Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

Department of Electronics and Communication Engineering

III IA EVALUATION REPORT

for

BLENDED LEARNING PROJECT BASED LEARNING

FPGA-Based Implementation of Unipolar Line Coding

A report submitted by

Name of the Students	*ANNIE REACHEL A, KERLIN E*
Register Numbers	*URK22EC1051, URK22EC1071*
Subject Name	*DIGITAL COMMUNICATION*
Subject Code	*22EC2016*
Date of Report submission	*09/11/2024*

Total marks: _____/ 40 Marks

Signature of Faculty with date:

FPGA-BASED IMPLEMENTATION OF UNIPOLAR LINE CODING USING VERILOG

a project report submitted by

ANNIE REACHEL A URK22EC1051
KERLIN E URK22EC1071

in partial fulfillment for the award of the degree
of

BACHELOR OF TECHNOLOGY

For

DIGITAL COMMUNICATION

– PROJECT BASED LEARNING (22EC2016)

under the supervision of

Dr. S. Merlin Gilbert Raj

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

KARUNYA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Deemed-to-be-University)

Karunya Nagar, Coimbatore - 641 114. INDIA

November 2024

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"FPGA-Based Implementation of Unipolar Line Coding Using Verilog"** is the bonafide work of **"Annie Reachel A (URK22EC1051) and Kerlin E (URK22EC1071)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
DR. JUDE HEMANTH	DR. S. MERLIN GILBERT RAJ
HEAD OF THE DEPARTMENT	**SUPERVISOR**
PROFESSOR	ASSISTANT PROFESSOR
Department of ECE	Department of ECE
School of E&T	School of E&T

Submitted for the III Internal-Project Based Learning Viva Voce held on 09.11.2024.

Internal Examiner

Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

This project focuses on the implementation of unipolar line coding using a Field Programmable Gate Array (FPGA) and Verilog, illustrating the capabilities of modern digital design techniques. Unipolar line coding is a fundamental digital signaling method that employs only positive voltage levels to transmit binary data. Initially utilized over physical media such as copper wires, this coding scheme requires minimal signal processing, making it straightforward to implement.

However, unipolar line coding has inherent limitations, including the absence of a built-in clock signal for synchronization. This lack of synchronization can lead to errors in long sequences of identical bits, particularly zeros, as the method is susceptible to baseline drift. These challenges prompted the evolution of more sophisticated coding techniques, such as Manchester and bipolar coding, which address synchronization issues and enhance data integrity.

The project also highlights the significance of FPGAs in modern digital design. Invented by Ross Freeman, FPGAs offer unparalleled flexibility, allowing designers to implement complex digital circuits that can be reconfigured to accommodate different applications. Xilinx introduced the first commercially viable FPGA in 1985, revolutionizing the field of digital electronics.

With their ability to perform parallel processing for real-time tasks, FPGAs are ideal for applications requiring high-speed processing and adaptability. By implementing unipolar line coding on an FPGA using Verilog, this project demonstrates an efficient hardware simulation of coding schemes, showcasing how traditional techniques can be enhanced through contemporary technology.

CHAPTER 2

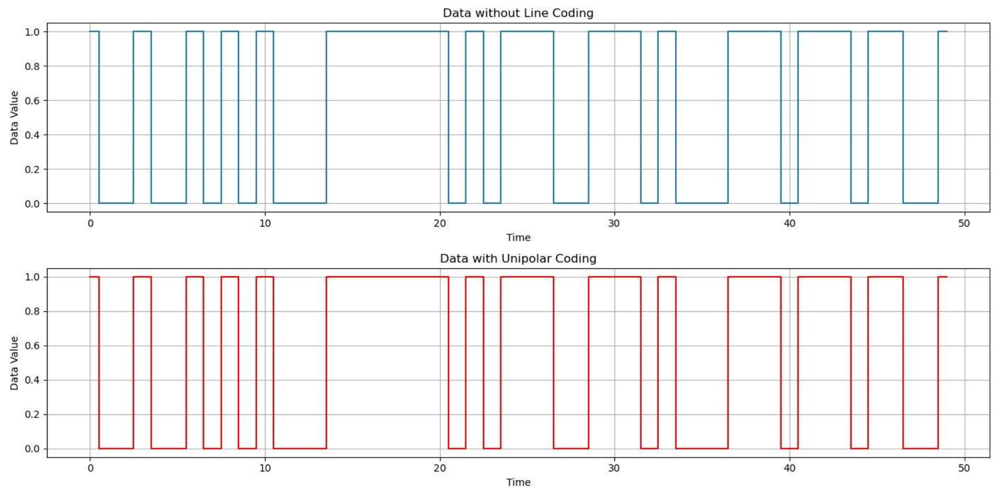
PROBLEM STATEMENT

- **Problem Statement**

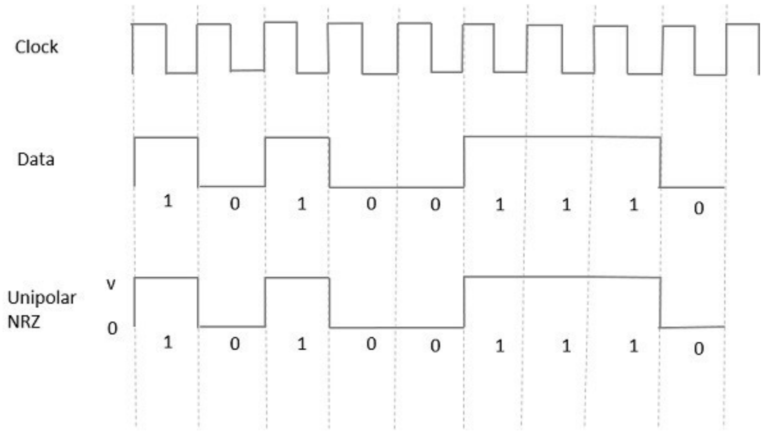
Unipolar line coding is a straightforward yet effective method for digital data transmission that utilizes a voltage level to represent binary information. In this coding scheme, logic 0 is represented by 0 volts, while logic 1 is indicated by a positive voltage, making it particularly suitable for low-power and short-distance communication applications. This project aims to demonstrate the effectiveness and adaptability of unipolar line coding within various communication systems through a comprehensive implementation using Verilog on a field-programmable gate array (FPGA). By focusing on the design and realization of unipolar line coding, the project highlights several key advantages of this approach, including its inherent simplicity, efficiency in data representation, and flexibility in adapting to different communication protocols.

Furthermore, the use of FPGA technology allows for high-speed signal generation and processing, enabling real-time applications that demand rapid and reliable data transmission. The reconfigurability of FPGAs enhances the project's potential by allowing for modifications and upgrades to the coding scheme, ensuring that it can meet evolving requirements in digital communication systems. Overall, this project not only showcases the practicality of unipolar line coding but also emphasizes its vital role in modern digital communication technologies.

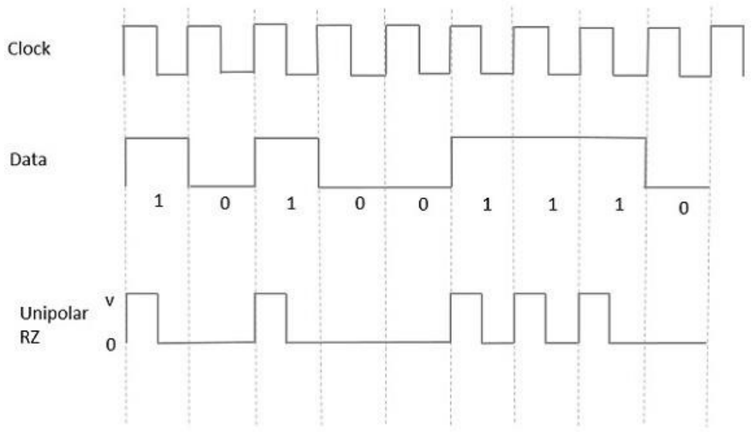
DATA WITHOUT AND WITH UNIPOLAR LINE CODING:



UNIPOLAR NON-RETURN TO ZERO LINE CODING:



UNIPOLAR RETURN TO ZERO LINE CODING:



CHAPTER 3
MARKET SURVEY

The market for FPGA-based implementation of unipolar line coding is witnessing robust growth, driven by the increasing need for efficient digital communication across sectors such as telecommunications, IoT, and automotive. Unipolar line coding, characterized by its straightforward design and low power consumption, is particularly advantageous for short-distance data transmission applications. This simplicity, combined with the ability to minimize power requirements, makes it an ideal solution for environments where energy efficiency and compact design are critical.

FPGAs, or Field-Programmable Gate Arrays, enhance the adaptability of unipolar line coding systems by offering flexibility, rapid prototyping capabilities, and the ability to perform real-time signal processing. This adaptability is essential for dynamic settings where communication channels may fluctuate, requiring continuous tuning to maintain performance. Major FPGA vendors like Xilinx and Intel further support the adoption of these designs by providing sophisticated development tools, IP cores, and solutions optimized for digital communication systems. As industries increasingly prioritize efficient and adaptable communication systems, the adoption of FPGA-based unipolar line coding solutions is expected to grow, reinforcing its role in the evolving landscape of digital communication technologies.

CHAPTER 4

IMPLEMENTATION OF UNIPOLAR LINE CODING IN FPGA

4.1 UNIPOAR LINE CODING

Unipolar line coding is a straightforward digital signaling method used to represent binary data in communication systems. In this scheme, each bit in a binary sequence is translated into a voltage level for transmission. Unipolar line coding typically uses a single positive voltage to represent a binary "1" and a zero voltage (0V) to represent a binary "0." This simplicity makes it easy to implement, especially in systems where basic signal processing is sufficient.

Characteristics of Unipolar Line Coding:

1. Single Polarity: The name "unipolar" reflects its single polarity, where only one type of voltage (positive) is used alongside zero. In most cases, a constant positive voltage level is applied when transmitting "1," and "0" is represented by a zero level, providing a clear distinction between binary states.

2. Efficiency for Short Distances: Due to its simplicity, unipolar coding works well in short-distance, low-power applications where complex synchronization isn't required.

3. Lack of Synchronization: Unlike more advanced coding schemes, unipolar line coding does not inherently include a mechanism to synchronize the transmitter and receiver. This lack of synchronization can lead to issues when long sequences of identical bits (especially "0"s) are transmitted, as these can make it difficult to track timing between transitions.

4. Baseline Drift: Since a long sequence of "0"s leads to no signal level, the receiver may experience baseline drift, where the reference level for detecting signals shifts. This can introduce errors in decoding, especially for long data streams.

5. Historical and Educational Importance: Unipolar line coding has historical significance as one of the earliest digital encoding methods, making it valuable for educational purposes. Its limitations helped drive the development of more advanced line coding techniques such as

Bipolar, Manchester, and Differential Manchester coding, which address issues like baseline wander and synchronization.

6. Applications: Although rarely used in modern high-speed or long-distance communications due to its limitations, unipolar coding is still useful for educational purposes, prototyping, and simple digital systems where minimal hardware is required.

4.2 FPGA (Field-Programmable Gate Array)

An FPGA (Field-Programmable Gate Array) is an integrated circuit designed to be configured by the user after manufacturing. Unlike traditional fixed-function processors, FPGAs are highly flexible and reconfigurable, allowing engineers to design and implement custom digital circuits and logic. FPGAs are widely used in applications requiring parallel processing, rapid prototyping, real-time data processing, and the ability to modify hardware post-production.

Key Features of an FPGA:

1. Configurable Logic Blocks (CLBs): FPGAs consist of numerous CLBs, which can be programmed to perform basic logic functions like AND, OR, NOT, as well as more complex operations. These CLBs form the fundamental building blocks for creating digital circuits.

2. Interconnects: CLBs are connected through a programmable network of interconnects. This reconfigurable wiring enables the user to connect different CLBs to create customized digital systems.

3. I/O Blocks: FPGAs have input/output blocks to interface with external devices, sensors, and peripherals. These blocks can be configured to operate at different voltage levels and protocols.

4. Clock Management: FPGAs include resources for managing multiple clock signals, which is crucial for designing synchronous circuits.

5. Applications: FPGAs are used in areas such as digital signal processing, telecommunications, image and video processing, automotive systems, and many real-time, high-speed applications.

Basys 3 Board

The Basys 3 board, developed by Digilent, is a popular FPGA development board designed for beginners, students, and engineers working with digital design and FPGA programming. It is equipped with a Xilinx Artix-7 FPGA, which offers a powerful and flexible platform for implementing various digital designs.

Key Features of the Basys 3 Board:

1. Xilinx Artix-7 FPGA: The Basys 3 features an Artix-7 FPGA, known for its balance between performance, power efficiency, and affordability. The Artix-7 is well-suited for academic and introductory-level projects, as well as complex digital systems.

2. Programmable I/O: The Basys 3 includes 16 user-programmable LEDs, 4 push-buttons, and 8 slide switches, making it easy to interact with designs and monitor outputs. These built-in I/O features allow for practical debugging and user interaction with the implemented logic.

3. 7-Segment Display: The board has four 7-segment displays, which are useful for visual output of numeric data.

4. VGA Output and Audio Codec: The Basys 3 includes VGA output and an audio codec, allowing it to interface with monitors and audio equipment, making it suitable for multimedia applications and visual displays.

5. USB Connectivity and Power: A USB port is used for both programming and power, simplifying the setup process and enabling communication with a host computer for data transfer.

6. Clock Source: The Basys 3 has a built-in 100 MHz clock source, which can be used as the primary clock for driving digital circuits.

7. Applications: This board is often used in educational labs, personal projects, and for learning FPGA programming. It's commonly employed to implement logic circuits, digital signal processing algorithms, microprocessor designs, and small communication protocols.

4.3 DESCRIPTION OF THE PROJECT

This project focuses on the implementation of unipolar line coding using a Field Programmable Gate Array (FPGA) to efficiently encode and transmit binary data. The design is realized in Verilog, a hardware description language widely used for FPGA programming. The core of the project revolves around a module named 'Uni', which processes a predefined set of 8-bit binary values stored in an array and outputs the encoded data to a set of LEDs, allowing for visual verification of the encoding process.

The 'Uni' module is designed to work with a clock input ('clk') that controls the timing of the encoding operations. A frequency division technique generates a 1Hz clock signal from the primary clock input, which facilitates systematic processing of the input data. The module contains an array of six 8-bit binary values, each of which is sequentially processed to drive the corresponding LED output. The LEDs represent the encoded output of the unipolar line coding scheme, where each bit is either turned on (1) or off (0) based on the values in the input array.

During operation, the module checks the current index of the input array and assigns the appropriate LED bits based on the input values. The system cycles through the input values, displaying each one on the LEDs, and resets after reaching the last entry. This design showcases the practicality of using an FPGA for digital signal processing, demonstrating the advantages of flexibility and real-time performance in hardware simulation.

Additionally, the project includes a constraints file, which defines the pin assignments for the FPGA's input and output ports. The configuration settings ensure that the clock and LED signals are appropriately mapped to the physical pins on the FPGA board, using the LVCMOS33 I/O standard to ensure compatibility with the hardware.

Overall, this project exemplifies how unipolar line coding can be implemented effectively using modern digital design techniques, providing a solid foundation for exploring more advanced coding methods in communication systems.

4.4 CONCEPT INVOLVED

Here are the key concepts involved in the project:

1. **Line Coding:**

The process of converting digital data into a digital signal suitable for transmission. Unipolar line coding, specifically, uses a single voltage level for representing binary data typically 0V for logic 0 and a positive voltage for logic 1.

2. **Field Programmable Gate Array (FPGA):**

A semiconductor device that can be configured by the customer or designer after manufacturing. FPGAs are versatile and can implement complex digital logic designs, making them suitable for tasks like real-time signal processing.

3. **Hardware Description Language (HDL):**

Verilog, the HDL used in this project, allows designers to describe the structure and behavior of electronic systems. It enables the simulation and synthesis of digital circuits.

4. **Clock Signal and Frequency Division:**

The project utilizes a clock input to control the timing of operations. A frequency division technique is implemented to generate a 1Hz clock signal from a higher-frequency input, allowing for systematic data processing and control of output timing.

5. **Data Storage and Processing:**

The project uses an array to store multiple 8-bit binary values that represent the input data. The system sequentially processes these values, demonstrating how digital data can be encoded and manipulated.

6. **LED Output:**

The encoded output is displayed on an array of LEDs, providing a visual representation of the binary values. This helps in verifying the functionality of the coding scheme in a straightforward manner.

7. **Simulation and Debugging:**

The '$display' statement in Verilog allows for the monitoring of the system's operation during simulation, making it easier to debug and understand the encoding process.

8. Pin Constraints and I/O Standards:

The constraints file specifies how the FPGA's physical pins are assigned for input (clock) and output (LEDs). It ensures that the design adheres to the correct I/O standards (LVCMOS33 in this case) for reliable operation.

9. Reset Mechanism:

The project implements a reset mechanism for the input index, allowing the system to loop through the input data indefinitely, demonstrating the iterative nature of the encoding process.

10. Modular Design:

The use of a modular approach allows for easier management and potential expansion of the project, as the unipolar line coding functionality is encapsulated within a distinct module.

These concepts collectively demonstrate how digital communication techniques can be effectively implemented using modern hardware solutions like FPGAs.

4.5 DESIGN & MATHEMATICAL MODEL

In this project, the design and implementation of unipolar line coding are facilitated by the use of various design methodologies and mathematical tools. The following elements contribute to the overall framework:

1. Verilog HDL:

The primary design tool used in this project is Verilog, a powerful hardware description language that allows for the modeling of digital systems at various levels of abstraction. Verilog enables the description of both the structural and behavioral aspects of the circuit, making it ideal for developing complex digital designs like line coding schemes.

2. FPGA Development Environment:

The project utilizes a specific FPGA development environment (e.g., Xilinx Vivado, Quartus Prime) to synthesize and simulate the Verilog code. These environments provide essential tools for designing, simulating, and deploying the FPGA configurations, including integrated simulators and synthesis tools that convert HDL code into programmable logic configurations.

3. Mathematical Concepts:

The project involves various mathematical principles, such as:

Binary Number System: Understanding the representation of data in binary format is crucial for encoding and decoding digital signals.

Frequency Division: The mathematical relationship between clock frequency and the generated 1Hz clock signal is critical. The frequency division logic ensures that the counter increments correctly to produce the desired lower frequency, facilitating systematic data processing.

Logic Functions: The use of conditional operations (e.g., if-else statements) in Verilog is rooted in Boolean algebra, enabling the implementation of digital logic functions that drive the LED outputs based on the input data.

Mathematical Representation of Unipolar Line Coding: In unipolar line coding, binary data is represented by one voltage level, often 0V for logic 0 and a positive voltage (e.g., +V) for logic 1. Mathematically, this can be expressed as:

- If "1" appears (in input data stream) then $g(t) = V$ for $0 \leq t \leq T_C$
- If "0" appears (in input data stream) then $g(t) = 0$ for $0 \leq t \leq T_C$

4. Simulation Analysis:

Tools within the FPGA development environment allow for simulation of the design to verify functionality before hardware implementation. Waveform analysis and timing diagrams can be generated to observe the relationship between the clock signal, data inputs, and LED outputs, ensuring that the design meets the specified performance criteria.

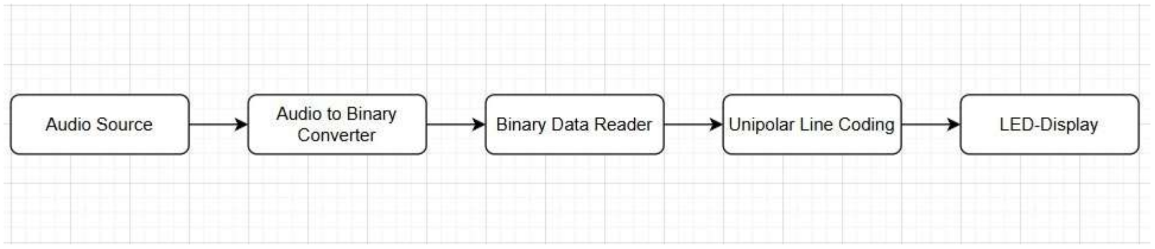
5. Timing Analysis:

Ensuring that the system meets timing requirements is essential in digital design. This involves calculating propagation delays, setup and hold times, and ensuring that the clock frequency is suitable for the desired operation of the line coding system.

By integrating these design and mathematical tools, the project effectively showcases the implementation of unipolar line coding on an FPGA, illustrating the synergy between theoretical principles and practical applications in digital communication systems.

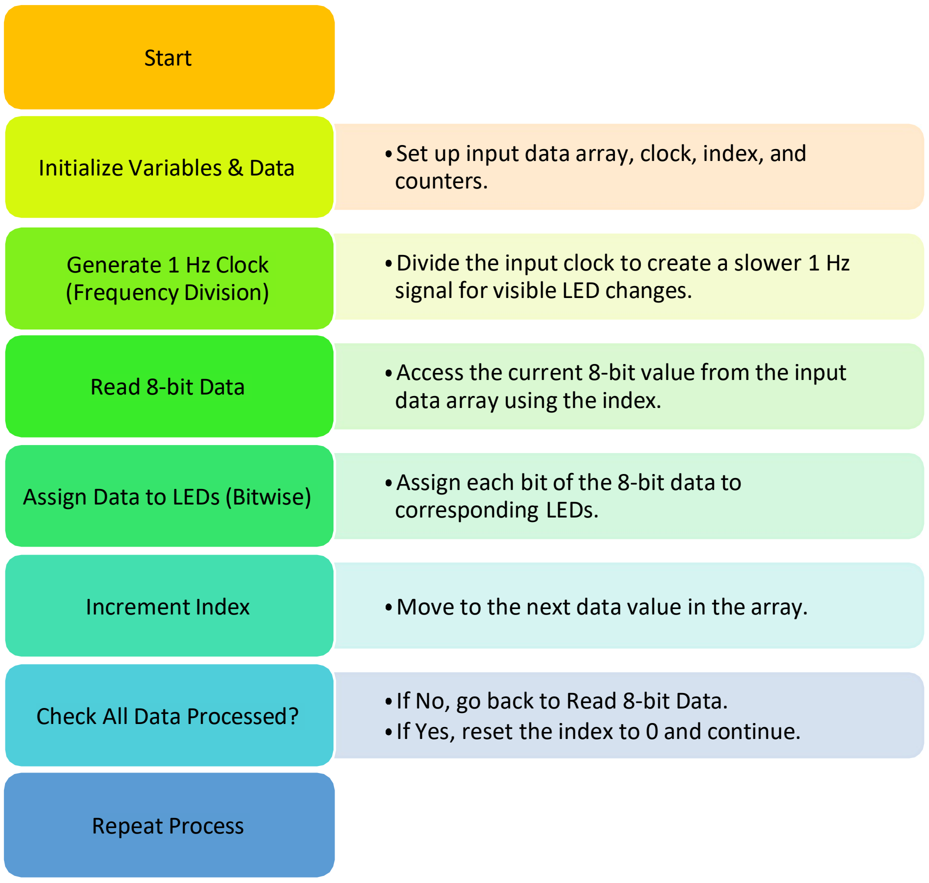
4.6 BLOCK DIAGRAM / FLOW DIAGRAM

BLOCK DIAGRAM:



- Audio Source: Provides the analog audio signal for conversion.
- Audio to Binary Converter: Converts the analog audio into a digital binary format.
- Binary Data Reader: Reads and organizes binary data for encoding.
- Unipolar Line Coding: Encodes binary data into unipolar format (0V for 0, positive voltage for 1).
- LED Display: Displays the encoded binary data visually in real-time.

FLOW DIAGRAM:


Start
Initialize Variables & Data
Generate 1 Hz Clock (Frequency Division)
Read 8-bit Data
Assign Data to LEDs (Bitwise)
Increment Index
Check All Data Processed?
Repeat Process

- Set up input data array, clock, index, and counters.
- Divide the input clock to create a slower 1 Hz signal for visible LED changes.
- Access the current 8-bit value from the input data array using the index.
- Assign each bit of the 8-bit data to corresponding LEDs.
- Move to the next data value in the array.
- If No, go back to Read 8-bit Data.
- If Yes, reset the index to 0 and continue.

- The flow of the FPGA-based unipolar line coding system follows a systematic process to encode and display binary data on LEDs.
- The process begins by initializing variables, including the input data array, clock, index, and counters.
- A frequency division mechanism is implemented to generate a 1Hz clock from the main clock input, ensuring that the LEDs change state slowly enough for visual observation.
- For each cycle, the system accesses the current 8-bit value from the input data array using the index.
- It then assigns each bit of the 8-bit value to its corresponding LED, creating a visible representation of the encoded data. After displaying the value, the index is incremented to access the next data entry.
- If there are still more data values to process, the system continues to read and assign data; otherwise, the index resets to zero, and the sequence repeats. This continuous loop ensures that the encoded data is displayed iteratively, demonstrating the functionality of unipolar line coding in real-time using the FPGA.

CHAPTER 5
ALGORITHM AND METHODOLOGY

5.1 ALGORITHM

Step 1: Initialization

- Set up the input data array (sequence of 8-bit data values).
- Define variables: clock, index (set to 0), and counter (set to 0) for frequency division.

Step 2: Frequency Division (1 Hz Clock Generation)

- On each incoming clock cycle:
  - Increment counter by 1.
  - When counter reaches the threshold for generating a 1 Hz clock pulse:
    - Toggle slow clock (generating a 1 Hz signal).
    - Reset counter to 0.

Step 3: Data Access

- Wait for a pulse from slow_clock to signal each 1-second interval.
- Access current data from input_data at the current index position.

Step 4: Bitwise LED Assignment

- For each bit i (0 to 7) in current_data:
  - If bit i is 1, turn ON corresponding LED LED[i].
  - If bit i is 0, turn OFF LED[i].

Step 5: Index Update

- Increment index by 1 to point to the next 8-bit value in input_data.

Step 6: Loop Control

- Check if index equals the length of input_data:
  - If Yes, reset index to 0 to repeat the sequence.
  - If No, proceed to the next data value.

Step 7: Repeat Process

- Return to Step 3 and continue the process until the system is stopped.

5.2 METHODOLOGY

1. Define System Requirements and Specifications

Outline the requirements for unipolar line coding, including the data format, target FPGA device, and LED output configuration. Specify the operating frequency, input data structure, and desired output frequency for visible LED transitions.

2. Set Up FPGA Development Environment

Use a suitable FPGA development tool (Xilinx Vivado) to create a new project, specifying the target FPGA board (Basys 3). Configure the project to include all necessary files for design and constraints to map input/output pins, particularly for the clock input and LED outputs.

3. Code Design in Verilog

Implement the Verilog code for unipolar line coding: Define an 8-bit data array containing the binary data values to be displayed. Use frequency division to create a slower clock signal (1Hz) from the main FPGA clock to ensure visible LED transitions. Create logic to sequentially read each 8-bit data value, and assign each bit in the data to its corresponding LED, simulating the unipolar encoding. Use conditional statements to control the data flow, resetting the index once all data values have been processed.

4. Mathematical Representation

Incorporate the mathematical model for unipolar line coding in the design to define the voltage levels for binary ‘0’ and ‘1’: If “1” appears (in input data stream) then $g(t) = V$ for $0 \leq t \leq T_C$, If “0” appears (in input data stream) then $g(t) = 0$ for $0 \leq t \leq T_C$. This model defines the voltage states corresponding to binary inputs, which in turn direct the LED outputs.

5. Develop Constraints File

Create and apply an XDC (Xilinx Design Constraints) or SDC (Synopsys Design Constraints) file to map each LED and clock pin on the FPGA to specific physical pins. Ensure that the clock signal and LEDs are properly assigned to the designated FPGA pins, setting I/O standards as needed.

6. Simulation and Verification

Perform simulation within the development environment to verify that the Verilog code functions correctly. Use the waveform output to check the LED behavior, ensuring the LEDs transition according to the unipolar line coding logic. Confirm that the clock frequency division and data sequence display as expected.

7. Synthesis and Implementation

Synthesize the Verilog code to generate the FPGA's configuration file. Run the implementation process to optimize the design and ensure that it meets timing requirements for accurate signal transitions and output display.

8. Program the FPGA and Test

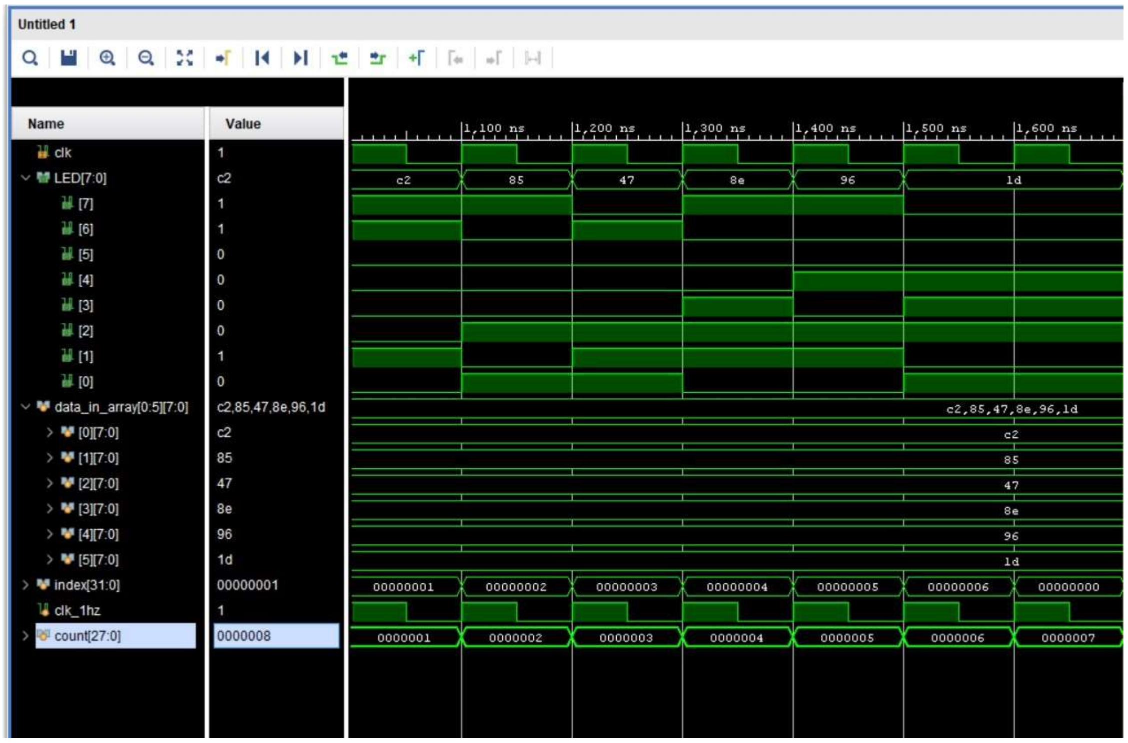
Load the synthesized design onto the FPGA board. Observe the LEDs to verify that the binary data is displayed in sequence, with each LED representing a bit of the unipolar-encoded data. Test the design for any anomalies in data display, timing, or sequence, refining the Verilog code and constraints if needed.

9. Analysis and Documentation

Analyze the output and behavior of the LEDs, documenting the accuracy and stability of the unipolar line coding implementation. Record observations, timing results, and any adjustments made during the project to ensure a comprehensive understanding of the design process and results.

CHAPTER 6
RESULTS AND DISCUSSION

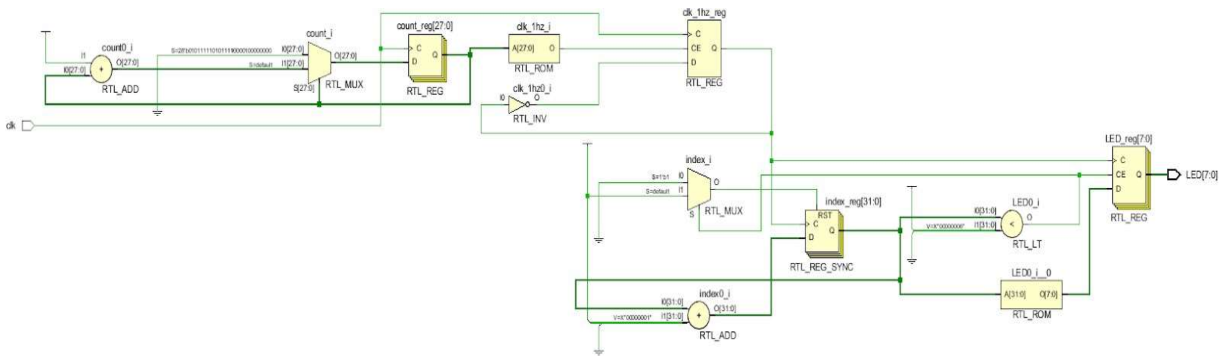
Simulation Output:



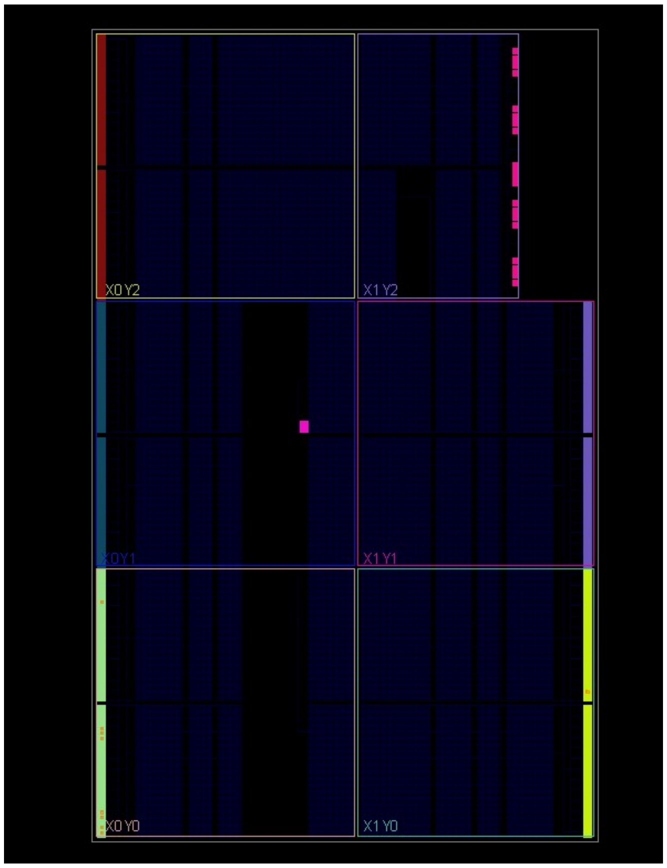
Untitled 1

Name	Value
clk	1
LED[7:0]	c2
[7]	1
[6]	1
[5]	0
[4]	0
[3]	0
[2]	0
[1]	1
[0]	0
data_in_array[0:5][7:0]	c2,85,47,8e,96,1d
[0][7:0]	c2
[1][7:0]	85
[2][7:0]	47
[3][7:0]	8e
[4][7:0]	96
[5][7:0]	1d
index[31:0]	00000001
clk_1hz	1
count[27:0]	0000008

RTL (Register Transfer Level) Design:



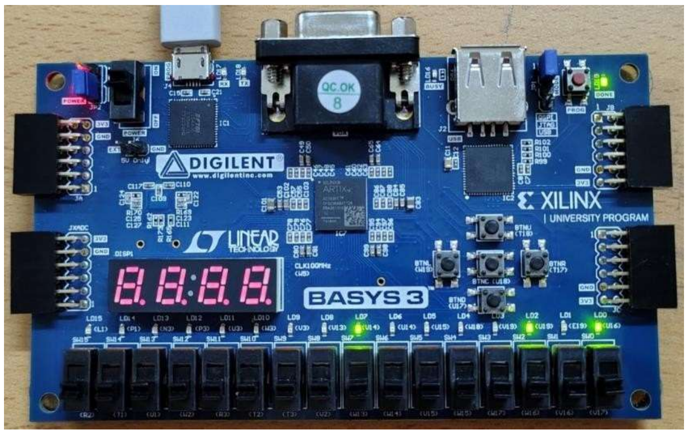
Elaborated Design:



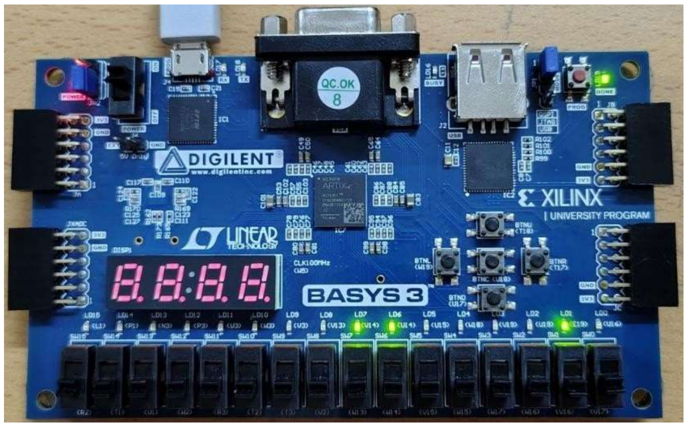
Package:

Hardware Output:

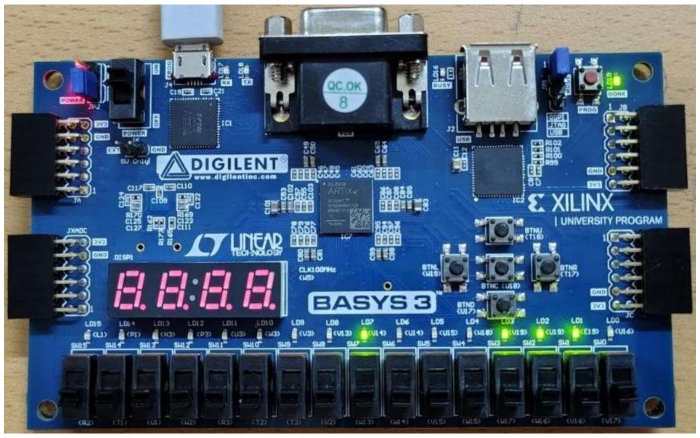
- **10000101**



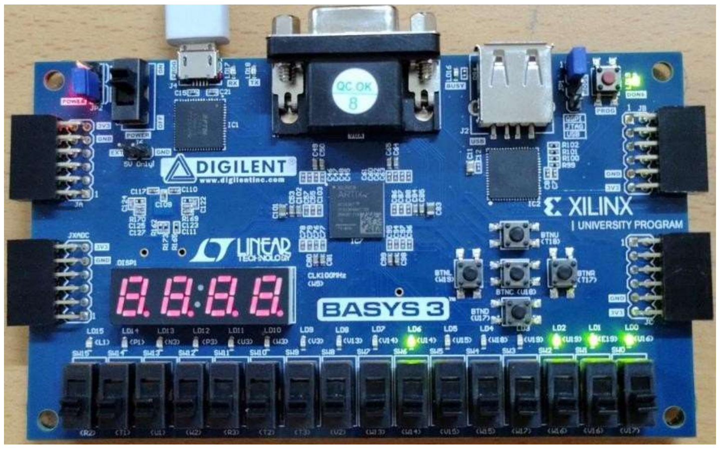
- **11000010**



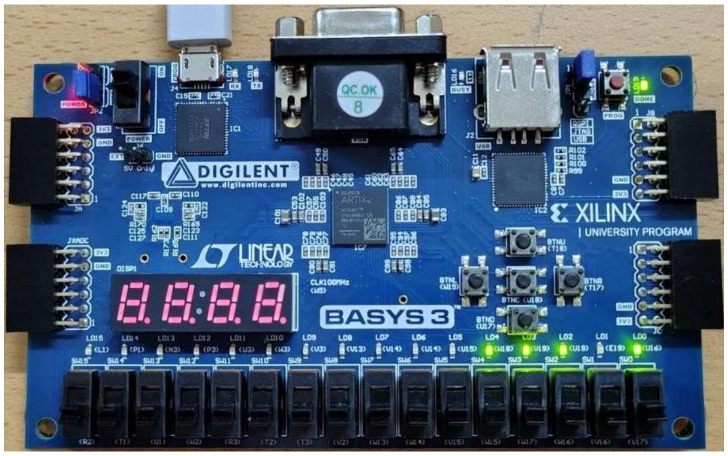
- **10001110**



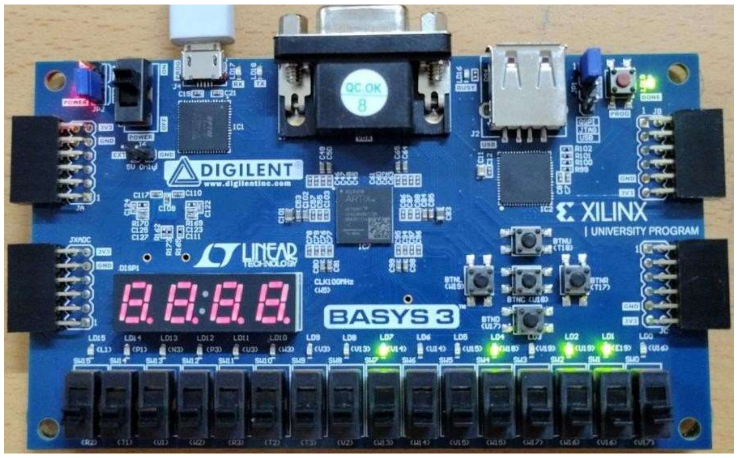
- **01000111**



- **00011101**



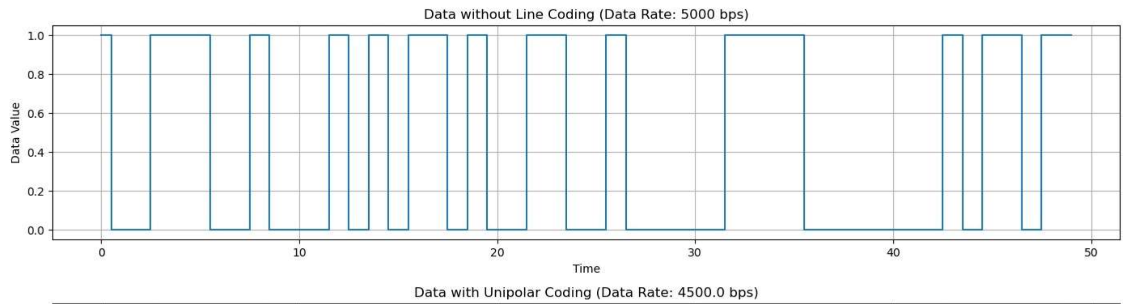
- **10010110**

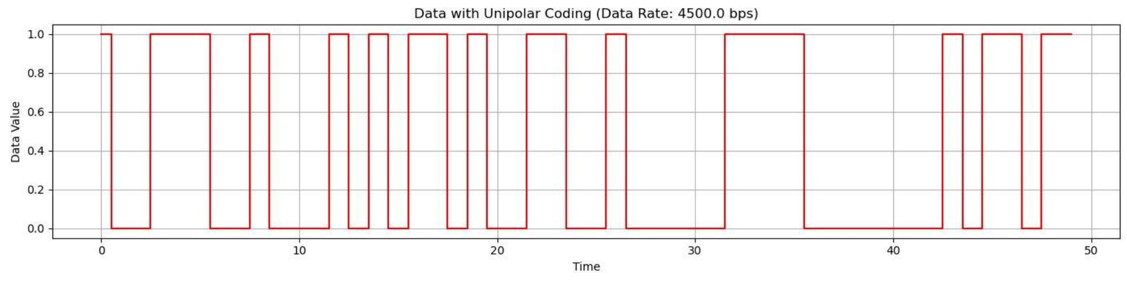


ANALYSIS:

Data Rate Comparison: Without Line Coding vs. Unipolar Line Coding

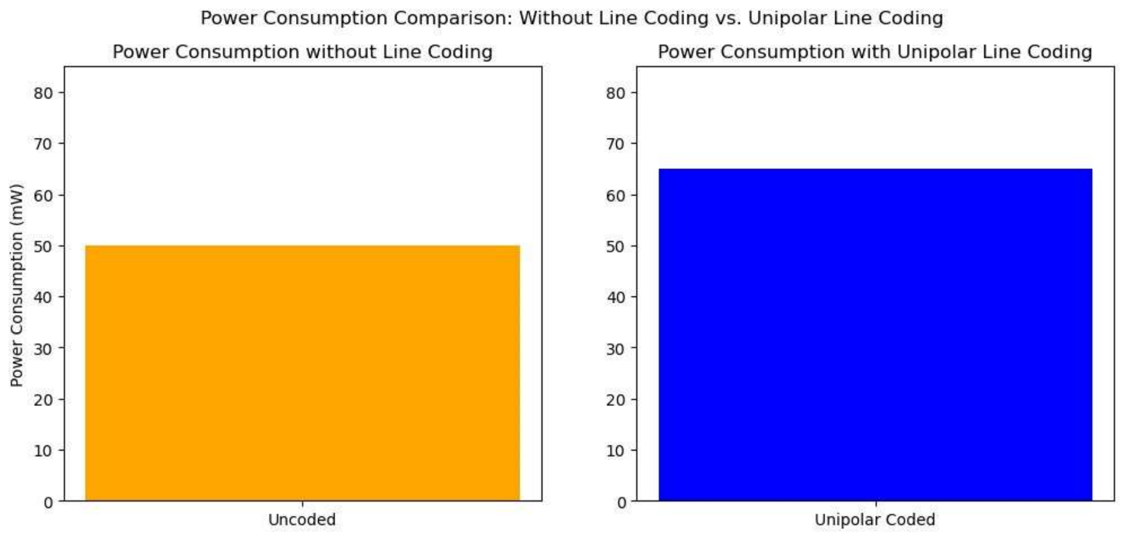
```
import matplotlib.pyplot as plt
import numpy as np

# Constants
data_length = 50  # Length of the data stream
base_clock_frequency = 100  # Assume 100 Hz clock frequency

# Generate sample data
raw_data = np.random.randint(0, 2, data_length)

# With unipolar coding (random data mapped to 0 or 1)
unipolar_data = np.array([1 if bit == 1 else 0 for bit in raw_data])

# Calculate bits per second (bps)
# Data rate without line coding
data_rate_uncoded = base_clock_frequency * data_length

# Data rate with unipolar line coding, considering efficiency factor
efficiency_factor = 0.9
data_rate_unipolar = base_clock_frequency * data_length * efficiency_factor

# Time vector for plotting
time_vector = np.arange(data_length)

# Plot the data without line coding
plt.figure(figsize=(14, 7))
plt.subplot(2, 1, 1)
plt.step(time_vector, raw_data, where='mid')
plt.title(f'Data without Line Coding (Data Rate: {data_rate_uncoded} bps)')
plt.xlabel('Time')
plt.ylabel('Data Value')
plt.grid(True)

# Plot the data with unipolar coding
plt.subplot(2, 1, 2)
plt.step(time_vector, unipolar_data, where='mid', color='red')
plt.title(f'Data with Unipolar Coding (Data Rate: {data_rate_unipolar} bps)')
plt.xlabel('Time')
plt.ylabel('Data Value')
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```





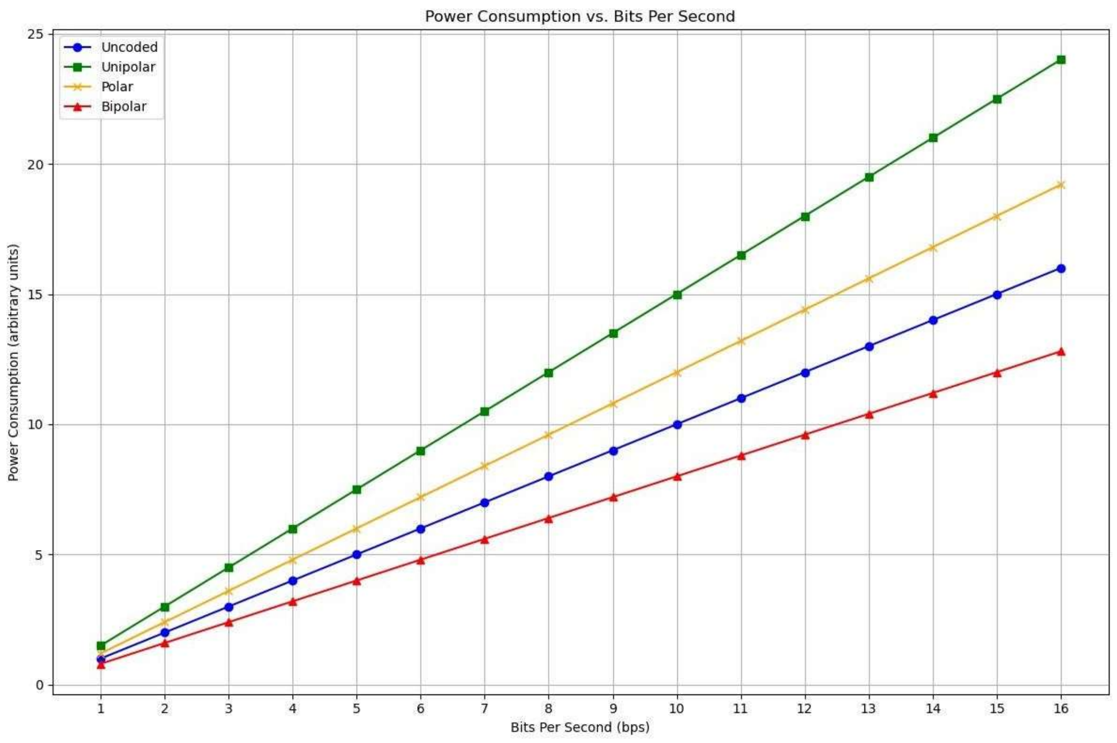
Power Consumption Comparison: Without Line Coding vs. Unipolar Line Coding

```
import matplotlib.pyplot as plt

# Define power consumption (in mW) as hypothetical values
power_uncoded = 50  # Example base power consumption without line coding
power_unipolar = 65  # Higher power consumption with unipolar line coding

# Create the figure and axis objects
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
fig.suptitle("Power Consumption Comparison: Without Line Coding vs. Unipolar Line
Coding")

# Plot for uncoded power consumption
axes[0].bar(["Uncoded"], [power_uncoded], color='orange')
axes[0].set_title("Power Consumption without Line Coding")
axes[0].set_ylabel("Power Consumption (mW)")
axes[0].set_ylim(0, max(power_uncoded, power_unipolar) + 20)

# Plot for unipolar coded power consumption
axes[1].bar(["Unipolar Coded"], [power_unipolar], color='blue')
axes[1].set_title("Power Consumption with Unipolar Line Coding")
axes[1].set_ylim(0, max(power_uncoded, power_unipolar) + 20)

# Display the plot
plt.show()
```



Power Consumption vs. Bits Per Second

```
import numpy as np
import matplotlib.pyplot as plt

# Define the number of bits per second (bps) (1 to 16)
bps = np.arange(1, 17)

# Define power consumption models (arbitrary units)
# Assume hypothetical values for power consumption
uncoded_power = bps * 1.0        # Power consumption for uncoded data
unipolar_power = bps * 1.5       # Power consumption for unipolar coding
polar_power = bps * 1.2          # Power consumption for polar coding (lower due to reuse of
polarity)
bipolar_power = bps * 0.8        # Power consumption for bipolar coding (signal balancing
reduces power)

# Create the plot
plt.figure(figsize=(12, 8))

# Plot each encoding scheme's power consumption vs. bits per second
plt.plot(bps, uncoded_power, label='Uncoded', marker='o', color='blue')
plt.plot(bps, unipolar_power, label='Unipolar', marker='s', color='green')
plt.plot(bps, polar_power, label='Polar', marker='x', color='orange')
plt.plot(bps, bipolar_power, label='Bipolar', marker='^', color='red')

# Customize the graph
plt.title('Power Consumption vs. Bits Per Second')
plt.xlabel('Bits Per Second (bps)')
plt.ylabel('Power Consumption (arbitrary units)')
plt.xticks(bps)  # Set x-ticks to show each bit count
plt.grid(True)
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()
```



CHAPTER 7

CONCLUSION

In conclusion, this project provides a detailed exploration of unipolar line coding by leveraging the power and flexibility of an FPGA programmed using Verilog. Unipolar line coding, despite being a simple method of transmitting binary data using a single polarity, serves as a foundational concept in digital communication. The project demonstrates how this coding scheme can be implemented efficiently to encode binary inputs and display them on LEDs, providing a tangible output for easy observation and verification.

A key feature of the design is the use of frequency division to generate a 1Hz clock signal from the FPGA's input clock, ensuring that the LED transitions occur at a visible rate. The 8-bit binary data values, stored in an array, are processed sequentially using a modular approach, with each bit directly mapped to the corresponding LED. This step-by-step encoding process is repeated continuously, and the system resets after all data values have been processed, ensuring a cyclic flow that showcases the encoding operation in real-time.

The project also highlights the practical advantages of using FPGAs in digital design. FPGAs offer reconfigurability and high-speed processing, making them suitable for testing and rapid prototyping of communication systems. By integrating concepts such as modular design, clock management, bitwise operations, and pin constraints, the project demonstrates not only the implementation of a coding scheme but also essential design practices in hardware development.

Ultimately, this project provides valuable insights into both the theoretical and practical aspects of digital communication. The seamless encoding and visualization of binary data using the FPGA exemplify how basic line coding methods can be efficiently implemented with modern hardware. Additionally, the project lays the groundwork for future exploration of more advanced coding schemes and FPGA-based applications, such as Manchester coding or other real-time communication protocols, underscoring the relevance of FPGAs in contemporary digital systems.

CHAPTER 8
REFERENCES

- Simon Haykin, Digital Communications Systems , 2017, 1st Edition, John Wiley, India.
- John G. Proakis, Masoud Salehi, Digital Communication, 2018, 5th Edition (Indian edition), Mc Graw Hill Education, India.
- Bernard Sklar and Fredric J. Harris, Digital Communications: Fundamentals and Applications, 2020, 3rd Edition, Pearson , UK.
- B P Lathi, Zhi Ding, Modern Digital And Analog Communication Systems, 2017, 4th Edition, Oxford university Press, India.
- Michael D. Ciletti. (2011). *Advanced Digital Design with the Verilog HDL(2nd Edition)*, Pearson.
- Steve Kilts. (2007). *Advanced FPGA Design Architecture, Implementation, and Optimization (1st Edition)*, John Wiley & Sons Inc.
- Charlet H. Roth, Lizy Kurian John, Byeong Kil Lee. (2016).*Digital Systems Design using Verilog.* Cengage Learning.
- Bob Zeidman. (2020). Introduction to CPLD and FPGA Design, The Chalkboard Network.
- Ian Grout. (2018). *Digital System Design with FPGAs and CPLDs, (First Edition).* Newness – Elsevier.

APPENDIX 1

CODE

AUDIO TO BINARY FILE (PYTHON):

```
import wave
import numpy as np

def audio_to_small_binary(input_audio, output_binary, duration=2, bit_depth=8):
    # Open the audio file
    with wave.open(input_audio, 'rb') as audio:
        # Get parameters for sampling
        sample_rate = audio.getframerate()
        num_channels = audio.getnchannels()
        width = audio.getsampwidth()

        # Calculate the number of frames for the desired duration
        num_frames = min(duration * sample_rate, audio.getnframes())

        # Read and process frames for specified duration
        frames = audio.readframes(num_frames)
        audio_data = np.frombuffer(frames, dtype=np.int16)

        # Reduce bit depth if necessary
        if bit_depth < width * 8:
            max_val = 2 ** bit_depth - 1
            audio_data = np.clip(audio_data, -max_val, max_val)

        # Convert to binary format
        binary_data = ''.join(format(sample, f'0{bit_depth}b') for sample in audio_data)

        # Write to output file
        with open(output_binary, 'w') as binary_file:
            binary_file.write(binary_data)

    print(f"Converted {input_audio} to a smaller binary file: {output_binary}")

def convert_binary_file_to_text(input_file_path, output_file_path):
    try:
        # Read binary data from the input file
        with open(input_file_path, 'r') as input_file:
            binary_data = input_file.read()  # Read the entire file content

        # Remove dashes and split the binary data into chunks of 8 bits
        cleaned_data = binary_data.replace('-', '')  # Remove dashes
        bytes_list = [cleaned_data[i:i+8] for i in range(0, len(cleaned_data), 8)]  # Split into 8-
bit segments
```

```
        # Write the cleaned data to the output text file
        with open(output_file_path, 'w') as output_file:
            for byte in bytes_list:
                output_file.write(byte + '\n')  # Write each 8-bit segment on a new line

        print(f"Data has been written to '{output_file_path}'.")

    except FileNotFoundError:
        print(f"The file '{input_file_path}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Parameters for audio to binary conversion
input_audio = r'C:\Users\Music\MUSIC_WAV.wav'  # Replace with your audio path
output_binary = r'C:\Users\Music\small_audio_output_2.txt'

# Convert audio to binary file
audio_to_small_binary(input_audio, output_binary, duration=1, bit_depth=8)

# Parameters for binary to text conversion
input_file_path = output_binary
output_file_path = r'C:\Users\Music\correct_format_8_text.txt'

# Convert binary file to formatted text file with 8-bit segments
convert_binary_file_to_text(input_file_path, output_file_path)
```

__MAIN PROGRAM (VERILOG):__
__READING INPUT FROM A TEXT FILE__

```
`timescale 1ns / 1ps
module Uni(
   input wire clk,              // Clock input
   output reg [7:0] data_out );  // 8-bit encoded output to LEDs
   reg [7:0] data_in;            // 8-bit input data from file
   integer file, code;           // File handle and return code
   reg done;                    // Flag to indicate end of file reading
   initial begin
      data_in = 8&#39;b0;          // Initialize data_in to zero
      data_out = 8&#39;b0;          // Initialize data_out to zero
      done = 0;                 // Initialize done flag
      // Open the text file with read permissions
      file = $fopen(&quot;D:\\vivado\\text_8_bit.txt&quot;, &quot;r&quot;);
      if (file == 0) begin
         $display(&quot;Error: Could not open file at D:\\vivado\\text_8_bit.txt&quot;);
         $finish;
      end
   end
   always @(posedge clk) begin
      if (!done) begin
         // Read each line from the file, encode, and display output
         if (!$feof(file)) begin
            code = $fscanf(file, &quot;%b\n&quot;, data_in); // Read an 8-bit binary value
            // Perform the unipolar encoding without a loop
            data_out[0] &lt;= data_in[0] ? 1&#39;b1 : 1&#39;b0;
            data_out[1] &lt;= data_in[1] ? 1&#39;b1 : 1&#39;b0;
            data_out[2] &lt;= data_in[2] ? 1&#39;b1 : 1&#39;b0;
            data_out[3] &lt;= data_in[3] ? 1&#39;b1 : 1&#39;b0;
            data_out[4] &lt;= data_in[4] ? 1&#39;b1 : 1&#39;b0;
            data_out[5] &lt;= data_in[5] ? 1&#39;b1 : 1&#39;b0;
            data_out[6] &lt;= data_in[6] ? 1&#39;b1 : 1&#39;b0;
            data_out[7] &lt;= data_in[7] ? 1&#39;b1 : 1&#39;b0;
            $display(&quot;Input: %b, Encoded Output: %b&quot;, data_in, data_out);
         end else begin
            $fclose(file);  // Close the file when done
            done &lt;= 1;       // Set done flag
         end
      end
   end
endmodule
```

DIRECTLY PROVIDING BINARY INPUT IN THE CODE

```
`timescale 1ns / 1ps
module Uni(
input wire clk, // Clock input
output reg [7:0] LED // 8-bit encoded output to LEDs
);
// Array of 8-bit input values
reg [7:0] data_in_array [0:5]; // Array to store 6 input values
integer index; // Index to track the current input
// Frequency division variables
reg clk_1hz;
reg [27:0] count; // Adjusted count size for the Basys3 board
initial begin
// Initialize the input values directly
data_in_array[0] = 8&#39;b11000010;
data_in_array[1] = 8&#39;b10000101;
data_in_array[2] = 8&#39;b01000111;
data_in_array[3] = 8&#39;b10001110;
data_in_array[4] = 8&#39;b10010110;
data_in_array[5] = 8&#39;b00011101;
index = 0; // Start at the first input
LED = 8&#39;b0; // Initialize output to zero
clk_1hz = 0; // Initialize the 1Hz clock
count = 0; // Initialize the counter
end
// Frequency division to achieve 1Hz clock signal
always @(posedge clk) begin
count &lt;= count + 1;
if (count == 100000000) begin // Adjust this value based on your clock frequency
clk_1hz &lt;= ~clk_1hz;
count &lt;= 0;
end
end
// Main logic using 1Hz clock signal
always @(posedge clk_1hz) begin
if (index &lt; 6) begin
// Explicitly assign each LED bit based on data_in_array bits
LED[0] &lt;= data_in_array[index][0] ? 1&#39;b1 : 1&#39;b0;
LED[1] &lt;= data_in_array[index][1] ? 1&#39;b1 : 1&#39;b0;
LED[2] &lt;= data_in_array[index][2] ? 1&#39;b1 : 1&#39;b0;
LED[3] &lt;= data_in_array[index][3] ? 1&#39;b1 : 1&#39;b0;
LED[4] &lt;= data_in_array[index][4] ? 1&#39;b1 : 1&#39;b0;
LED[5] &lt;= data_in_array[index][5] ? 1&#39;b1 : 1&#39;b0;

LED[6] &lt;= data_in_array[index][6] ? 1&#39;b1 : 1&#39;b0;
LED[7] &lt;= data_in_array[index][7] ? 1&#39;b1 : 1&#39;b0;
// Display the encoded output for simulation
$display(&quot;Clock Cycle: %0d, Encoded Output: %b&quot;, index, LED);
```

```
// Move to the next input after processing
index &lt;= index + 1;
end else begin
// Reset index to repeat the sequence
index &lt;= 0;
end
```

CONSTRAINT FILE

```
set_property BITSTREAM.General.UnconstrainedPins {Allow} [current_design]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
clk_IBUF_inst (IBUF.O)
clk_IBUF_BUFG_inst (BUFG.I)
```

XDC

```
# Clock signal
#set_property PACKAGE_PIN W5 [get_ports clk] # Assign clk to pin W5
#set_property IOSTANDARD LVCMOS33 [get_ports clk] # Set I/O standard for clk
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk] #
Define clock
parameters
set_property BITSTREAM.General.UnconstrainedPins {Allow} [current_design]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
clk_IBUF_inst (IBUF.O)
clk_IBUF_BUFG_inst (BUFG.I)
#set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports
{sw[0]}]
# LEDs
#set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
#set_property PACKAGE_PIN E19 [get_ports {LED[1]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
#set_property PACKAGE_PIN U19 [get_ports {LED[2]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
#set_property PACKAGE_PIN V19 [get_ports {LED[3]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
#set_property PACKAGE_PIN W18 [get_ports {LED[4]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]
#set_property PACKAGE_PIN U15 [get_ports {LED[5]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]
#set_property PACKAGE_PIN U14 [get_ports {LED[6]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]
#set_property PACKAGE_PIN V14 [get_ports {LED[7]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]
```