeop=8'b00000110;
      else if (pcmeip<=12'd16)
         pcmeop=8'b00000111;
      else if (pcmeip<=12'd18)
         pcmeop=8'b00001000;
      else if (pcmeip<=12'd20)
         pcmeop=8'b00001001;
      else if (pcmeip<=12'd22)
         pcmeop=8'b00001010;
      else if (pcmeip<=12'd24)
         pcmeop=8'b00001011;
      else if (pcmeip<=12'd26)
         pcmeop=8'b00001100;
      else if (pcmeip<=12'd28)
         pcmeop=8'b00001101;
      else if (pcmeip<=12'd30)
         pcmeop=8'b00001110;
      else if (pcmeip<=12'd32)
         pcmeop=8'b00001111;

      else if (pcmeip<=12'd34)
         pcmeop=8'b00010000;
      else if (pcmeip<=12'd36)
         pcmeop=8'b00010001;
      else if (pcmeip<=12'd38)
         pcmeop=8'b00010010;
      else if (pcmeip<=12'd40)
         pcmeop=8'b00010011;
      else if (pcmeip<=12'd42)
         pcmeop=8'b00010100;
      else if (pcmeip<=12'd44)
         pcmeop=8'b00010101;
      else if (pcmeip<=12'd46)
         pcmeop=8'b00010110;
      else if (pcmeip<=12'd48)
         pcmeop=8'b00010111;
      else if (pcmeip<=12'd50)
         pcmeop=8'b00011000;
      else if (pcmeip<=12'd52)
         pcmeop=8'b00011001;
      else if (pcmeip<=12'd54)
         pcmeop=8'b00011010;
      else if (pcmeip<=12'd56)
         pcmeop=8'b00011011;
      else if (pcmeip<=12'd58)
```

```
    pcmeop=8'b00011100;
else if (pcmeip<=12'd60)
    pcmeop=8'b00011101;
else if (pcmeip<=12'd62)
    pcmeop=8'b00011110;
else if (pcmeip<=12'd64)
    pcmeop=8'b00011111;


else if (pcmeip<=12'd68)
    pcmeop=8'b00100000;
else if (pcmeip<=12'd72)
    pcmeop=8'b00100001;
else if (pcmeip<=12'd76)
    pcmeop=8'b00100010;
else if (pcmeip<=12'd80)
    pcmeop=8'b00100011;
else if (pcmeip<=12'd84)
    pcmeop=8'b00100100;
else if (pcmeip<=12'd88)
    pcmeop=8'b00100101;
else if (pcmeip<=12'd92)
    pcmeop=8'b00100110;
else if (pcmeip<=12'd96)
    pcmeop=8'b00100111;
        else if
(pcmeip<=12'd100)
            pcmeop=8'b00101000;
        else if
(pcmeip<=12'd104)
            pcmeop=8'b00101001;
        else if
(pcmeip<=12'd108)
            pcmeop=8'b00101010;
        else if
(pcmeip<=12'd112)
            pcmeop=8'b00101011;
        else if
(pcmeip<=12'd116)
            pcmeop=8'b00101100;
        else if
(pcmeip<=12'd120)
            pcmeop=8'b00101101;
        else if
(pcmeip<=12'd124)
            pcmeop=8'b00101110;
        else if
(pcmeip<=12'd128)
            pcmeop=8'b00101111;


        else if
(pcmeip<=12'd136)
            pcmeop=8'b00110000;
        else if
(pcmeip<=12'd144)
            pcmeop=8'b00110001;
        else if
(pcmeip<=12'd152)
            pcmeop=8'b00110010;
        else if
(pcmeip<=12'd160)
            pcmeop=8'b00110011;
        else if
(pcmeip<=12'd168)
            pcmeop=8'b00110100;
        else if
(pcmeip<=12'd176)
            pcmeop=8'b00110101;
        else if
(pcmeip<=12'd184)
            pcmeop=8'b00110110;
        else if
(pcmeip<=12'd192)
            pcmeop=8'b00110111;
        else if
(pcmeip<=12'd200)
            pcmeop=8'b00111000;
        else if
(pcmeip<=12'd208)
            pcmeop=8'b00111001;
        else if
(pcmeip<=12'd216)
            pcmeop=8'b00111010;
        else if
(pcmeip<=12'd224)
```

```
        pcmeop=8'b00111011;
    else if (pcmeip<=12'd232)
        pcmeop=8'b00111100;
    else if (pcmeip<=12'd240)
        pcmeop=8'b00111101;
    else if (pcmeip<=12'd248)
        pcmeop=8'b00111110;
    else if (pcmeip<=12'd256)
        pcmeop=8'b00111111;
    else if (pcmeip<=12'd272)
        pcmeop=8'b01000000;
    else if (pcmeip<=12'd288)
        pcmeop=8'b01000001;
    else if (pcmeip<=12'd304)
        pcmeop=8'b01000010;
    else if (pcmeip<=12'd320)
        pcmeop=8'b01000011;
    else if (pcmeip<=12'd336)
        pcmeop=8'b01000100;
    else if (pcmeip<=12'd352)
        pcmeop=8'b01000101;
    else if (pcmeip<=12'd368)
        pcmeop=8'b01000110;
    else if (pcmeip<=12'd384)
        pcmeop=8'b01000111;
    else if (pcmeip<=12'd400)
        pcmeop=8'b01001000;
    else if (pcmeip<=12'd416)
        pcmeop=8'b01001001;
    else if (pcmeip<=12'd432)
        pcmeop=8'b01001010;
    else if (pcmeip<=12'd448)
        pcmeop=8'b01001011;
    else if (pcmeip<=12'd464)
        pcmeop=8'b01001100;
    else if (pcmeip<=12'd480)
        pcmeop=8'b01001101;
    else if (pcmeip<=12'd496)
        pcmeop=8'b01001110;
    else if (pcmeip<=12'd512)
        pcmeop=8'b01001111;
    else if (pcmeip<=12'd544)
        pcmeop=8'b01010000;
    else if (pcmeip<=12'd576)
        pcmeop=8'b01010001;
    else if (pcmeip<=12'd608)
        pcmeop=8'b01010010;
    else if (pcmeip<=12'd640)
        pcmeop=8'b01010011;
    else if (pcmeip<=12'd672)
        pcmeop=8'b01010100;
    else if (pcmeip<=12'd704)
        pcmeop=8'b01010101;
    else if (pcmeip<=12'd736)
        pcmeop=8'b01010110;
    else if (pcmeip<=12'd768)
        pcmeop=8'b01010111;
    else if (pcmeip<=12'd800)
```

```
        pcmeop=8'b01011000;
    else if
(pcmeip<=12'd832)
        pcmeop=8'b01011001;
    else if
(pcmeip<=12'd864)
        pcmeop=8'b01011010;
    else if
(pcmeip<=12'd896)
        pcmeop=8'b01011011;
    else if
(pcmeip<=12'd928)
        pcmeop=8'b01011100;
    else if
(pcmeip<=12'd960)
        pcmeop=8'b01011101;
    else if
(pcmeip<=12'd992)
        pcmeop=8'b01011110;
    else if
(pcmeip<=12'd1024)
        pcmeop=8'b01011111;



    else if
(pcmeip<=12'd1088)
        pcmeop=8'b01100000;
    else if
(pcmeip<=12'd1152)
        pcmeop=8'b01100001;
    else if
(pcmeip<=12'd1216)

pcmeop=8'b011000101;
    else if
(pcmeip<=12'd1280)
        pcmeop=8'b01100011;
    else if
(pcmeip<=12'd1344)
        pcmeop=8'b01100100;
    else if
(pcmeip<=12'd1408)
        pcmeop=8'b01100101;
    else if
(pcmeip<=12'd1472)
        pcmeop=8'b01100110;
    else if
(pcmeip<=12'd1536)
        pcmeop=8'b01100111;
    else if
(pcmeip<=12'd1600)
        pcmeop=8'b01101000;
    else if
(pcmeip<=12'd1664)
        pcmeop=8'b01101001;
    else if
(pcmeip<=12'd1728)
        pcmeop=8'b01101010;
    else if
(pcmeip<=12'd1792)
        pcmeop=8'b01101011;
    else if
(pcmeip<=12'd1856)
        pcmeop=8'b01101100;
    else if
(pcmeip<=12'd1920)
        pcmeop=8'b01101101;
    else if
(pcmeip<=12'd1984)
        pcmeop=8'b01101110;
    else if
(pcmeip<=12'd2048)
        pcmeop=8'b01101111;



    else if
(pcmeip<=12'd2176)
        pcmeop=8'b01110000;
    else if
(pcmeip<=12'd2304)
        pcmeop=8'b01110001;
    else if
(pcmeip<=12'd2432)
        pcmeop=8'b01110010;
    else if
(pcmeip<=12'd2560)
        pcmeop=8'b01110011;
    else if
(pcmeip<=12'd2688)
        pcmeop=8'b01110100;
```

```
    else if (pcmeip<=12'd2816)
        pcmeop=8'b01110101;
    else if (pcmeip<=12'd2944)
        pcmeop=8'b01110110;
    else if (pcmeip<=12'd3072)
        pcmeop=8'b01110111;
    else if (pcmeip<=12'd3200)
        pcmeop=8'b01111000;
    else if (pcmeip<=12'd3328)
        pcmeop=8'b01111001;
    else if (pcmeip<=12'd3456)
        pcmeop=8'b01111010;
    else if (pcmeip<=12'd3584)
        pcmeop=8'b01111011;
    else if (pcmeip<=12'd3712)
        pcmeop=8'b01111100;
    else if (pcmeip<=12'd3840)
        pcmeop=8'b01111101;
    else if (pcmeip<=12'd3968)
        pcmeop=8'b01111110;
    else if (pcmeip<=12'd4095)
        pcmeop=8'b01111111;
  end
endmodule
```

(c) PCM DECODER

```
module pcmdecoder(pcmdip,clk,pcmdop);
   input [7:0]pcmdip;
 input clk;
   output reg [11:0]pcmdop;

   always @(posedge clk)
      begin

         if (pcmdip==8'b00000000)
            pcmdop=0;
         else if (pcmdip==8'b00000001)
            pcmdop=3;
         else if (pcmdip==8'b00000010)
            pcmdop=5;
         else if(pcmdip==8'b00000011)
            pcmdop=9;
         else if(pcmdip==8'b00000101)
            pcmdop=11;
         else if(pcmdip==8'b00000110)
            pcmdop=13;
         else if(pcmdip==8'b00000111)
            pcmdop=15;
         else if(pcmdip==8'b00001000)
            pcmdop=17;
         else if(pcmdip==8'b00001001)
            pcmdop=19;
         else if(pcmdip==8'b00001010)
            pcmdop=21;
         else if(pcmdip==8'b00001011)
            pcmdop=23;
         else if(pcmdip==8'b00001011)
            pcmdop=25;
         else if(pcmdip==8'b00001100)
            pcmdop=27;
         else if(pcmdip==8'b00001101)
            pcmdop=29;
         else if(pcmdip==8'b00001110)
            pcmdop=31;
         else if(pcmdip==8'b00001111)
            pcmdop=32;
         else if(pcmdip==8'b00010000)
            pcmdop=33;
         else if(pcmdip==8'b00010001)
            pcmdop=35;
         else if(pcmdip==8'b00010010)
            pcmdop=37;
         else if(pcmdip==8'b00010011)
            pcmdop=39;
         else if(pcmdip==8'b00010100)
            pcmdop=41;
         else if(pcmdip==8'b00010101)
            pcmdop=43;
         else if(pcmdip==8'b00010110)
            pcmdop=45;
         else if(pcmdip==8'b00010111)
            pcmdop=47;
         else if(pcmdip==8'b00011000)
            pcmdop=49;
         else if(pcmdip==8'b00011001)
            pcmdop=51;
         else if(pcmdip==8'b00011010)
            pcmdop=53;
         else if(pcmdip==8'b00011011)
            pcmdop=55;
         else if(pcmdip==8'b00011100)
            pcmdop=57;
         else if(pcmdip==8'b00011101)
            pcmdop=59;
         else if(pcmdip==8'b00011110)
            pcmdop=61;
         else if(pcmdip==8'b00011111)
            pcmdop=63;

         else if(pcmdip==8'b00100000)
            pcmdop=66;
         else if(pcmdip==8'b00100001)
            pcmdop=70;
         else if(pcmdip==8'b00100010)
            pcmdop=74;
         else if(pcmdip==8'b00100011)
            pcmdop=78;
         else if(pcmdip==8'b00100100)
            pcmdop=82;
         else if(pcmdip==8'b00100101)
            pcmdop=86;
         else if(pcmdip==8'b00100110)
            pcmdop=90;
         else if(pcmdip==8'b00100111)
```

```
    pcmdop=94;
else if(pcmdip==8'b00101000)
    pcmdop=98;
else if(pcmdip==8'b00101001)
    pcmdop=102;
else if(pcmdip==8'b00101010)
    pcmdop=106;
else if(pcmdip==8'b00101011)
    pcmdop=110;
else if(pcmdip==8'b00101100)
    pcmdop=114;
else if(pcmdip==8'b00101101)
    pcmdop=118;
else if(pcmdip==8'b00101110)
    pcmdop=122;
else if(pcmdip==8'b00101111)
    pcmdop=126;
else if(pcmdip==8'b00110000)
    pcmdop=132;
else if(pcmdip==8'b00110001)
    pcmdop=140;
else if(pcmdip==8'b00110010)
    pcmdop=148;
else if(pcmdip==8'b00110011)
    pcmdop=156;
else if(pcmdip==8'b00110100)
    pcmdop=164;
else if(pcmdip==8'b00110101)
    pcmdop=172;
else if(pcmdip==8'b00110110)
    pcmdop=180;
else if(pcmdip==8'b00110111)
    pcmdop=188;
else if(pcmdip==8'b00111000)
    pcmdop=196;
else if(pcmdip==8'b00111001)
    pcmdop=204;
else if(pcmdip==8'b00111010)
    pcmdop=212;

else if(pcmdip==8'b00111011)
    pcmdop=220;
else if(pcmdip==8'b00111100)
    pcmdop=228;
else if(pcmdip==8'b00111101)
    pcmdop=236;
else if(pcmdip==8'b00111110)
    pcmdop=244;
else if(pcmdip==8'b00111111)
    pcmdop=252;

else if(pcmdip==8'b01000000)
    pcmdop=264;
else if(pcmdip==8'b01000001)
    pcmdop=280;
else if(pcmdip==8'b01000010)
    pcmdop=296;
else if(pcmdip==8'b01000011)
    pcmdop=312;
else if(pcmdip==8'b01000100)
    pcmdop=328;
else if(pcmdip==8'b01000101)
    pcmdop=344;
else if(pcmdip==8'b01000110)
    pcmdop=360;
else if(pcmdip==8'b01000111)
    pcmdop=376;
else if(pcmdip==8'b01001000)
    pcmdop=392;
else if(pcmdip==8'b01001001)
    pcmdop=408;
else if(pcmdip==8'b01001010)
    pcmdop=424;
else if(pcmdip==8'b01001011)
    pcmdop=440;
else if(pcmdip==8'b01001100)
    pcmdop=456;
else if(pcmdip==8'b01001101)
    pcmdop=472;
else if(pcmdip==8'b01001110)
    pcmdop=488;
else if(pcmdip==8'b01001111)
    pcmdop=504;

else if(pcmdip==8'b01010000)
    pcmdop=528;
else if(pcmdip==8'b01010001)
    pcmdop=560;
else if(pcmdip==8'b01010010)
    pcmdop=592;
else if(pcmdip==8'b01010011)
    pcmdop=624;
```

```
else if(pcmdip==8'b01010100)
   pcmdop=656;
else if(pcmdip==8'b01010101)
   pcmdop=688;
else if(pcmdip==8'b01010110)
   pcmdop=720;
else if(pcmdip==8'b01010111)
   pcmdop=752;
else if(pcmdip==8'b01011000)
   pcmdop=784;
else if(pcmdip==8'b01011001)
   pcmdop=816;
else if(pcmdip==8'b01011010)
   pcmdop=848;
else if(pcmdip==8'b01011011)
   pcmdop=880;
else if(pcmdip==8'b01011100)
   pcmdop=912;
else if(pcmdip==8'b01011101)
   pcmdop=944;
else if(pcmdip==8'b01011110)
   pcmdop=976;
else if(pcmdip==8'b01011111)
   pcmdop=1008;

else if(pcmdip==8'b01100000)
   pcmdop=1056;
else if(pcmdip==8'b01100001)
   pcmdop=1120;
else if(pcmdip==8'b01100010)
   pcmdop=1184;
else if(pcmdip==8'b01100011)
   pcmdop=1248;
else if(pcmdip==8'b01100100)
   pcmdop=1312;
else if(pcmdip==8'b01100101)
   pcmdop=1376;
else if(pcmdip==8'b01100110)
   pcmdop=1440;
else if(pcmdip==8'b01100111)
   pcmdop=1504;
else if(pcmdip==8'b01101000)
   pcmdop=1568;
else if(pcmdip==8'b01101001)
   pcmdop=1632;
else if(pcmdip==8'b01101010)
   pcmdop=1696;
else if(pcmdip==8'b01101011)
   pcmdop=1760;
else if(pcmdip==8'b01101100)
   pcmdop=1824;
else if(pcmdip==8'b01101101)
   pcmdop=1888;
else if(pcmdip==8'b01101110)
   pcmdop=1952;
else if(pcmdip==8'b01101111)
   pcmdop=2016;

else if(pcmdip==8'b01110000)
   pcmdop=2112;
else if(pcmdip==8'b01110001)
   pcmdop=2240;
else if(pcmdip==8'b01110010)
   pcmdop=2368;
else if(pcmdip==8'b01110011)
   pcmdop=2496;
else if(pcmdip==8'b01110100)
   pcmdop=2624;
else if(pcmdip==8'b01110101)
   pcmdop=2752;
else if(pcmdip==8'b01110110)
   pcmdop=2880;
else if(pcmdip==8'b01110111)
   pcmdop=3008;
else if(pcmdip==8'b01111000)
   pcmdop=3136;
else if(pcmdip==8'b01111001)
   pcmdop=3264;
else if(pcmdip==8'b01111010)
   pcmdop=3392;
else if(pcmdip==8'b01111011)
   pcmdop=3520;
else if(pcmdip==8'b01111100)
   pcmdop=3648;
else if(pcmdip==8'b01111101)
   pcmdop=3776;
else if(pcmdip==8'b01111110)
   pcmdop=3904;
else if(pcmdip==8'b01111111)
   pcmdop=4095;
 end
endmodule
```

(d) VITERBI DECODER

```
module ViterbiDecoder (
    input [1:0] data_in,
    input clk,
    output reg data_out
);

// Define types directly without using typedef
// 2-bit word type
logic [1:0] word_2;
// 4-bit next state type
logic [1:0] word_4_NextState [0:3];
// 3-bit word type
logic [1:0] word_3 [0:2];
// 3-bit word as bit type
bit [1:0] word_3_bit [0:2];
// 4-bit word type
bit [3:0] word_4 [0:3];
// 4-bit word as bit type
bit [3:0] word_4_bit [0:3];
// 2-bit memory type
bit [1:0] memory_4 [0:3][0:1];
// 4-bit memory as bit type
bit [3:0] memory_4_bit [0:3][0:3];
// 4-bit memory for next state type
bit [1:0] memory_4_NextState [0:3][0:3];
// 8-bit memory type
bit [7:0] memory_8 [0:7];
// 3-bit memory for traceback row type
bit [1:0] memory_traceback_row [0:7][0:2];
// 3-bit memory for traceback table type
bit [1:0] memory_traceback_table [0:3][0:7][0:2];
// Constants
parameter [1:0][1:0][1:0] traceback_table = {{{2'b00, 2'b00, 2'b00}, {2'b11, 2'b10, 2'b11}, {2'b00, 2'b11,
2'b10}, {2'b11, 2'b01, 2'b01}, {2'b00, 2'b00, 2'b11}, {2'b11, 2'b10, 2'b00}, {2'b00, 2'b11, 2'b01}, {2'b11,
2'b01, 2'b10}},
                                {{2'b11, 2'b00, 2'b00}, {2'b00, 2'b10, 2'b11}, {2'b11, 2'b11, 2'b10}, {2'b00,
2'b01, 2'b01}, {2'b11, 2'b00, 2'b11}, {2'b00, 2'b10, 2'b00}, {2'b11, 2'b11, 2'b01}, {2'b00, 2'b01, 2'b10}},
                                {{2'b10, 2'b11, 2'b00}, {2'b01, 2'b01, 2'b11}, {2'b10, 2'b00, 2'b10}, {2'b01,
2'b10, 2'b01}, {2'b10, 2'b11, 2'b11}, {2'b01, 2'b01, 2'b00}, {2'b10, 2'b00, 2'b01}, {2'b01, 2'b10, 2'b10}},
                                {{2'b01, 2'b11, 2'b00}, {2'b10, 2'b01, 2'b11}, {2'b01, 2'b00, 2'b10}, {2'b10,
2'b10, 2'b01}, {2'b01, 2'b11, 2'b11}, {2'b10, 2'b01, 2'b00}, {2'b01, 2'b00, 2'b01}, {2'b10, 2'b10, 2'b10}}};
parameter [1:0][3:0] outputTable = {{1'b0, 1'b0, 1'b0, 1'b1}, {1'b1, 1'b0, 1'b0, 1'b0}, {1'b0, 1'b1, 1'b0, 1'b0},
{1'b0, 1'b0, 1'b1, 1'b0}};
```

```
parameter [1:0][3:0][1:0] nextStateTable = {{{2'b00, 2'b00, 2'b00, 2'b10}, {2'b10, 2'b00, 2'b00, 2'b00},
{2'b00, 2'b11, 2'b01, 2'b00}, {2'b00, 2'b01, 2'b11, 2'b00}},
                                {{2'b11, 2'b00, 2'b00, 2'b10}, {2'b00, 2'b10, 2'b00, 2'b00}, {2'b11, 2'b11, 2'b10,
2'b00}, {2'b00, 2'b01, 2'b01, 2'b00}},
                                {{2'b10, 2'b11, 2'b00, 2'b10}, {2'b01, 2'b01, 2'b11, 2'b00}, {2'b10, 2'b00, 2'b10,
2'b10}, {2'b01, 2'b10, 2'b01, 2'b10}},
                                {{2'b01, 2'b11, 2'b00, 2'b10}, {2'b10, 2'b01, 2'b11, 2'b00}, {2'b01, 2'b00, 2'b10,
2'b01}, {2'b10, 2'b10, 2'b01, 2'b10}}};

// Function definitions
function integer hammingDistance;
   input [1:0] a;
   begin
      case(a)
         2'b00: hammingDistance = 0;
         2'b01: hammingDistance = 1;
         2'b10: hammingDistance = 1;
         2'b11: hammingDistance = 2;
         default: hammingDistance = -1; // Invalid operation
      endcase
   end
endfunction

function integer conv_int;
   input [1:0] a;
   begin
      case(a)
         2'b00: conv_int = 0;
         2'b01: conv_int = 1;
         2'b10: conv_int = 2;
         2'b11: conv_int = 3;
         default: conv_int = -1; // Invalid operation
      endcase
   end
endfunction

// Registers
reg [1:0] InitialState = 2'b00;
reg [7:0] TracebackResult = 8'b0;
reg [2:0] InputLevel = 3'd0;
reg [2:0] i = 3'd0;
reg [2:0] chosenPathIndex;
reg [2:0] lowestPathMetricError = 3'd6;
reg [1:0] currentState;
reg [2:0] outputVector_index = 3'd0;
reg [1:0] temp_output;
```

```
always @(posedge clk) begin
   if (data_in != 2'bUU) begin
      i <= 3'd0;
      // Branch Metric Calculations
      while (i < 8) begin
         TracebackResult[i] <= TracebackResult[i] + hammingDistance(traceback_table[3-
conv_int(InitialState)][7-i][2-InputLevel] ^ data_in);
         i <= i + 3'd1;
      end
      // Output the decoded data, delayed for 3 clock cycles
      data_out <= outputVector[outputVector_index];
      InputLevel <= InputLevel + 3'd1;
      if (InputLevel == 3'd3) begin
         // Select the correct path with the lowest path metric error
         i <= 3'd0;
         while (i < 8) begin
            if (lowestPathMetricError > TracebackResult[i]) begin
               lowestPathMetricError <= TracebackResult[i];
               chosenPathIndex <= i;
            end
            i <= i + 3'd1;
         end
         // Convert the selected path to corresponding output
         currentState <= InitialState;
         i <= 3'd0;
         while (i < 3) begin
            temp_output <= traceback_table[3-conv_int(InitialState)][7-chosenPathIndex][2-i];
            outputVector[outputVector_index] <= outputTable[3-conv_int(currentState)][3-
conv_int(temp_output)];
            currentState <= nextStateTable[3-conv_int(currentState)][3-conv_int(temp_output)];
            i <= i + 3'd1;
         end
         // Set the initial state of the next stage
         InitialState <= currentState;
         // Reset variables
         InputLevel <= 3'd0;
         for (i = 0; i < 8; i = i + 3'd1) begin
            TracebackResult[i] <= 8'b0;
         end
         lowestPathMetricError <= 3'd6;
         outputVector_index <= outputVector_index + 3'd1;
      end
   end
end
endmodule
```

CHAPTER 8
RESULTS AND DISCUSSION

In this section, the results obtained by the implementation of source encoder using Xilinx ISE 14.7 is shown below:

The encoded output for a input bit stream is achieved with the convolution encoder.

Figure 9: Output for an input of 001110110111

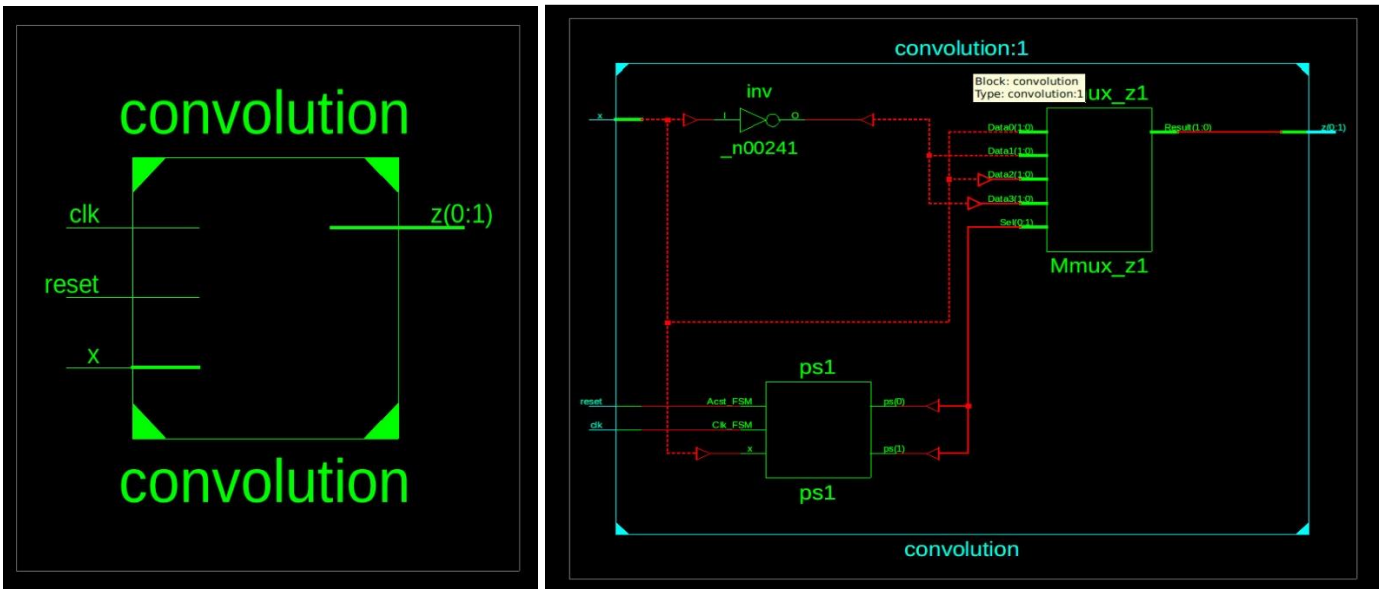


Figure 10: RTL Diagram of a convolutional encoder

2. PCM ENCODER

In this section, the results obtained by the implementation of PCM Encoder using Xilinx ISE 14.7 is shown below:

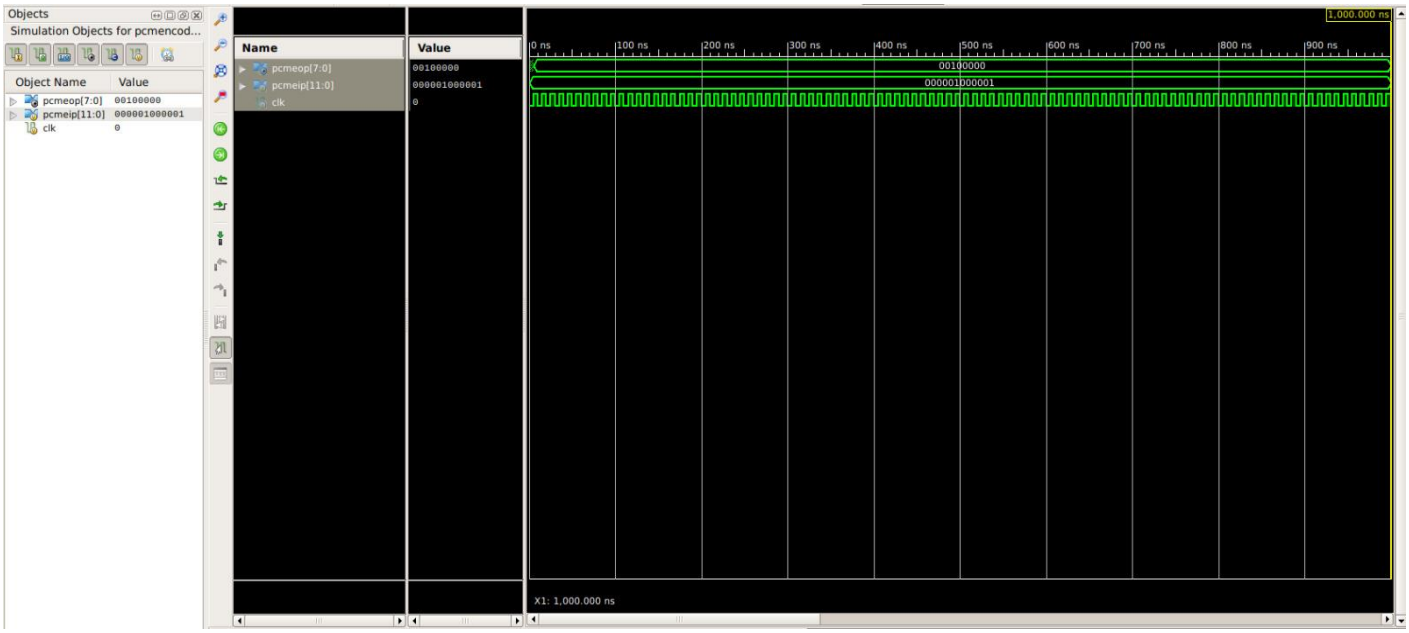


Figure 11: Output for an input of 000001000001

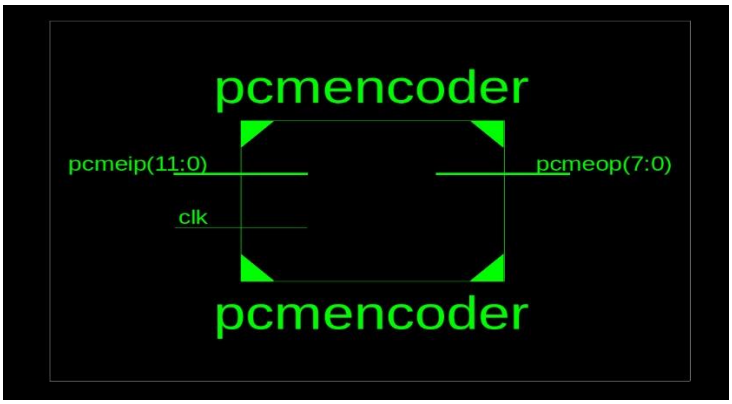
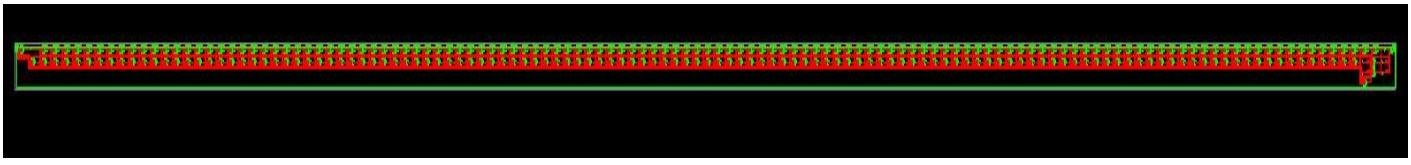
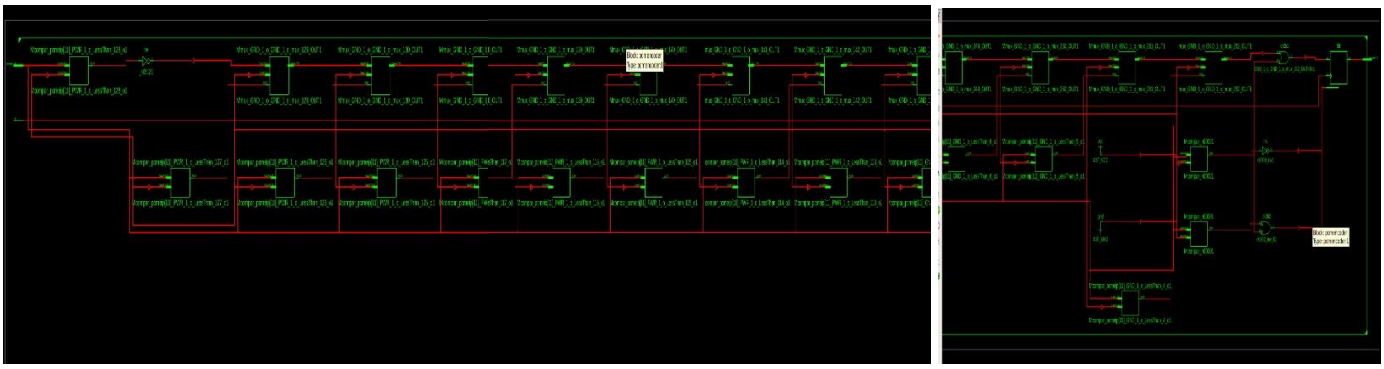


Figure 12: RTL diagram





3. PCM DECODER

In this section, the results obtained by the implementation of PCM Decoder using Xilinx ISE 14.7 is shown below:

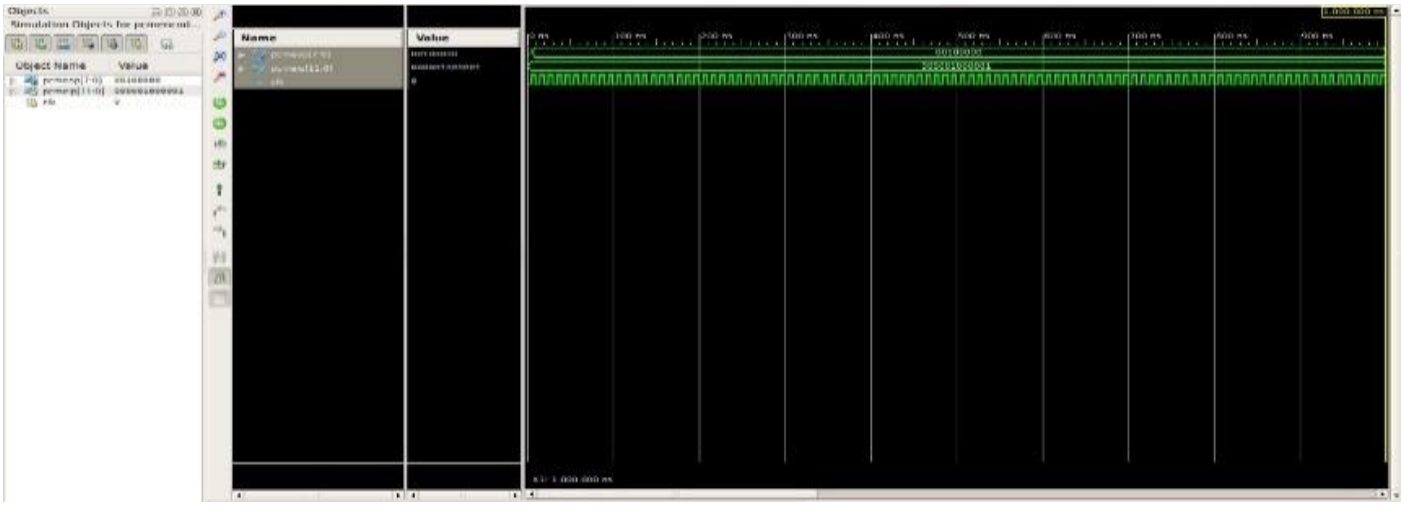


Figure 13: Output for an input of 01000101

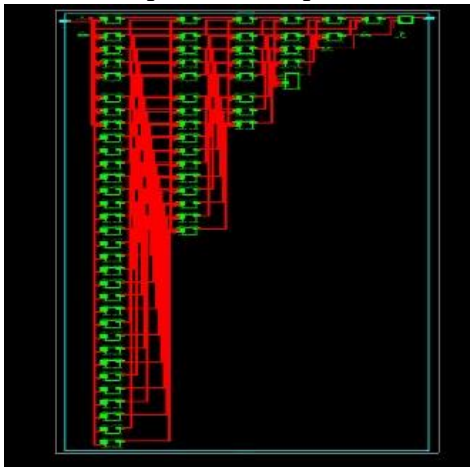
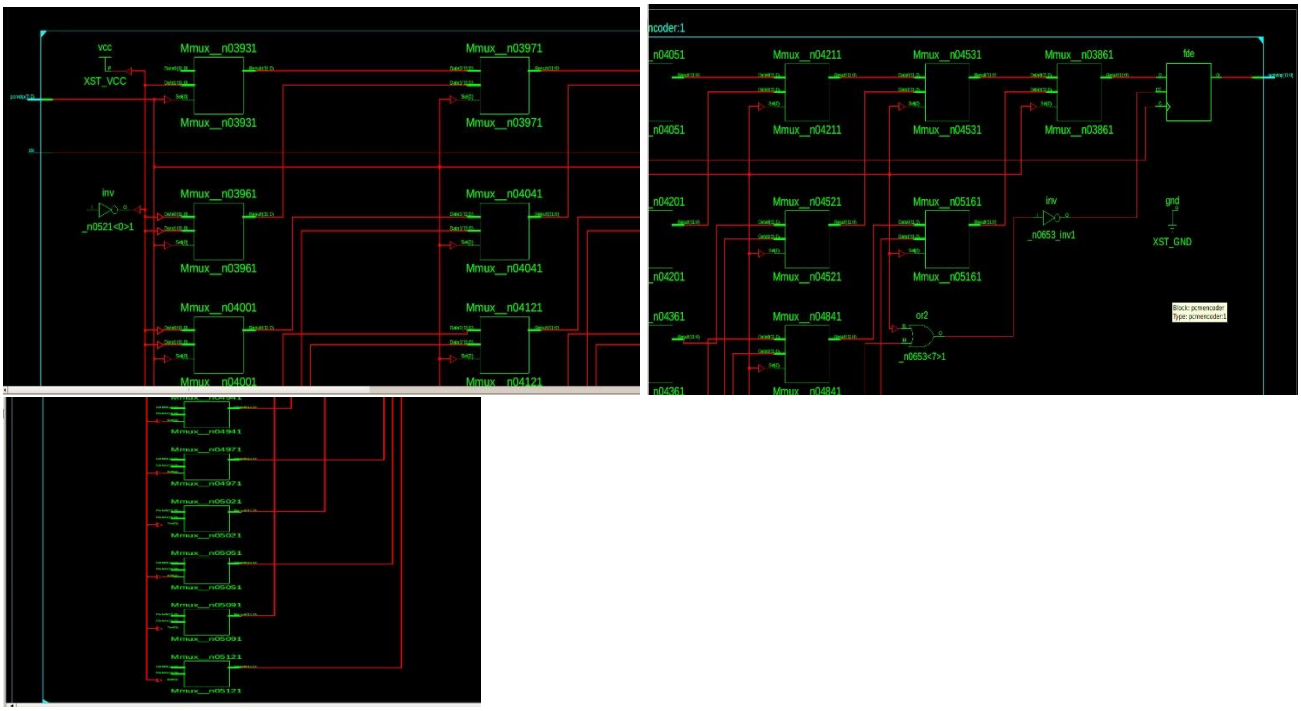


Figure 14: RTL Diagram



4. VITERBI DECODER

In this section, the results obtained by the implementation of Viterbi Decoder using Xilinx ISE 14.7 is shown below:

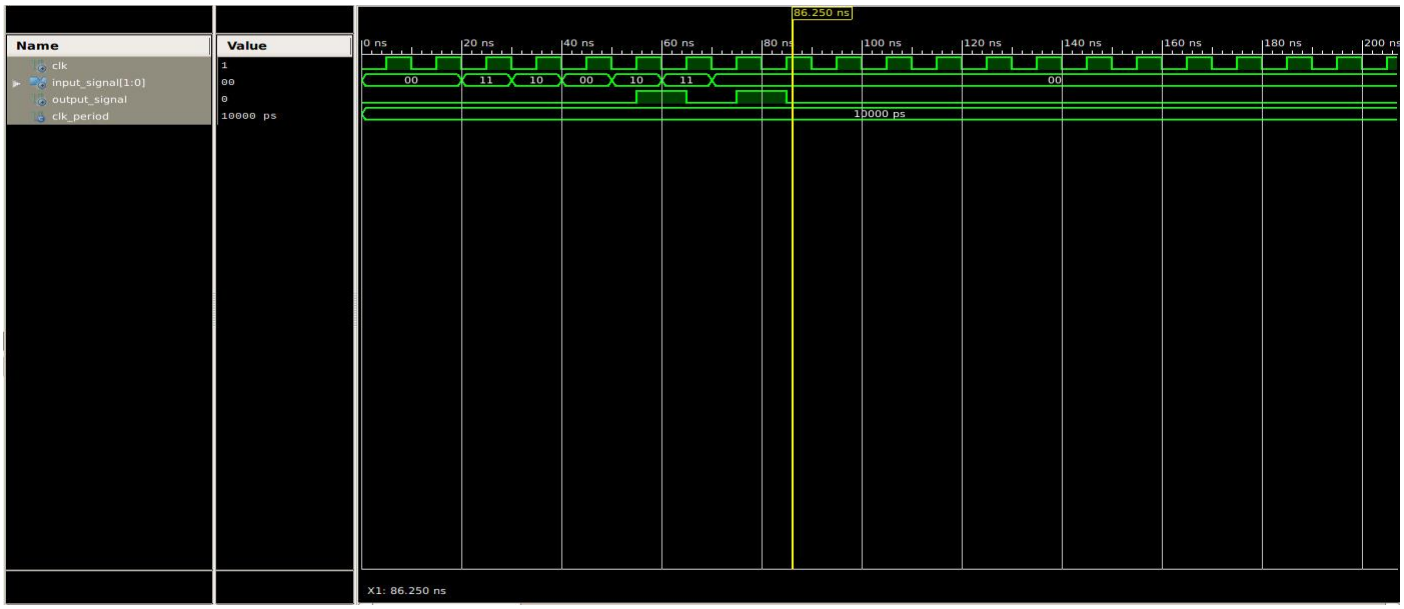


Figure 15: Output for an input of 001110001011

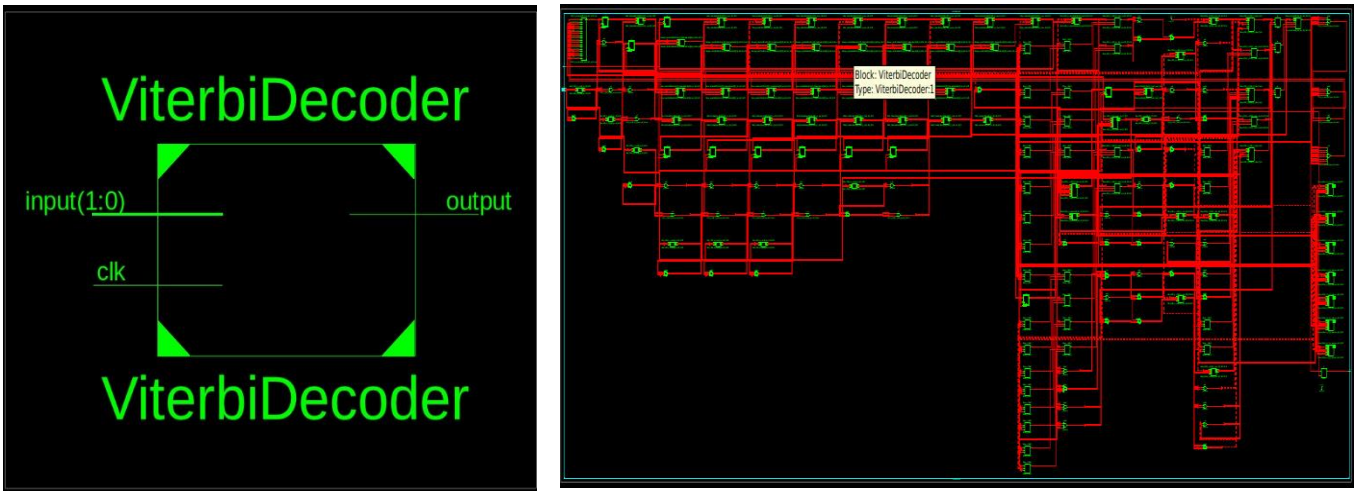


Figure 16: RTL Diagram

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

In conclusion, the study has underscored the rapid growth of error control coding applications in diverse fields of communication and information storage. Various error correction techniques based on applied mathematics have been explored, each with its own set of limitations, whether in mathematical, practical considerations, or other aspects. While it remains impossible to correct all errors, the focus has shifted towards minimizing errors. Challenges arise when attempting to design codes that efficiently correct an increased number of errors, highlighting the need for practical solutions. Although small error correction codes with desired capabilities can be developed easily, creating codes with substantial error correction capability poses a significant practical challenge. The implemented convolutional encoder of constraint length 3 bits and rate ½ using Verilog and Xilinx ISE Design Suite 14.3 has been cross-verified for its functionality under various error scenarios.The study presents a foundation for future work in the realm of error control coding, offering several avenues for exploration and improvement. One potential avenue is the consideration of additional performance evaluation parameters, as outlined in Chapter 4, to enhance the overall effectiveness of error correction codes. Additionally, there is scope for increasing the constraint length and developing new low code rate convolutional codes, potentially extending their applicability to various fields. Another avenue involves exploring the combination of two block codes and studying their synergistic effects. The quest for more perfect and large Hamming distance codes is an ongoing pursuit for improved error correction capabilities.

Looking ahead, future work could also focus on addressing the challenges of developing error correcting codes capable of correcting errors with high probability. Advancements in cellular services, particularly in higher generation services, can be explored to improve quality by incorporating low code rates with high data rates. Furthermore, there is an opportunity to enhance the quality of real-time data transmission in space communication through systematic studies.

CHAPTER 10

REFERENCES

1. V. Kavinilavu, S. Salivahanan, V. S. K. Bhaaskaran, S. Sakthikumaran, B. Brindha and C. Vinoth,

   "Implementation of Convolutional encoder and Viterbi decoder using Verilog HDL," *2011 3rd International Conference on Electronics Computer Technology*, Kanyakumari, India, 2011, pp. 297300, doi: 10.1109/ICECTECH.2011.5941609.

2. G. S. Suganya and G. Kavya, "RTL design and VLSI implementation of an efficient convolutional encoder and adaptive Viterbi decoder," *2013 International Conference on Communication and Signal Processing*, Melmaruvathur, India, 2013, pp. 494-498, doi: 10.1109/iccsp.2013.6577103.

3. K. Venusamy, S. Kannadhasan, A. Vimalesh, P. Chandramohan, M. Shanmugam and K. Priyadarsini, "Execution of Convolution Coding Method for FPGA in Industrial Automation using VERILOG HDL," *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, 2023, pp. 1496-1500, doi: 10.1109/ICCMC56507.2023.10084251.

4. G. Sadhukhan, M. Sandhu and R. P. Singh, "VLSI based implementation of PCM MUX encoder for range telemetry system," *Proceedings of the IEEE INDICON 2004. First India Annual Conference, 2004.*, Kharagpur, India, 2004, pp. 23-26, doi: 10.1109/INDICO.2004.1497698.

5. C. E. Shannon, “A mathematical theory of communication,” Bell Sys. Tech. J., vol. 27, pp. 379–423 and 623–656, 1948. [2] R. W. Hamming, “Error detecting and correcting codes,” Bell Sys. Tech. J., vol. 29, pp. 147–160, 1950.

6. Irfan Habib, Özgün Paker, Sergei Sawitzki, “Design Space Exploration of Hard-Decision Viterbi Decoding: Algorithm and VLSI Implementation” IEEE Tran. On Very Large Scale Integration (VLSI) Systems, Vol. 18, Pp. 794-807, May 2010

7. Hema.S, Suresh Babu.V and Ramesh.P, “FPGA Implementation of Viterbi decoder” proceedings of the 6th WSEAS ICEHWOC,Feb. 2007.

PO MAPPING

TITLE	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12
Design and Simulation of Delta Sigma ADC Modulator	3	3	3	3	3	1	0	3	3	3	1	3

PO1 Engineering Knowledge: The project was successfully executed by applying communication, information theory, and Analog to Digital converters knowledge.

PO2 Problem Analysis: The objective was to enhance SNR and other parameters for low-frequency applications in Second Order delta sigma modulators.

PO3 Design/Development of solutions: A new second order modulator topology was suggested, which exhibited improved SNR and other parameters.

PO4 Conduct investigations of complex problems: A comparative analysis was done for six different existing topologies of Second Order Delta Sigma ADC and their parameter values were determined. **PO5 Modern tool usage**: MATLAB SIMULINK was utilized as the simulation tool.

PO6 The Engineer and Society: The future application of this project involves the analysis of brainsignals.

PO7 Environment and Sustainability: This project is software-based, promoting environmental sustainability.

PO8 Ethics: The work is completely original and free from any instances of plagiarism. All external content used is appropriately referenced.

PO9 Individual and team work: The project was completed through the collective effort and hard work of all team members.

PO10 Communication: The project idea and its novelty were effectively communicated.

PO11 Project Management and Finance: The project was efficiently completed by successfully managing time through the development of a timeline and strict adherence to it. The licensed version of MATLAB was employed for this project.

PO12 Lifelong learning: Successful completion of the project involved a detailed study about various topologies , proposing a novel topology and gaining familiarity with the simulation tools used, through a self-learning process.