“DESIGN, VERIFICATION, AND IMPLEMENTATION OF SPI PROTOCOL”

Submitted in Partial Fulfillment for the award of

Post Graduate Diploma in VLSI Design (PG-DVLSI)

from

C-DAC, ACTS (Pune)

Guided By:

Mr. Rohit Kankal

Presented by:

Kartikesh Wasankar	**PRN:** 240340133011
Deven Prajapati	**PRN:** 240340133016
Preeti Maddhyeshia	**PRN:** 240340133017
Smriti Karn	**PRN:** 240340133021
Vibhu Gupta	**PRN:** 240340133027
Yash Dhanak	**PRN:** 240340133029

Centre for Development of Advanced Computing (C-DAC), Pune

TABLE OF CONTENTS

ABSTRACT

The Serial Peripheral Interface (SPI) bus is a synchronous serial interface data bus with full duplex, signal lines, simple protocol, and fast transmission speed. The SPI designed by Motorola's are general purpose solutions offering viable ways to controlling SPI-bus, and highly flexible to suit any needs. Based on these characteristics of SPI, parallel high-speed computing with Field programmable gate array devices of Digilent Basys 3 board is used to meet device expansion and experiment in high-rate environments.

This project introduces the structure and working principle of SPI communication bus, analyzes its timing structure, and uses a different approach than the state machine method to realize its SPI bus communication function on FPGA. The module circuit of SPI is written by Verilog hardware description language, and the waveform is simulated in Questa Sim simulator. After the simulation waveform analysis, System Verilog Hardware Description language is used for the verification and the feasibility of the said system method.

The project concludes with a discussion on the results, highlighting the SPI core's effectiveness in real-time data communication and its adaptability to different system configurations. Insights are provided into optimization strategies and potential areas for future improvements. This work contributes to the field of FPGA-based communication systems by offering a well-documented approach to SPI protocol implementation and verification, serving as a reference for engineers and researchers working with embedded FPGA designs.

CHAPTER 1 : INTRODUCTION

1.1. INTRODUCTION

The serial peripheral interface (SPI) is a communication interface used to send data between multiple devices. Devices connected with SPI are in a master-slave relationship. The master is the controlling device (usually a micro-controller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave.

SPI is a synchronous machine having maximum speed up to 10Mbps. Master generates the serial clock on which data communication/transfer operations with slaves takes place. The CS is use to select the slave and it is generally active low, when its value is high it disconnect's the interface connection. MOSI and MISO are the data lines. MOSI transmits data from the master to the slave and MISO transmits data from the slave to the master. The data bus is a single bit data bus line, so the data transfer is done bit by bit.

SPI can have multiple master – slave configuration this mode of operation is known as daisy – chain mode. In daisy-chain mode, the slaves are configured such that the chip select signal for all slaves is tied together and data propagates from one slave to the next. In this configuration, all slave receives the same SPI clock at the same time. The data from the master is directly connected to the first slave and that slave become master for next slave and provides data to the next slave and so on.

The SPI bus transmission has a total of four modes, which are defined by clock polarity (CPOL, Clock Polarity) and clock phase (CPHA, Clock Phase), where the CPOL parameter specifies the level of the SCLK clock signal idle state. CPHA specifies whether data is sampled on the rising edge of the SCLK clock or on the falling edge.

1.2. AIM OF THE PROJECT

This project explores the design and implementation of a Serial Peripheral Interface (SPI) communication protocol, employing a novel approach that sets it apart from traditional designs based on finite state machines (FSMs). SPI, a widely-used synchronous communication protocol, typically relies on FSMs to control the sequence of data transfer between a master and one or more slave devices. However, the conventional FSM approach can be limited by its rigidity, complexity, and the overhead associated with state transitions, especially in systems requiring higher levels of efficiency, flexibility, and scalability.

In this project, we have undertaken the design and implementation of a Serial Peripheral Interface (SPI) protocol system, comprising a master, a slave, and a top-level module, all intricately integrated within a comprehensive test bench and verification environment developed using System Verilog. The SPI protocol, widely recognized for its simplicity and efficiency in facilitating serial communication between devices, is implemented in mode 1,1, characterized by Clock Polarity (CPOL) set to 1 and Clock Phase (CPHA) set to 1. This configuration ensures that data is captured on the clock's rising edge and changes on the falling edge, making it suitable for a range of applications requiring synchronized data transfer. To accommodate the SPI protocol's timing requirements, we have engineered a clock conversion mechanism within our design that reduces the initial 100 MHz clock signal to a more suitable 1 MHz SPI clock.

The architecture leverages several always blocks, strategically implemented to enable parallel processing of the protocol's various functions. This parallelism is key to achieving efficient and reliable communication, as it allows the system to handle multiple operations simultaneously, thereby reducing latency and improving overall performance. The master and slave modules are designed to interact seamlessly, with signals such as clk for the main test bench clock, sclk and i_sclk for the SPI clock signals, data_in (a 9-bit wide input bus), chip select for enabling the slave, MOSI (Master Out Slave In) for transmitting data from the master to the slave, MISO (Master In Slave Out) for receiving data from the slave, and data_out for output data

In summary, this project showcases a detailed and rigorous approach to designing an SPI protocol system, with a strong emphasis on timing accuracy, parallel processing, modularity, and thorough verification. By integrating these elements, the project delivers a reliable and efficient communication interface that not only meets the immediate requirements but is also adaptable for future extensions and applications. This level of detail and precision highlights the robustness of the design, ensuring that it is well-suited for deployment in real-world systems where reliable and efficient data communication is critical.

CHAPTER 2 : KNOWING SPI PROTOCOL

2.1. INTRODUCTION TO SPI

History of SPI

The SPI protocol was first introduced by Motorola in the 1980s as a proprietary interface for their microcontrollers. Over time, the protocol gained popularity and became a de facto standard in the industry. Today, SPI is supported by a wide range of microcontrollers and peripheral devices from various manufacturers.

The Serial Peripheral Interface (SPI) is a synchronous, full-duplex, serial communication protocol used to interface microcontrollers or microprocessors with peripheral devices such as sensors, displays, memory devices, and other ICs.

The SPI communication protocol consists of four wires:

1. **SCLK (Clock)**: The clock signal is generated by the master device and is used to synchronize the data transmission.
2. **MOSI (Master Out Slave In)**: The MOSI wire is used to transmit data from the master device to the slave devices.
3. **MISO (Master In Slave Out)**: The MISO wire is used to transmit data from the slave devices to the master device.
4. **SS (Slave Select)**: The SS wire is used to select the slave device that the master device wants to communicate with.

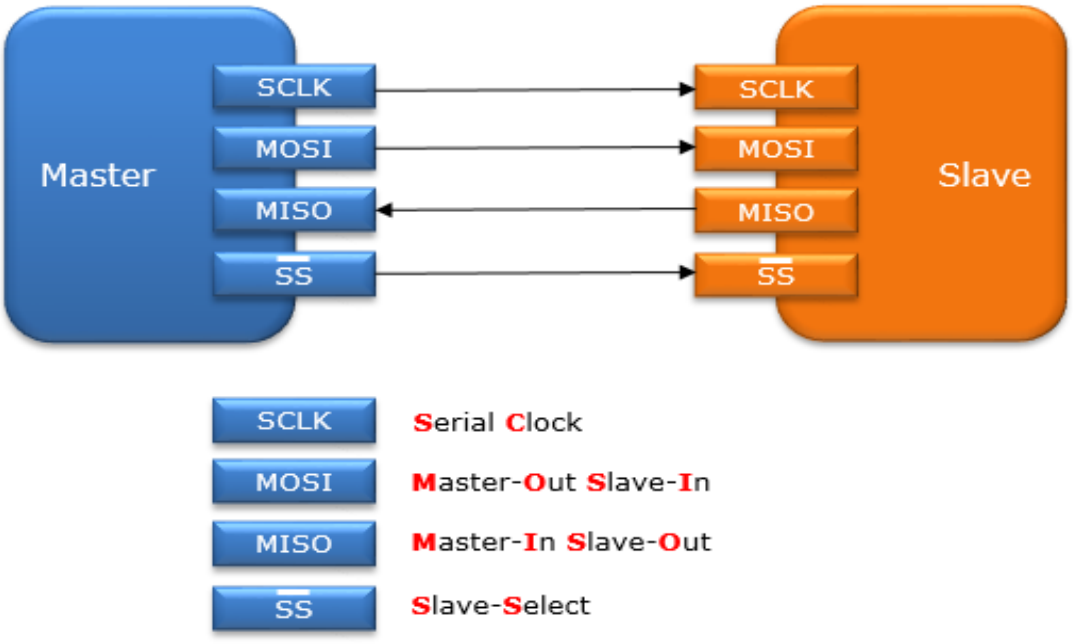

Fig – 1 : SPI Block Diagram

The SPI communication protocol operates in the following manner:

1. The master device generates a clock signal and sends it to the slave devices.
2. The master device selects the slave device it wants to communicate with by asserting the SS wire.
3. The master device transmits data to the slave device over the MOSI wire.
4. The slave device receives the data and transmits its response to the master device over the MISO wire.
5. The master device receives the response from the slave device and processes it accordingly.

Advantages of SPI

1. **High-Speed Data Transfer**: SPI allows for high-speed data transfer rates, making it ideal for applications that require fast data transfer.
2. **Low Power Consumption**: SPI is a low-power protocol, making it suitable for battery-powered devices.
3. **Simple Implementation**: SPI is a simple protocol to implement, requiring minimal hardware and software resources.
4. **Flexibility**: SPI is a flexible protocol that can be used in a wide range of applications, from simple sensors to complex systems.

Applications of SPI

1. **Microcontrollers and Microprocessors**: SPI is widely used in microcontrollers and microprocessors to interface with peripheral devices such as sensors, displays, and memory devices.
2. **Robotics and Automation**: SPI is used in robotics and automation applications to interface with sensors, actuators, and other devices.
3. **IoT Devices**: SPI is used in IoT devices to interface with sensors, displays, and other devices.
4. **Medical Devices**: SPI is used in medical devices such as patient monitoring systems and medical imaging devices.

The SPI communication protocol is a widely used protocol in embedded systems, robotics, and IoT applications due to its simplicity, flexibility, and high-speed data transfer capabilities. Its master-slave architecture, full-duplex communication, and multi-device support make it an ideal protocol for a wide range of applications.

2.2. FEATURES OF SPI

Key Features of SPI

1. Synchronous Communication

SPI is a synchronous protocol, meaning that the data transmission is synchronized with a clock signal. This clock signal is generated by the master device and is used to coordinate the data transmission between the master and slave devices. The clock signal ensures that the data is transmitted and received at the same rate, reducing errors, and increasing reliability.

Advantages of Synchronous Communication:

- Reduced errors: Synchronous communication reduces the likelihood of data transmission errors, as the clock signal ensures that the data is transmitted and received at the same rate.
- Increased reliability: Synchronous communication increases the reliability of the data transmission, as the clock signal ensures that the data is transmitted and received correctly.

2. Full-Duplex Communication

SPI is a full-duplex protocol, meaning that data can be transmitted and received simultaneously. This enables efficient communication between devices and reduces the overall communication time.

Advantages of Full-Duplex Communication:

- Increased efficiency: Full-duplex communication increases the efficiency of the data transmission, as data can be transmitted and received simultaneously.
- Reduced communication time: Full-duplex communication reduces the overall communication time, as data can be transmitted and received in parallel.

3. Serial Communication

SPI is a serial protocol, meaning that data is transmitted one bit at a time over a single wire. This reduces the number of wires required for communication and makes it ideal for applications where space is limited.

Advantages of Serial Communication:

- Reduced wire count: Serial communication reduces the number of wires required for communication, making it ideal for applications where space is limited.
- Simplified design: Serial communication simplifies the design of the communication system, as only a single wire is required for data transmission.

4. Master-Slave Architecture

SPI uses a master-slave architecture, where one device acts as the master and the other devices act as slaves. The master device controls the communication and generates the clock signal, while the slave devices respond to the master's requests.

Advantages of Master-Slave Architecture:

- Simplified design: The master-slave architecture simplifies the design of the communication system, as the master device controls the communication and the slave devices respond to its requests.
- Improved performance: The master-slave architecture improves the performance of the data transmission, as the master device can control the communication and optimize the data transfer rate.

5. Multi-Device Support

SPI allows for multiple slave devices to be connected to a single master device, making it ideal for applications where multiple devices need to be controlled or monitored.

Advantages of Multi-Device Support:

- Increased flexibility: Multi-device support increases the flexibility of the communication system, as multiple devices can be connected to a single master device.
- Improved performance: Multi-device support improves the performance of the data transmission, as multiple devices can be controlled or monitored simultaneously.

6. Mode 0 and Mode 1 Operation

SPI has two modes of operation: Mode 0 and Mode 1. In Mode 0, the clock signal is idle low, and in Mode 1, the clock signal is idle high. The mode of operation is determined by the master device and is used to configure the slave devices.

Advantages of Mode 0 and Mode 1 Operation:

- Increased flexibility: Mode 0 and Mode 1 operation increase the flexibility of the communication system, as the master device can configure the slave devices to operate in either mode.
- Improved performance: Mode 0 and Mode 1 operation improve the performance of the data transmission, as the master device can optimize the clock signal for the specific application.

7. Clock Polarity and Phase

SPI has two clock polarity options: clock polarity 0 (CPOL=0) and clock polarity 1 (CPOL=1). The clock polarity determines the idle state of the clock signal. Additionally, SPI has two clock phase options: clock phase 0 (CPHA=0) and clock phase 1 (CPHA=1). The clock phase determines when the data is sampled.

Advantages of Clock Polarity and Phase:

- Increased flexibility: Clock polarity and phase options increase the flexibility of the communication system, as the master device can configure the clock signal to optimize the data transmission.
- Improved performance: Clock polarity and phase options improve the performance of the data transmission, as the master device can optimize the clock signal for the specific application.

8. Data Transfer Rate

SPI allows for high-speed data transfer rates, making it ideal for applications that require fast data transfer.

Advantages of High-Speed Data Transfer:

- Increased performance: High-speed data transfer rates increase the performance of the communication system, as data can be transmitted quickly and efficiently.
- Improved responsiveness: High-speed data transfer rates improve the responsiveness of the system, as data can be transmitted and received quickly.

2.3. MODES OF SPI

SPI supports various modes of operation, determined by two key parameters: Clock Polarity (CPOL) and Clock Phase (CPHA). These parameters control the timing relationship between the data signal (MOSI/MISO) and the clock signal (SCLK), enabling SPI to operate in four distinct modes.

1. **Clock Polarity (CPOL)**
   - **CPOL = 0**: The clock signal (SCLK) is low when idle. The clock signal starts with a low state and switches to high during the active phase of communication.
   - **CPOL = 1**: The clock signal (SCLK) is high when idle. The clock signal starts with a high state and switches to low during the active phase of communication.

2. **Clock Phase (CPHA)**
   - **CPHA = 0**: Data is sampled on the leading edge of the clock signal. For CPOL = 0, this means data is captured on the rising edge; for CPOL = 1, data is captured on the falling edge.
   - **CPHA = 1**: Data is sampled on the trailing edge of the clock signal. For CPOL = 0, this means data is captured on the falling edge; for CPOL = 1, data is captured on the rising edge.

3. SPI Modes Overview

The combination of CPOL and CPHA defines the four SPI modes:

- **Mode 0 (CPOL = 0, CPHA = 0):**
  - **Clock Polarity**: The clock is low when idle.
  - **Clock Phase**: Data is sampled on the rising edge of the clock (leading edge).
  - **Operation**: Data transmission begins on the first clock edge after the falling edge of the chip select signal (CS). Data is valid when the clock transitions from low to high.

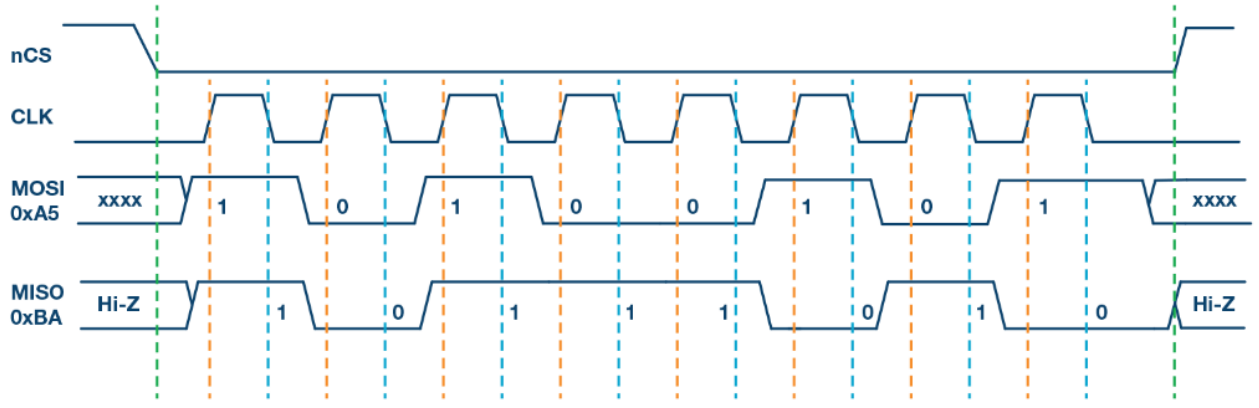

Fig – 2 : SPI Mode 0

- **Mode 1 (CPOL = 0, CPHA = 1):**
  - **Clock Polarity**: The clock is low when idle.
  - **Clock Phase**: Data is sampled on the falling edge of the clock (trailing edge).
  - **Operation**: Data transmission begins on the second clock edge after the falling edge of the chip select signal (CS). Data is valid when the clock transitions from high to low.

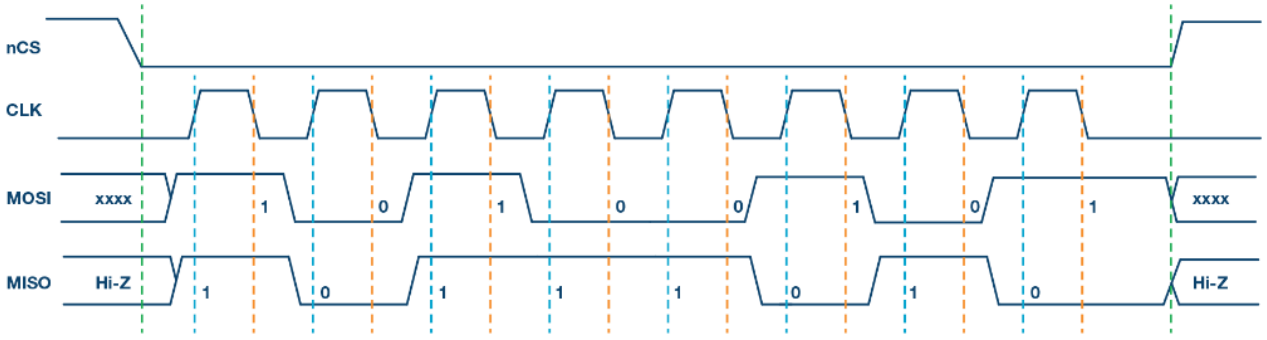

Fig – 3 : SPI Mode 1

- **Mode 2 (CPOL = 1, CPHA = 0):**
  - **Clock Polarity**: The clock is high when idle.
  - **Clock Phase**: Data is sampled on the falling edge of the clock (leading edge).
  - **Operation**: Data transmission begins on the first clock edge after the falling edge of the chip select signal (CS). Data is valid when the clock transitions from high to low.

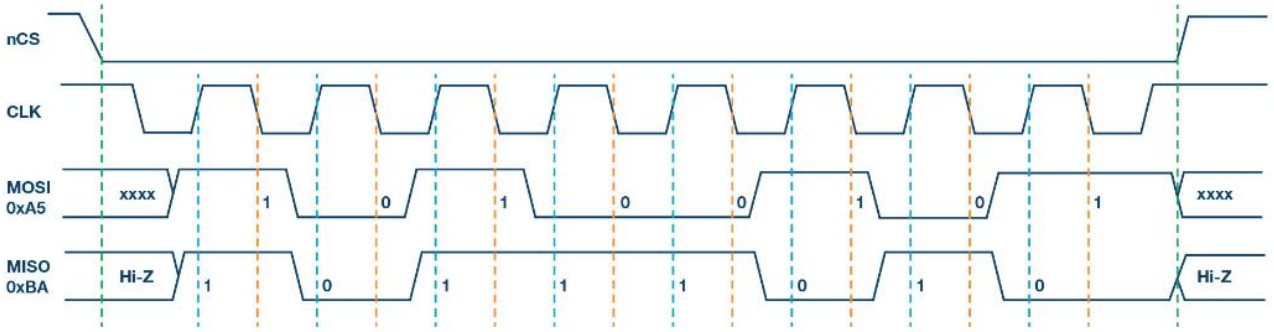

Fig – 4 : SPI Mode 2

- **Mode 3 (CPOL = 1, CPHA = 1):**
  - **Clock Polarity**: The clock is high when idle.
  - **Clock Phase**: Data is sampled on the rising edge of the clock (trailing edge).
  - **Operation**: Data transmission begins on the second clock edge after the falling edge of the chip select signal (CS). Data is valid when the clock transitions from low to high.

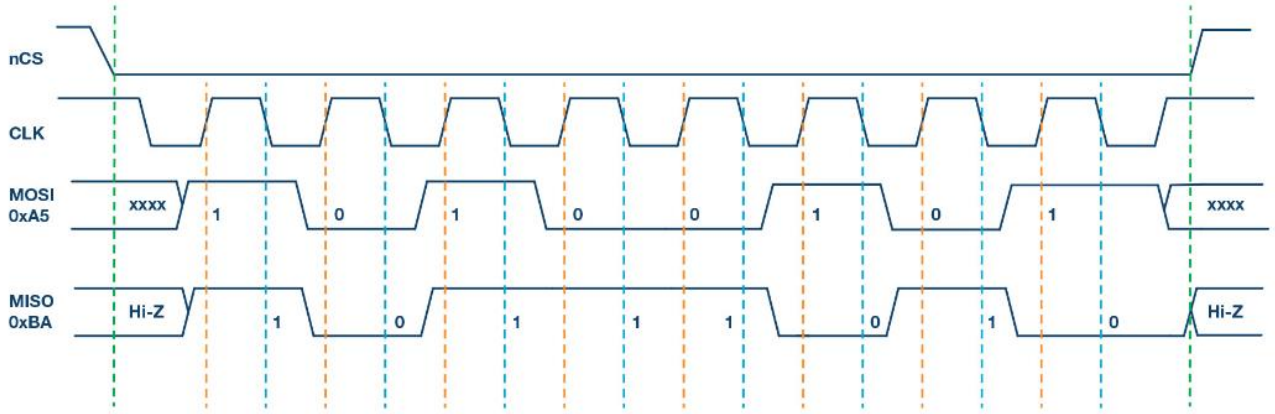

Fig – 5 : SPI Mode 3

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

Table – 1 : SPI Modes

2.4. SPI MASTER

The SPI Master is a synchronous serial communication interface that enables data exchange between a master device (typically a microcontroller) and one or multiple slave devices (peripherals, sensors, or other microcontrollers). The communication process is initiated and controlled by the master device, which orchestrates the data transfer sequence.

Data Transfer Initiation

The master device initiates a data transfer by asserting the Slave Select (SS) line, which is connected to the slave device. This signal indicates to the slave device that it is being addressed and should prepare to receive or transmit data. The SS line is typically active low, meaning that the master device sets it to a logic low level to select the slave device.

Clock Generation

The master device generates a clock signal, known as the Serial Clock (SCK), which is transmitted to the slave device. The SCK signal is used to synchronize the data transfer and defines the bit transfer rate. The clock frequency can vary depending on the specific application and the devices involved.

Data Transmission

Once the slave device is selected and the clock signal is generated, the master device begins transmitting data through the Master Out Slave In (MOSI) line. The data is transmitted one bit at a time, with each bit being synchronized with the SCK signal. The MOSI line is used for data transmission from the master device to the slave device.

Data Reception

The slave device receives the data transmitted by the master device and responds by transmitting data back to the master device through the Master In Slave Out (MISO) line. The MISO line is used for data transmission from the slave device to the master device.

Data Transfer Sequence

The data transfer sequence is as follows:

1. The master device asserts the SS line to select the slave device.
2. The master device generates the SCK signal to synchronize the data transfer.
3. The master device transmits data through the MOSI line, one bit at a time, synchronized with the SCK signal.
4. The slave device receives the data and responds by transmitting data back to the master device through the MISO line.
5. The master device receives the data transmitted by the slave device and de-asserts the SS line to end the data transfer sequence.

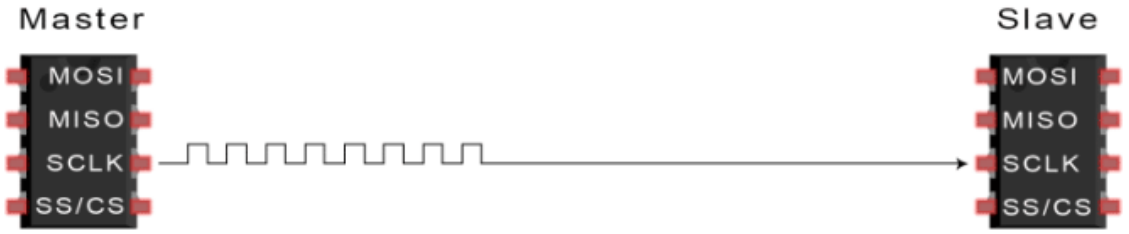

Fig – 6 : The master outputs the clock signal

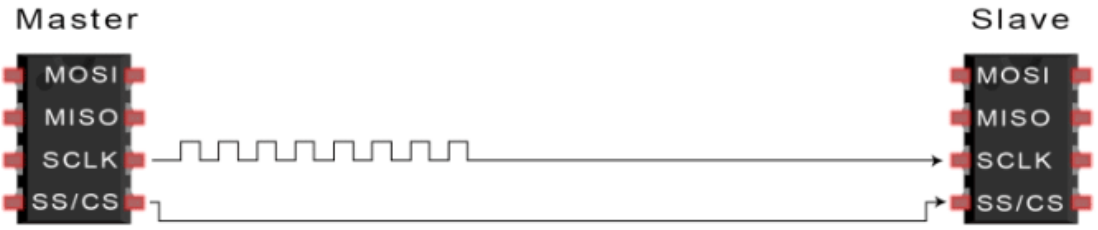

Fig – 7 : The master switches the SS/CS pin to a low voltage state, which activates the slave

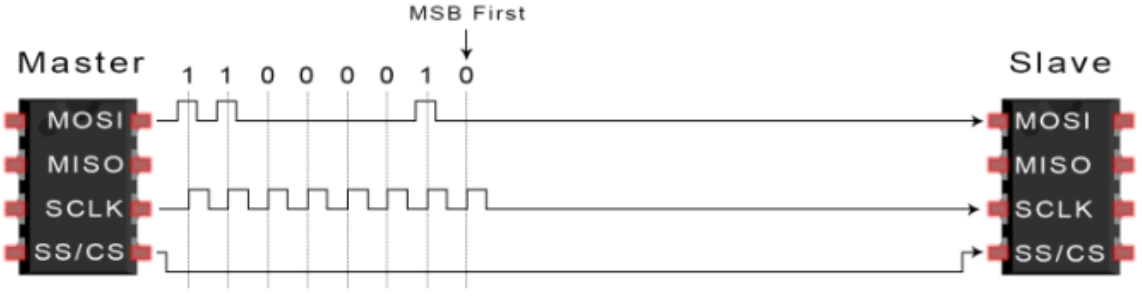

Fig – 8 : The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received

2.5. SPI SLAVE

The SPI Slave is a synchronous serial communication interface that enables data exchange between a slave device (typically a peripheral or sensor) and a master device (typically a microcontroller). The slave device responds to the master device's commands and transmits data back to the master device.

Slave Device Selection

The slave device is selected by the master device through the Slave Select (SS) line. When the SS line is asserted (typically set to a logic low level), the slave device is activated and prepares to receive or transmit data.

Clock Reception

The slave device receives the Serial Clock (SCK) signal generated by the master device. The SCK signal is used to synchronize the data transfer and defines the bit transfer rate.

Data Reception

The slave device receives data transmitted by the master device through the Master Out Slave In (MOSI) line. The data is received one bit at a time, with each bit being synchronized with the SCK signal.

Data Transmission

The slave device transmits data back to the master device through the Master In Slave Out (MISO) line. The data is transmitted one bit at a time, with each bit being synchronized with the SCK signal.

Data Transfer Sequence

The data transfer sequence from the slave device's perspective is as follows:

1. The slave device is selected by the master device through the SS line.
2. The slave device receives the SCK signal generated by the master device.
3. The slave device receives data transmitted by the master device through the MOSI line.
4. The slave device processes the received data and prepares to transmit data back to the master device.
5. The slave device transmits data back to the master device through the MISO line.
6. The slave device is deselected by the master device through the SS line, indicating the end of the data transfer sequence.

Slave Device Operation

The slave device operates in one of the following modes:

- **Receive-only mode**: The slave device only receives data from the master device and does not transmit any data back.
- **Transmit-only mode**: The slave device only transmits data to the master device and does not receive any data.
- **Receive-transmit mode**: The slave device receives data from the master device and transmits data back to the master device.

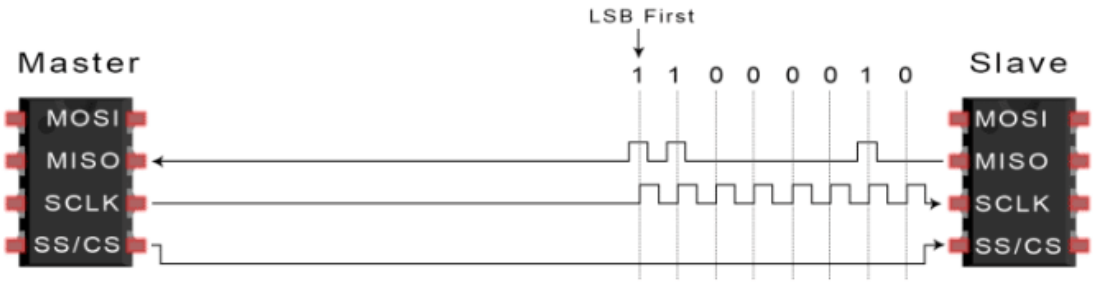

Fig – 9 : If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received

CHAPTER 3 : CONFIGURATIONS OF SPI

3.1. INDEPENDENT SLAVE CONFIGURATION

In an independent slave configuration, each slave device is connected to the SPI bus with its own dedicated Chip Select (CS) line. This setup allows the master device to communicate with each slave individually by selecting the appropriate CS line.

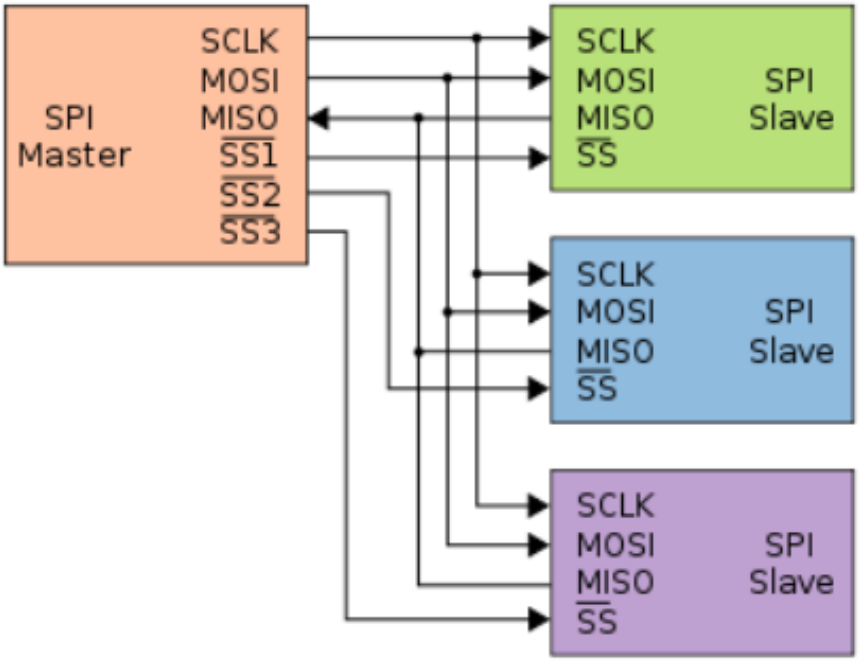

Fig – 10 : SPI Independent slave configuration block diagram

- Chip Select Lines:
  - Each slave device has its own CS line, which is used by the master to enable or disable communication with that specific slave.
  - When the master wants to communicate with a particular slave, it pulls that slave's CS line low (active) while keeping other slaves' CS lines high (inactive).

- Communication:
  - The SPI bus consists of four main lines: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock), and the CS line for each slave.
  - Data transfer occurs between the master and the selected slave through the MISO and MOSI lines, while the SCK line provides the clock signal.

- Bus Sharing:
  - The MISO, MOSI, and SCK lines are shared among all slaves, but the CS lines are used to ensure that only one slave device communicates with the master at any given time.

- Advantages:
  - Simplicity: Each slave device operates independently, simplifying the communication protocol.
  - No Data Collision: Since only one CS line is active at a time, there is no risk of data collisions on the shared SPI lines.

- Disadvantages:
  - Number of Pins: Each additional slave requires a separate CS line, which can be a limitation if the number of GPIO pins on the master device is limited.
  - Wiring Complexity: As the number of slaves increases, the number of wires can become cumbersome, leading to potential complexity in the physical layout.

Example Use Case:

- Sensor Network: A microcontroller communicating with multiple sensors, where each sensor has a dedicated CS line, making it easy to individually address and read data from each sensor.

3.2. DAISY CHAIN CONFIGURATION

In a daisy chain configuration, multiple slave devices are connected in series. The data output from one slave device is fed into the input of the next slave in the chain. This reduces the number of CS lines required to just one, but introduces additional complexity in managing data flow.

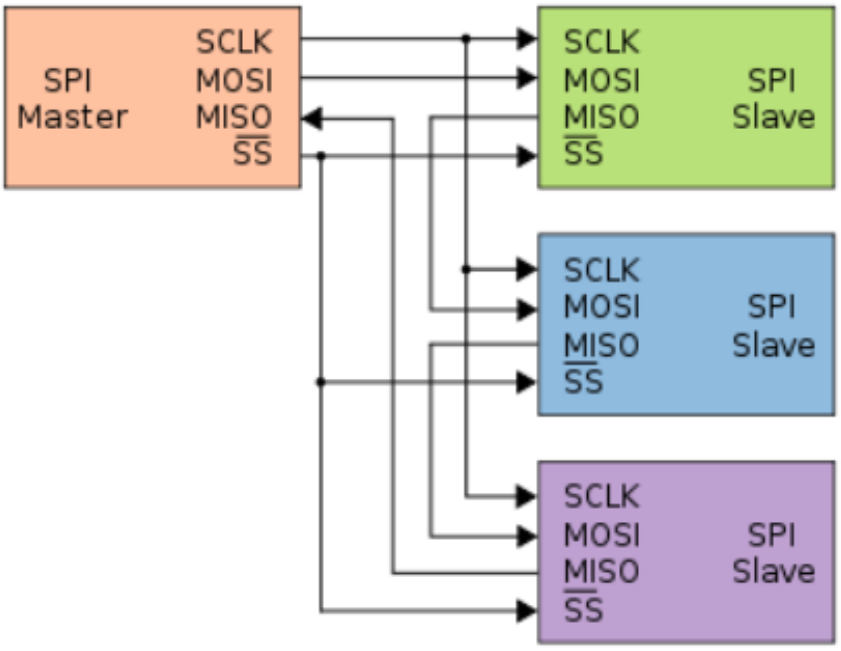

Fig – 11 : SPI Daisy chain configuration block diagram

- Serial Data Path:
  - Each slave device is connected in series such that the output of one device (MISO) becomes the input (MOSI) of the next device in the chain.
  - Data is shifted through the chain, and each slave receives and/or forwards data sequentially.

- Single CS Line:
  - Only one CS line is used for the entire chain. This line enables or disables the chain of slave devices.
  - The master communicates with the entire chain of devices simultaneously, and each slave responds based on its position in the chain and the data it receives.

- Data Propagation:
    - Data is shifted serially from the master to the last slave in the chain and back. This can introduce latency, as each device must process and forward data to the next device.

- Advantages:
    - Reduced Pin Count: Requires fewer GPIO pins since only one CS line is needed for multiple slaves.
    - Simplified Wiring: Reduces the complexity of the wiring by minimizing the number of CS lines.

- Disadvantages:
    - Increased Latency: Data must pass through each slave in the chain, which can introduce delays, especially with long chains.
    - Complex Data Management: Ensuring that each slave receives and processes data correctly can be complex and requires careful management of the data shifting process.

Example Use Case:

- LED Displays: A system where multiple LED driver ICs are connected in series to control a large display, where reducing the number of control lines is beneficial.

Conclusion:

Both configurations offer different benefits and drawbacks:

- Independent Slave Configuration is straightforward and suitable for systems where individual addressing of multiple devices is required, and the number of GPIO pins is not a constraint.

- Daisy Chain Configuration is advantageous in scenarios where reducing the number of control lines is critical, though it adds complexity in data management and may introduce latency.

3.3. MESSAGE STRUCTURE

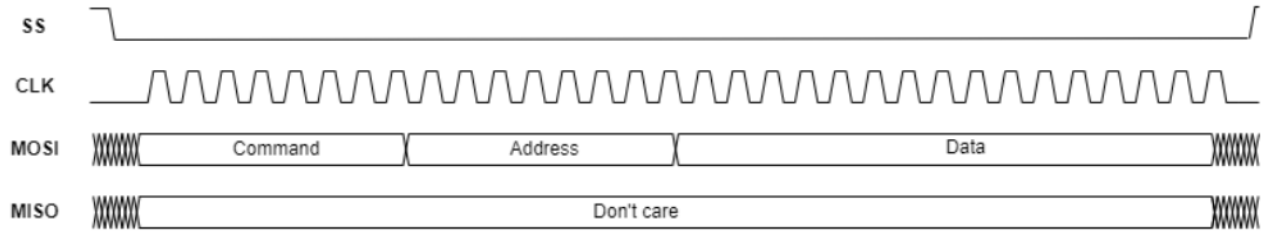

Fig – 12 : SPI Write Frame

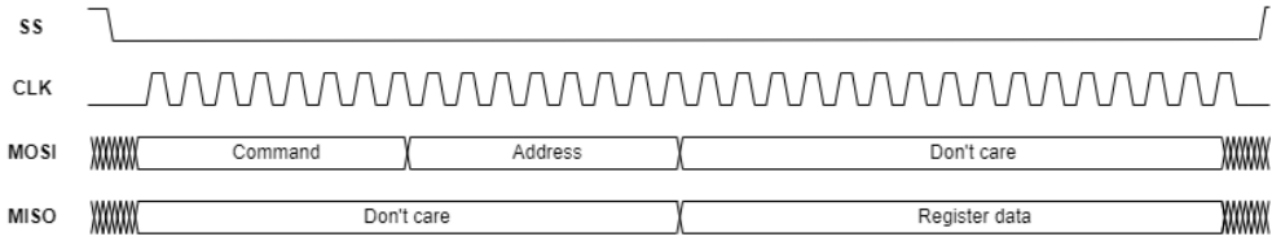

Fig – 13 : Read frame with output frame in the same message

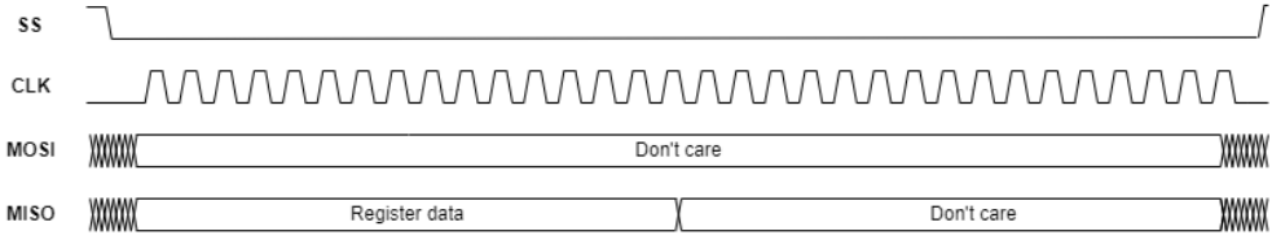

Fig – 14 : Output frame in a new message

CHAPTER 4 : RESULTS AND TOOLS USED

4.1. SIMULATION RESULTS

4.1.1. MASTER

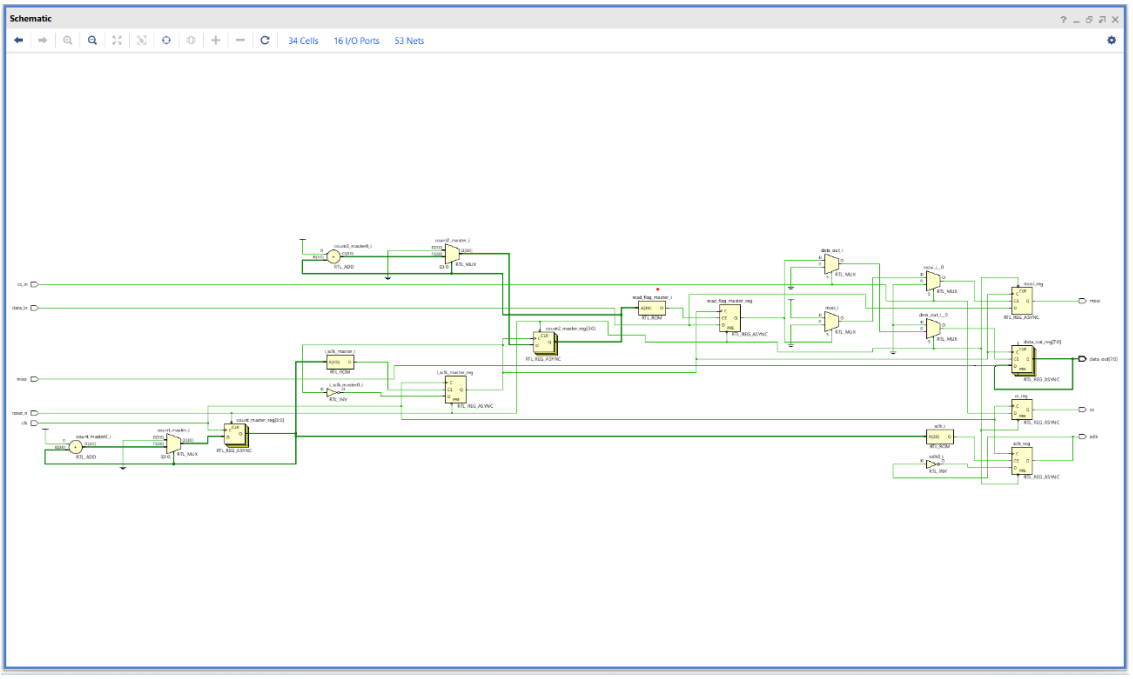

Fig – 15 : Elaborated design of master (Pre-synthesis)

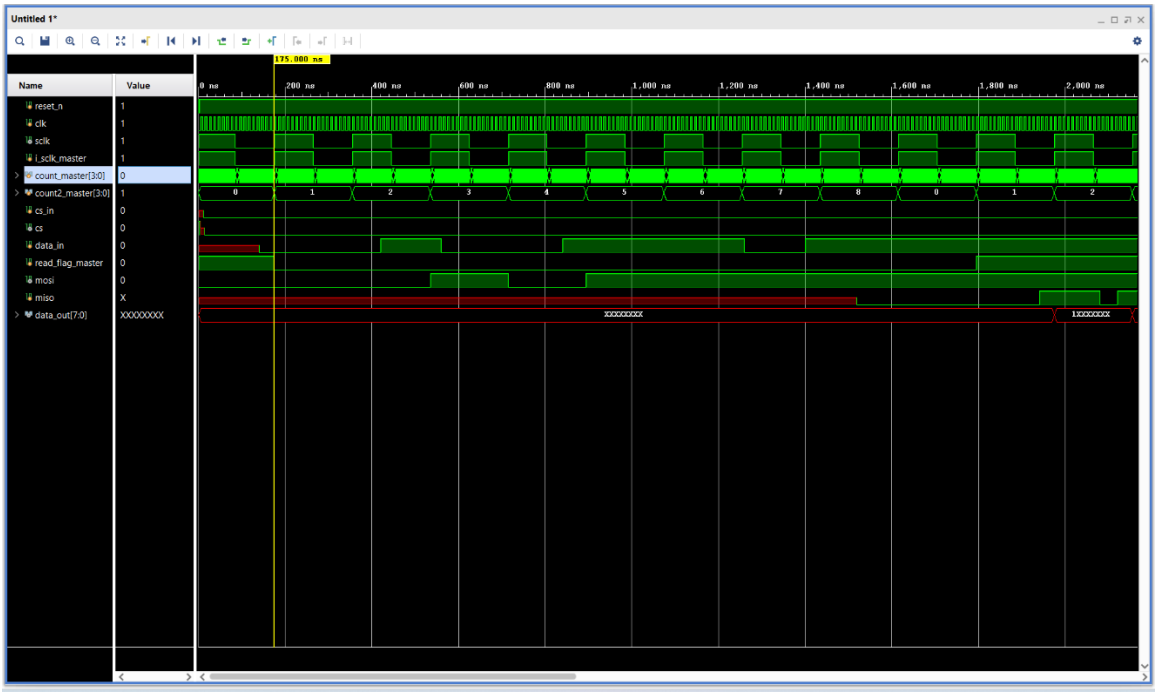

Fig – 16 : Write operation (MOSI drive)

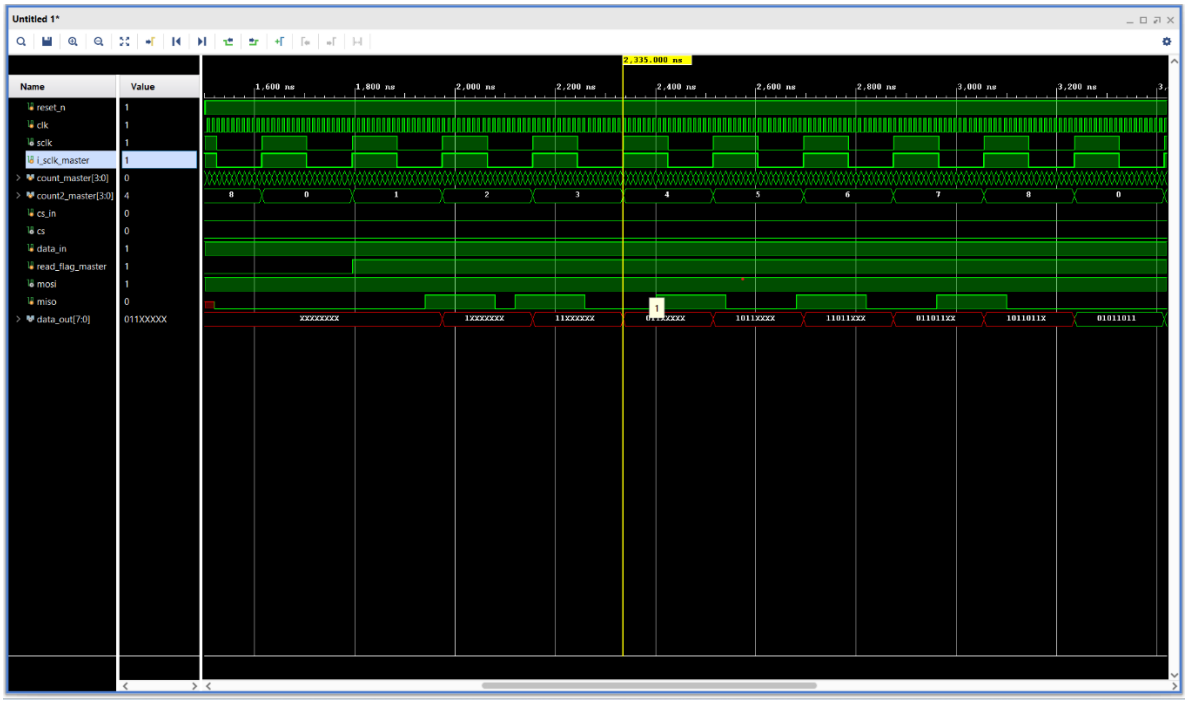

Fig – 17 : Read operation (Sampling MISO)

4.1.2. SLAVE

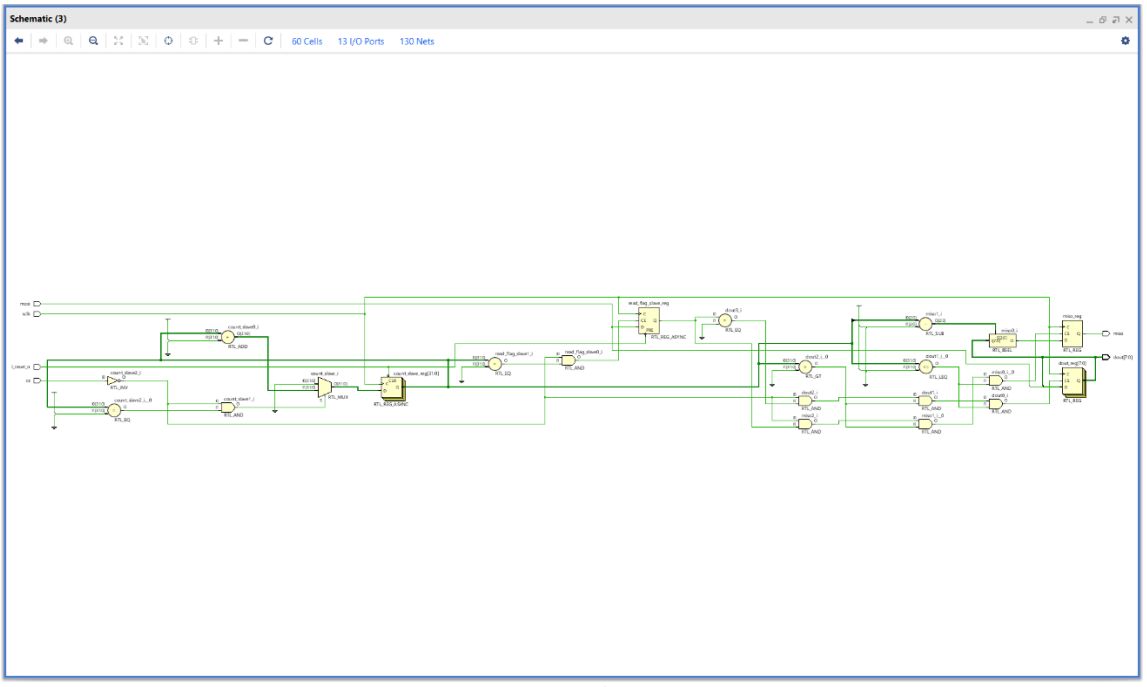

Fig – 18 : Elaborated design of slave (Pre-synthesis)

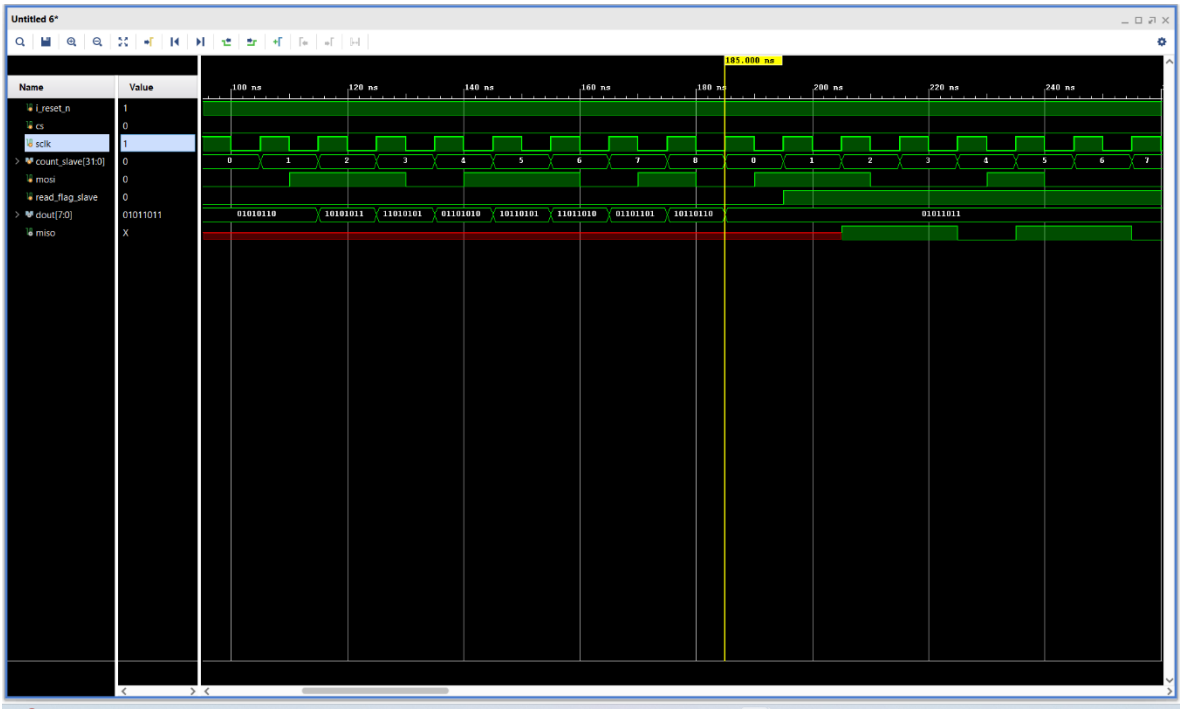

Fig – 19 : Write operation

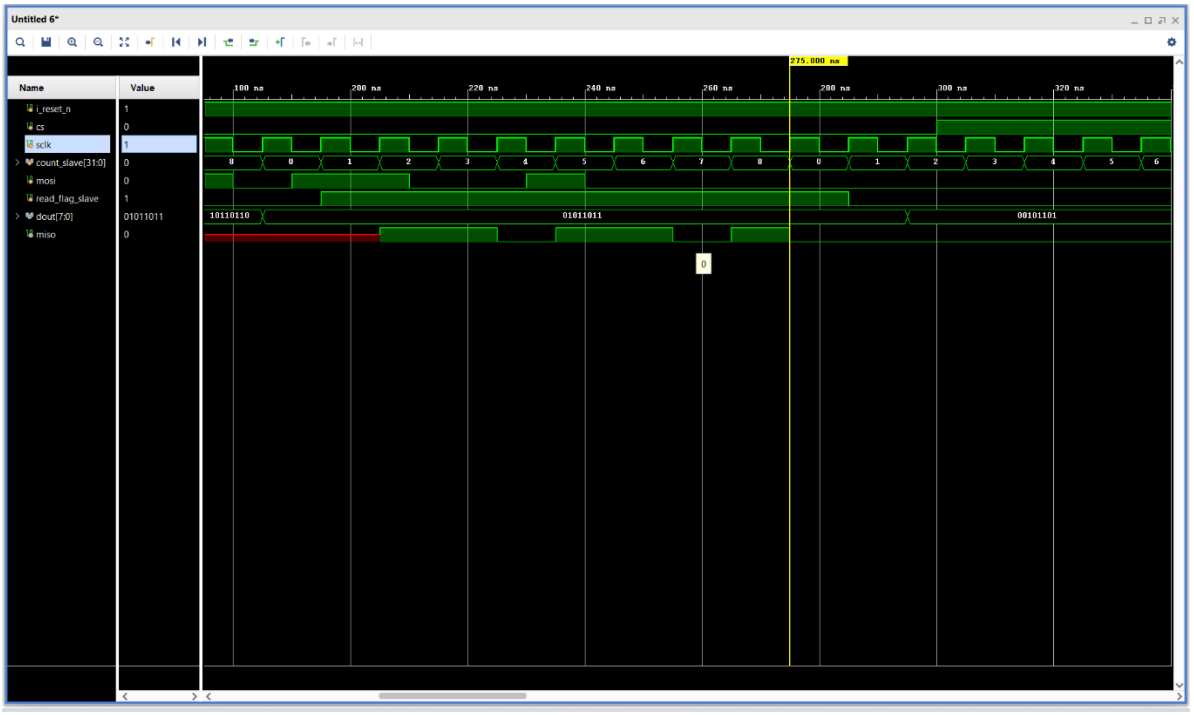

Fig – 20 : Read operation

4.1.3. TOP MODULE

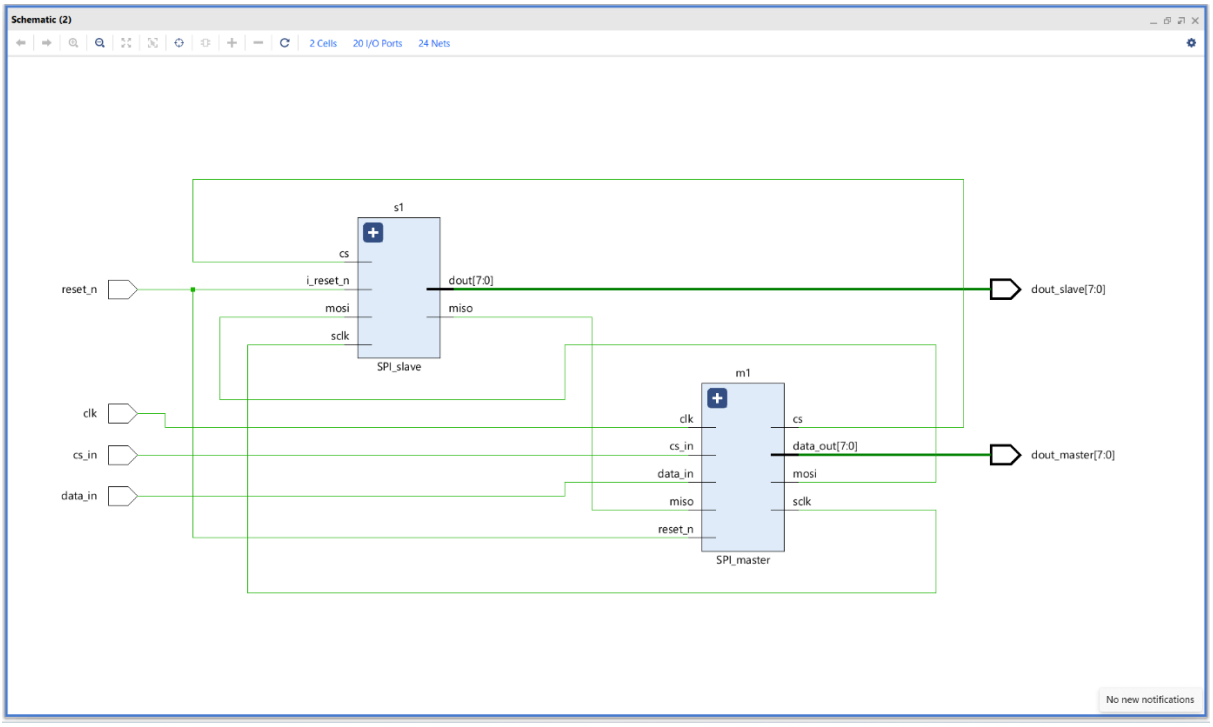

Fig – 21 : Schematic of top module (Pre-synthesis)

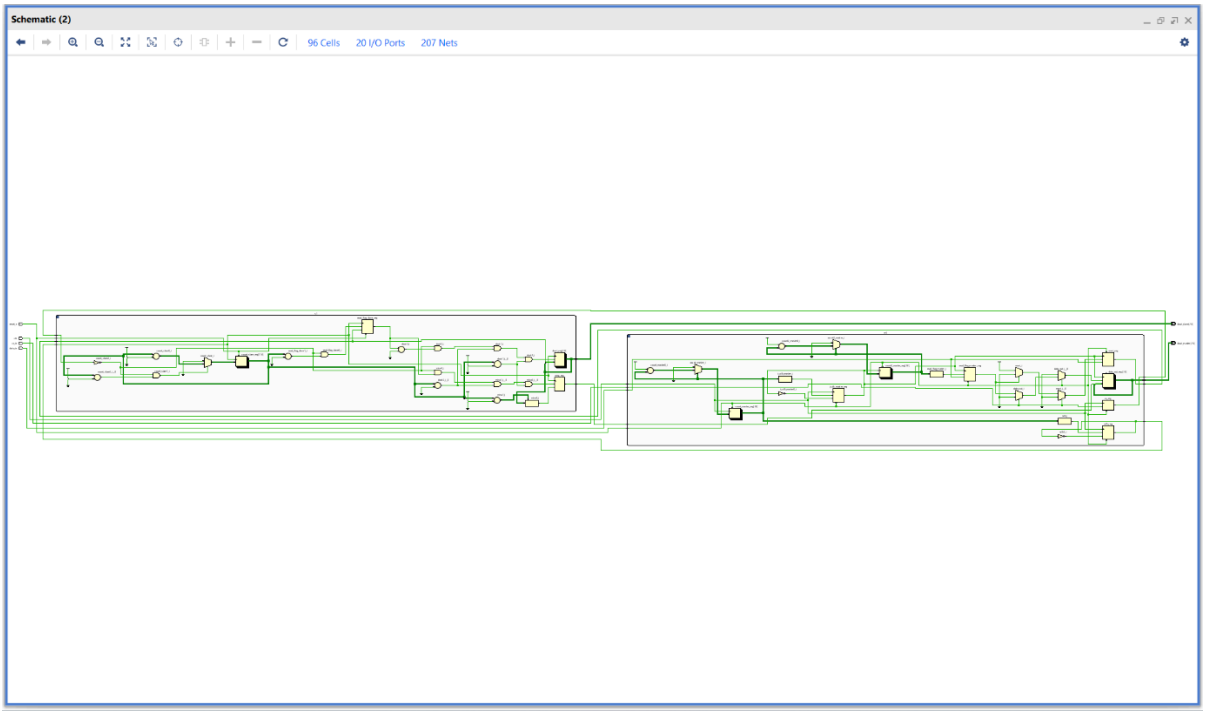

Fig – 22 : Elaborated design of top module (Pre-synthesis)

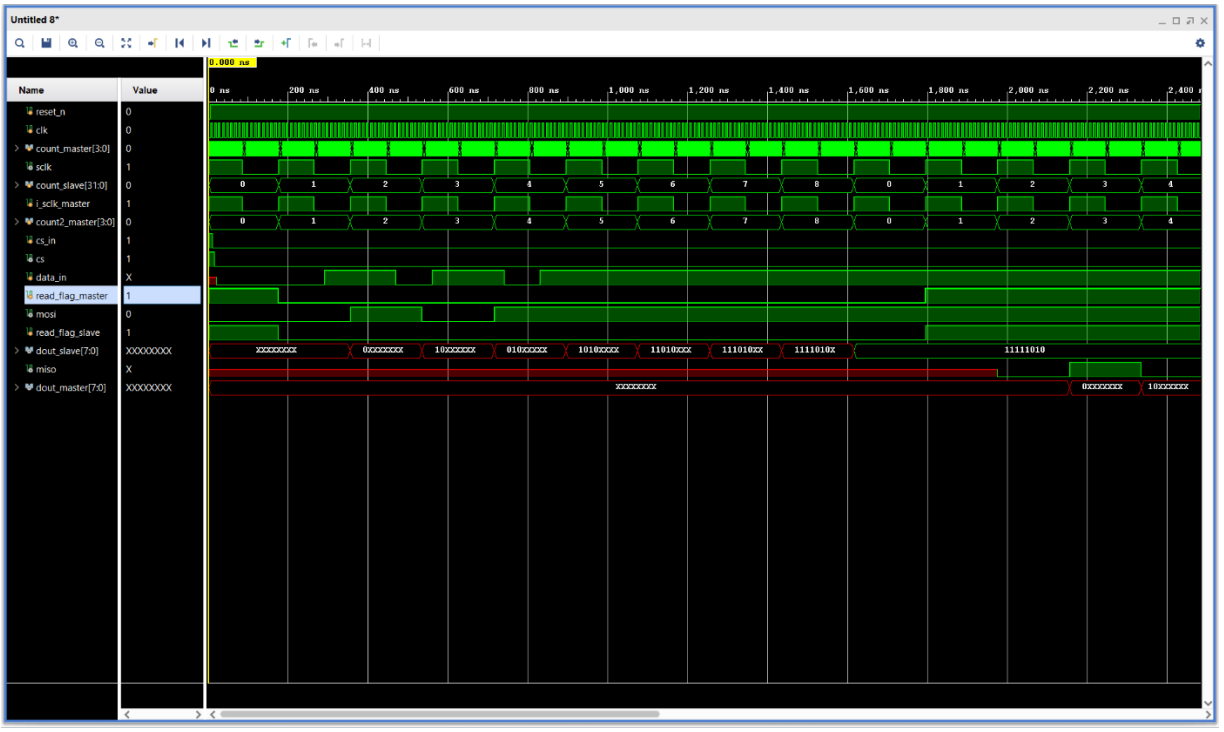

Fig – 23 : Write operation

Fig – 24 : Read operation

4.2. SYNTHESIS RESULTS

4.2.1. MASTER

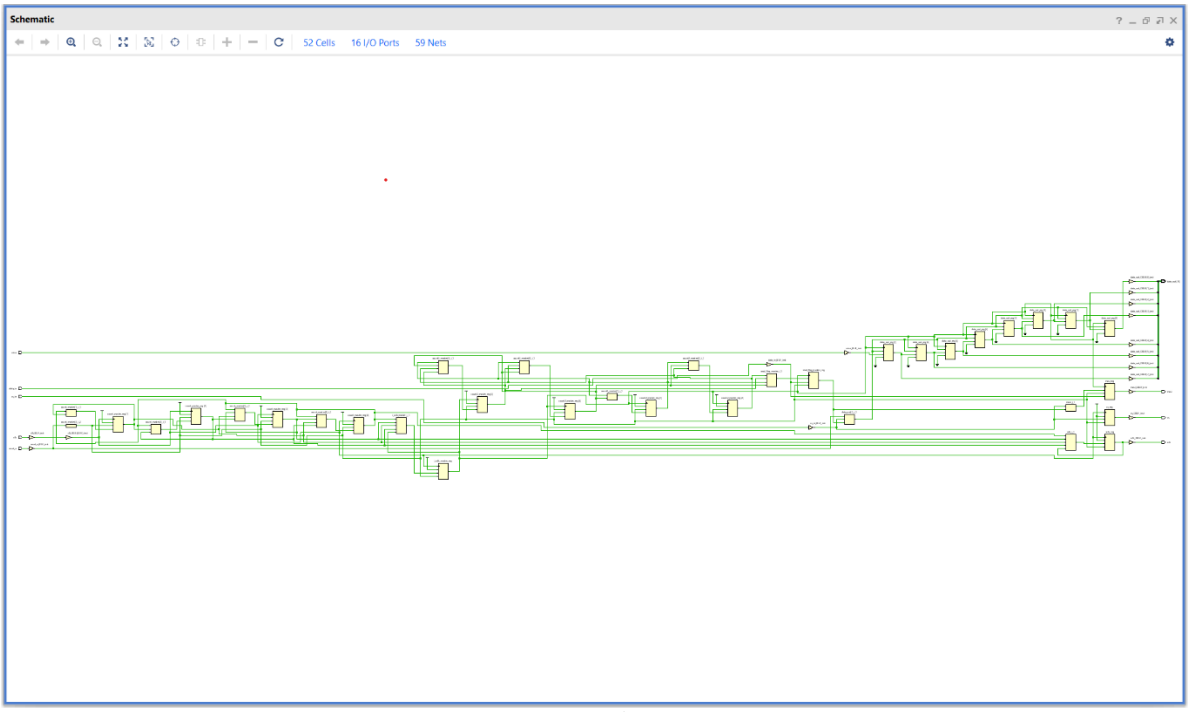

Fig – 25 : Elaborated design of master (Post-synthesis)

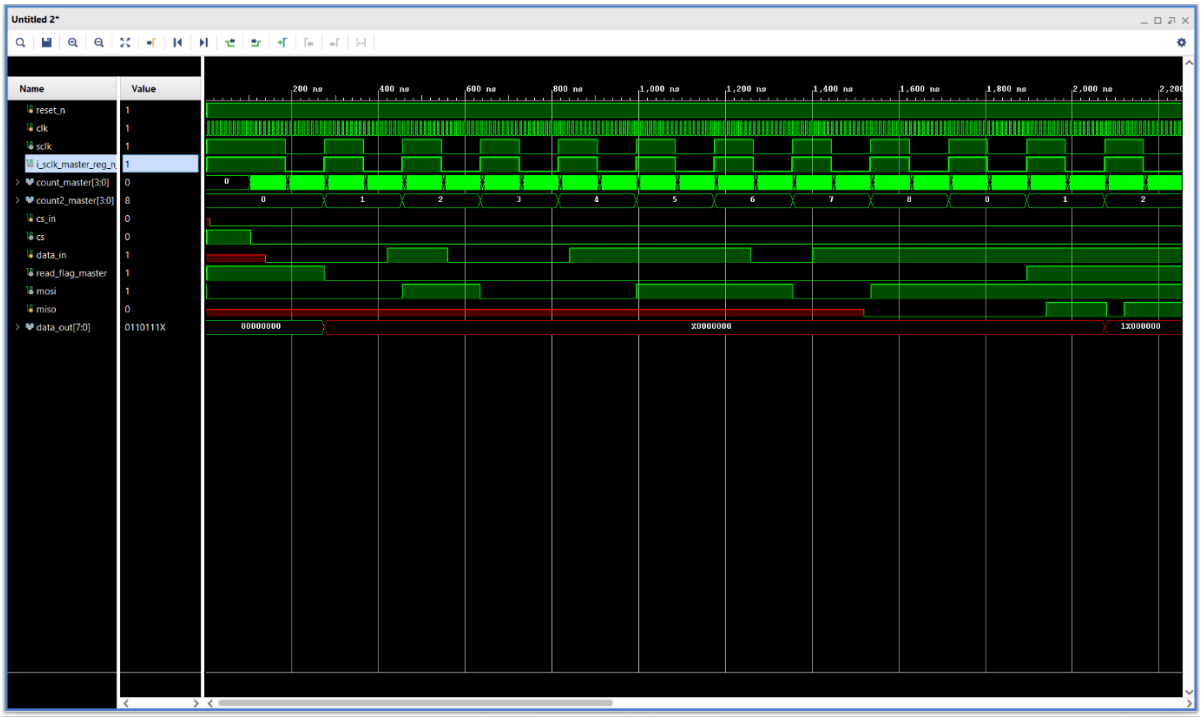

Fig – 26 : Write operation

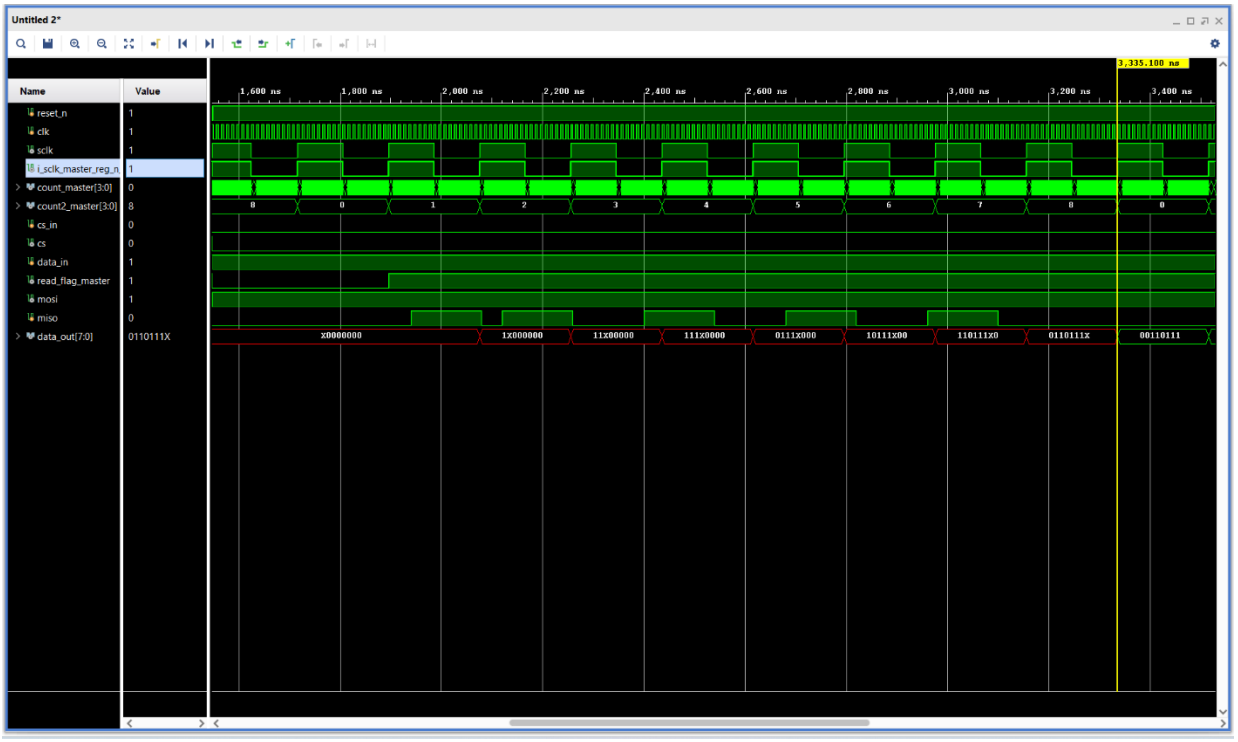

Fig – 27 : Read operation

4.2.2. SLAVE

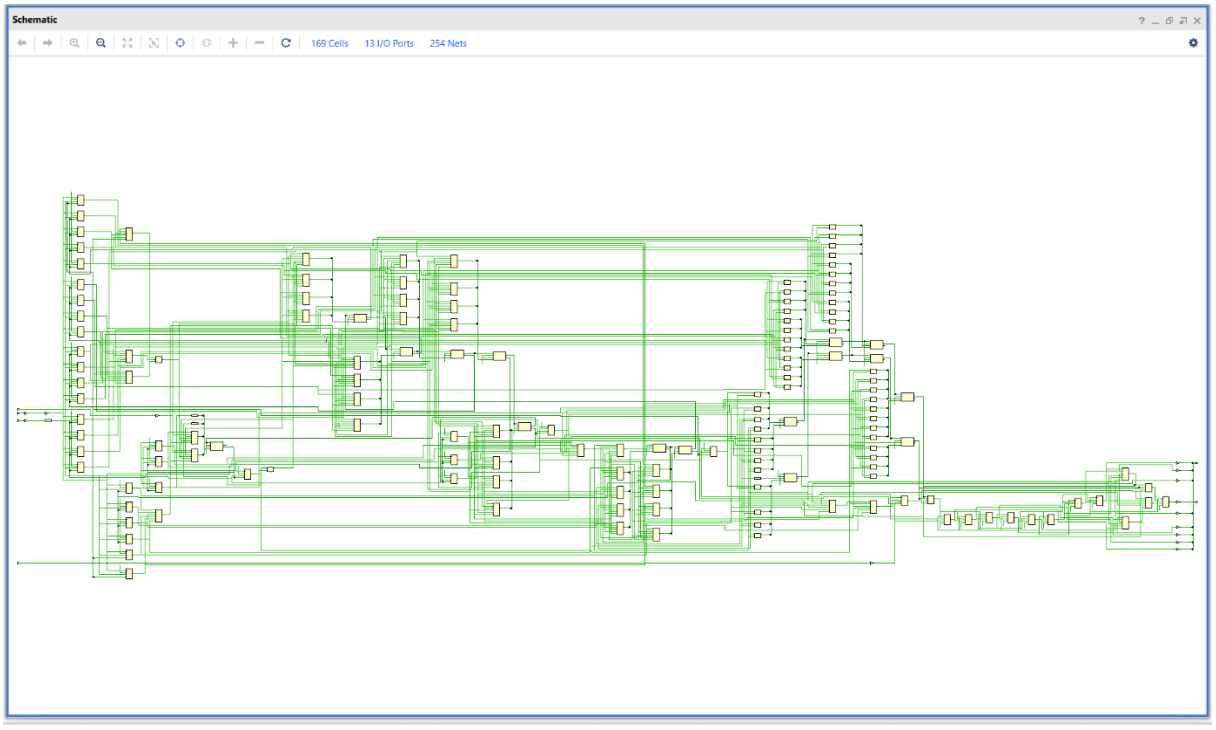

Fig – 28 : Elaborated design of slave (Post-synthesis)

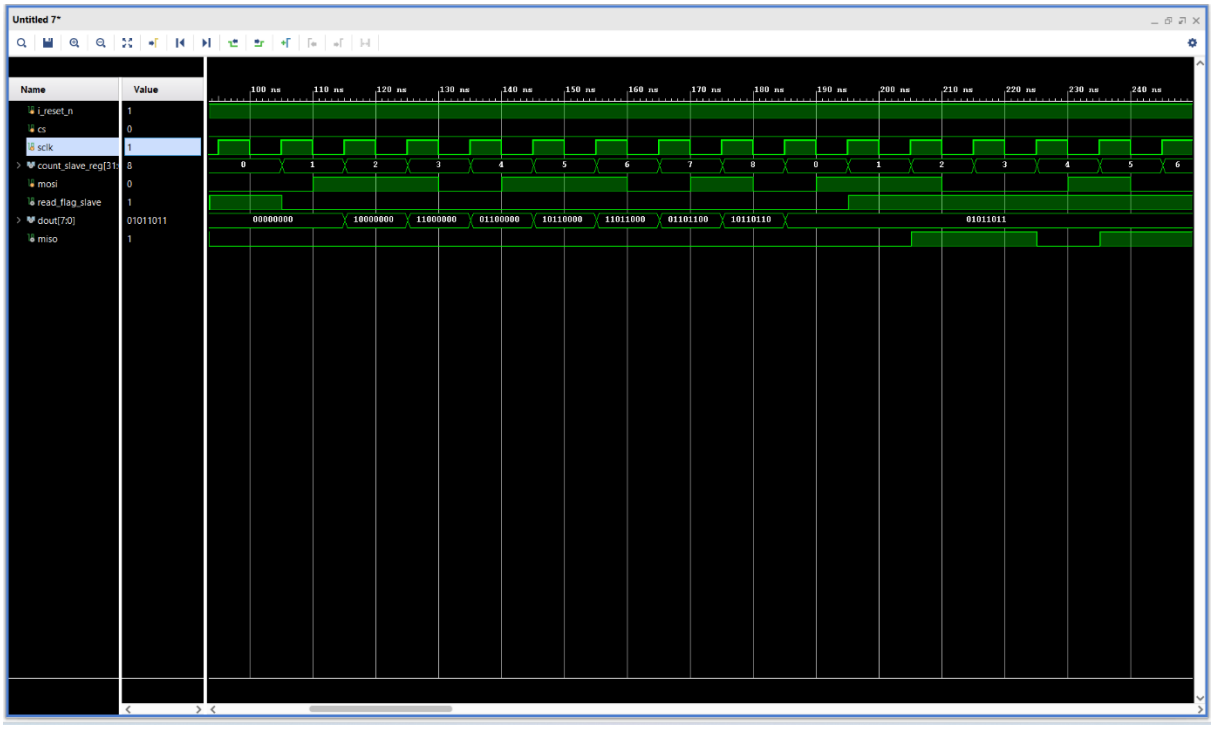

Fig – 29 : Write operation

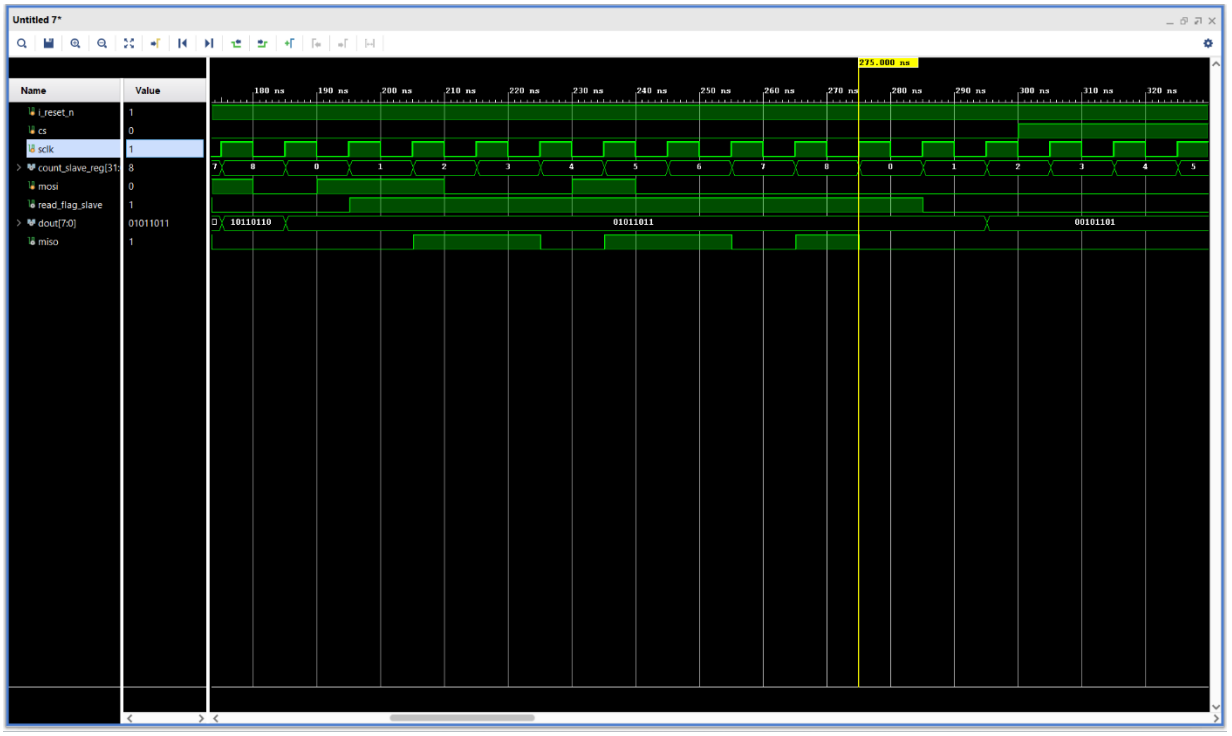

Fig – 30s : Read operation

4.2.3. TOP MODULE

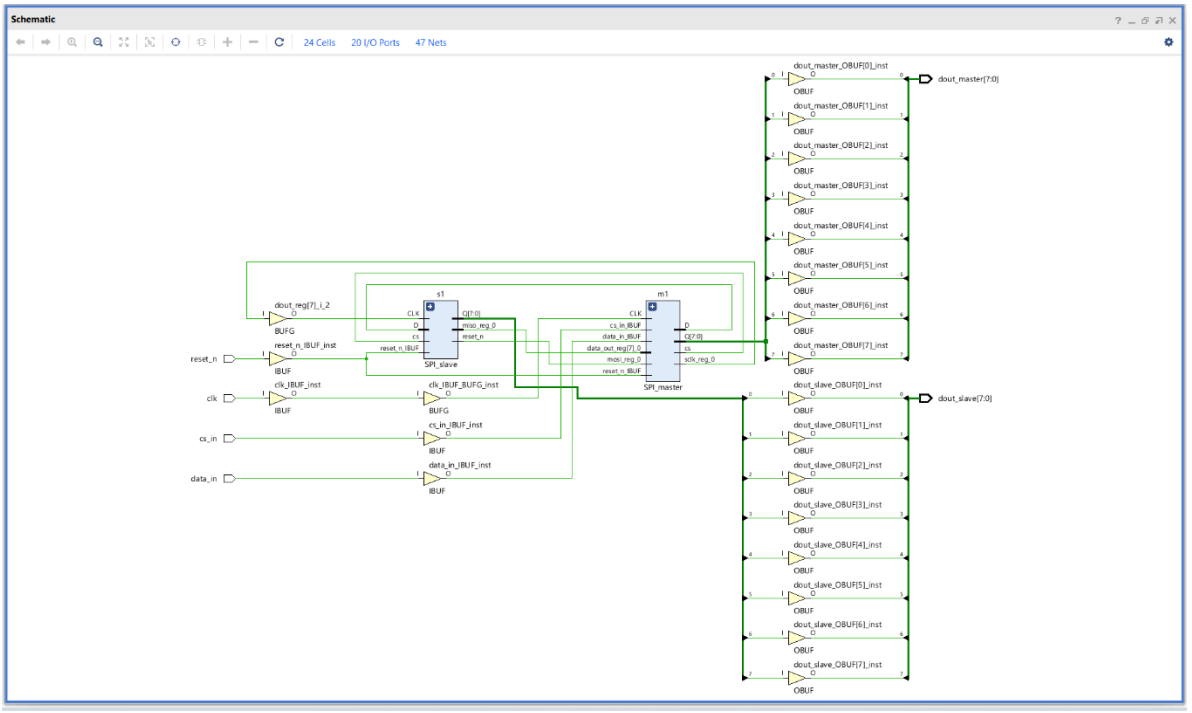

Fig – 31 : Schematic of top module (Post-synthesis)

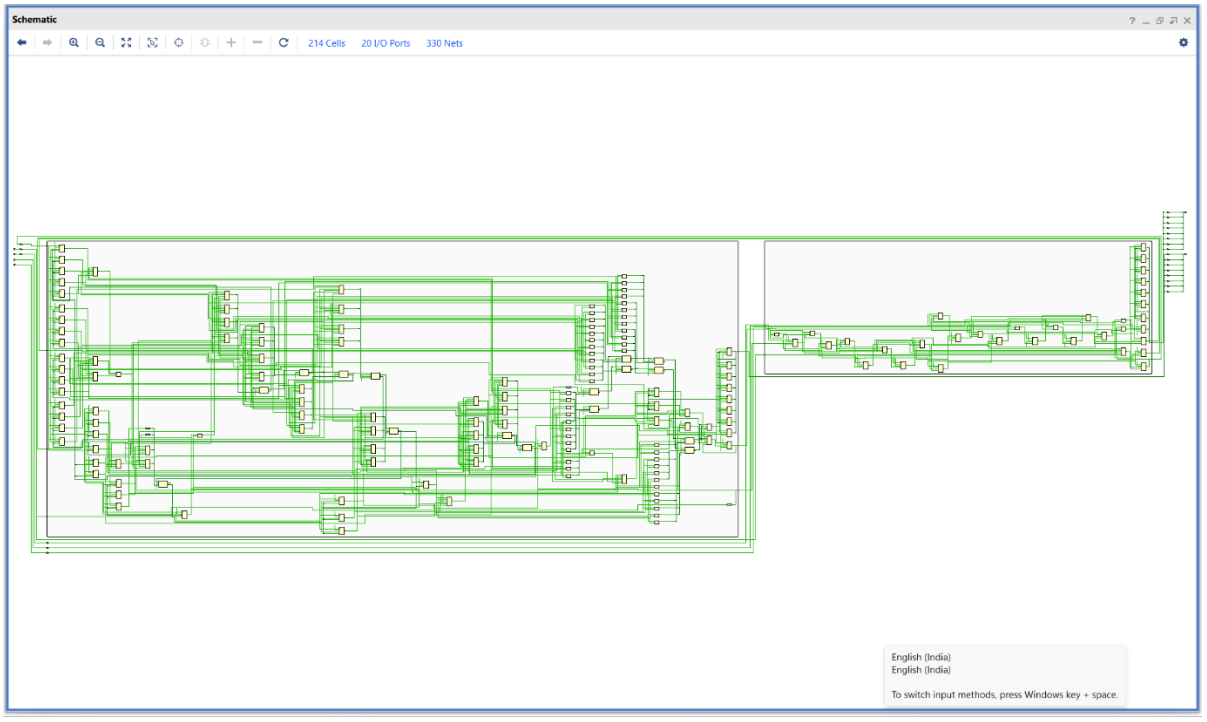

Fig – 32 : Elaborated design of top module (Post-synthesis)

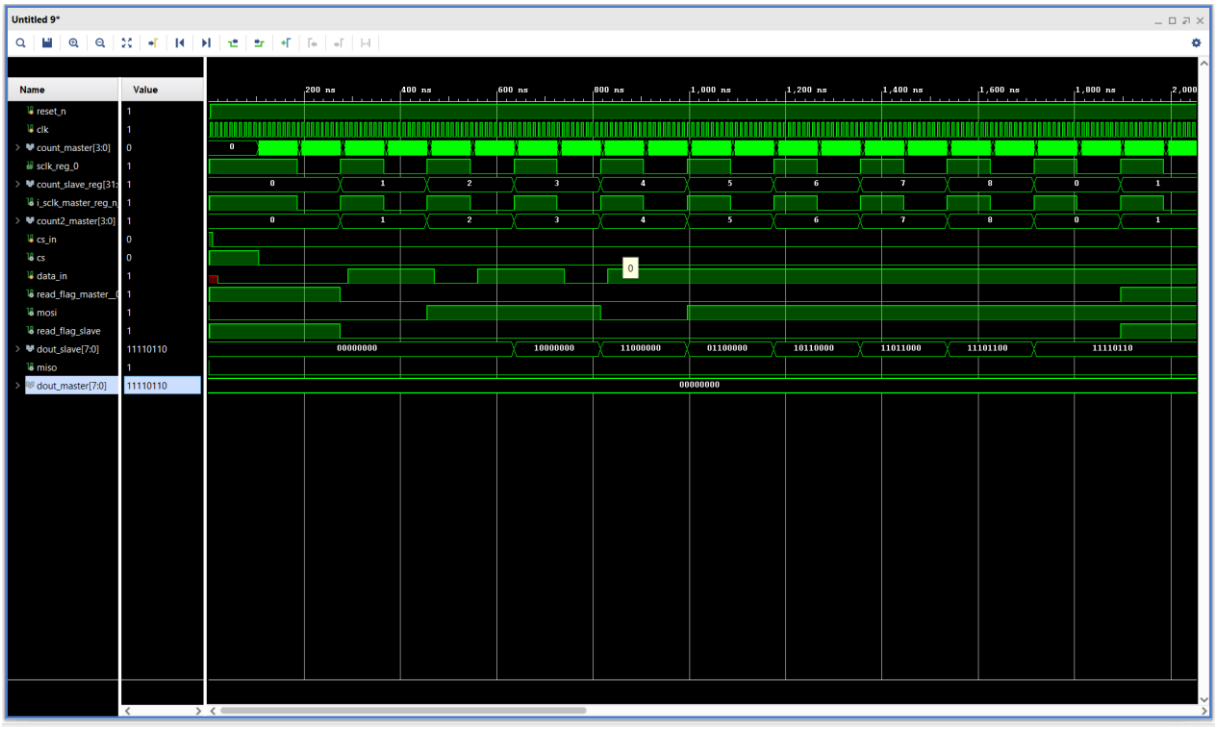

Fig – 33 : Write operation

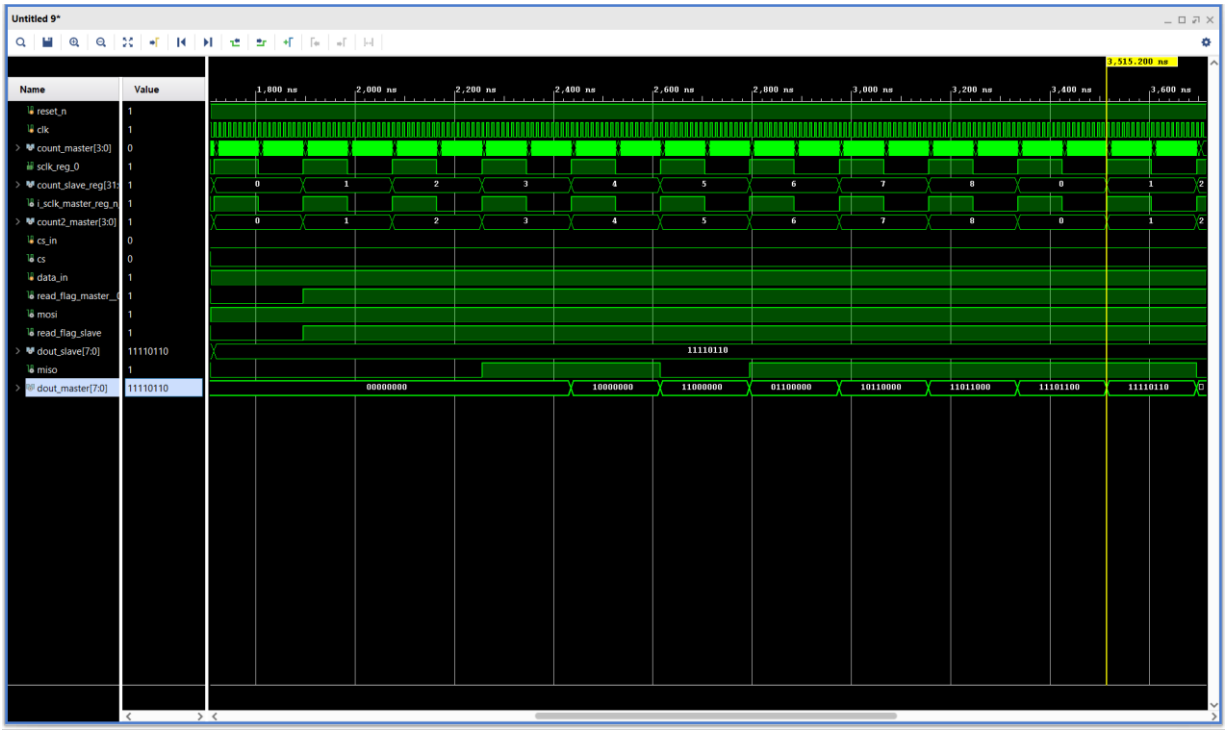

Fig – 34 : Read operation

CHAPTER 5 : POWER TIMING UTILIZATION

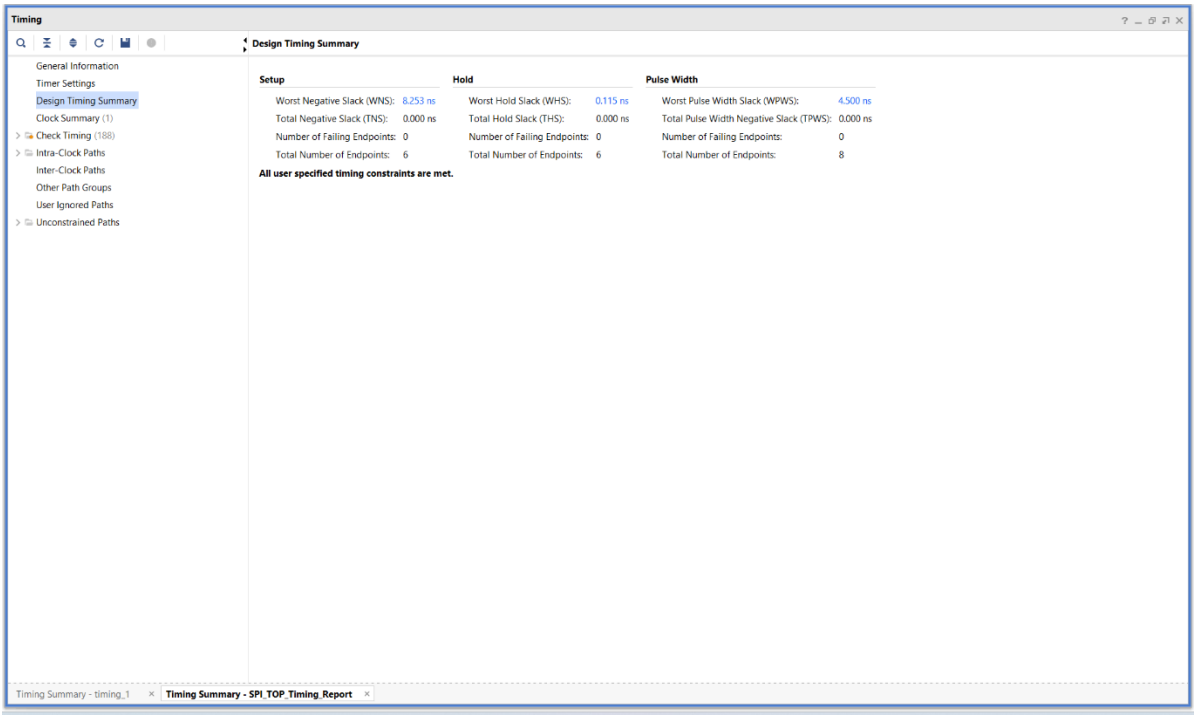

Fig – 35 : Timing report

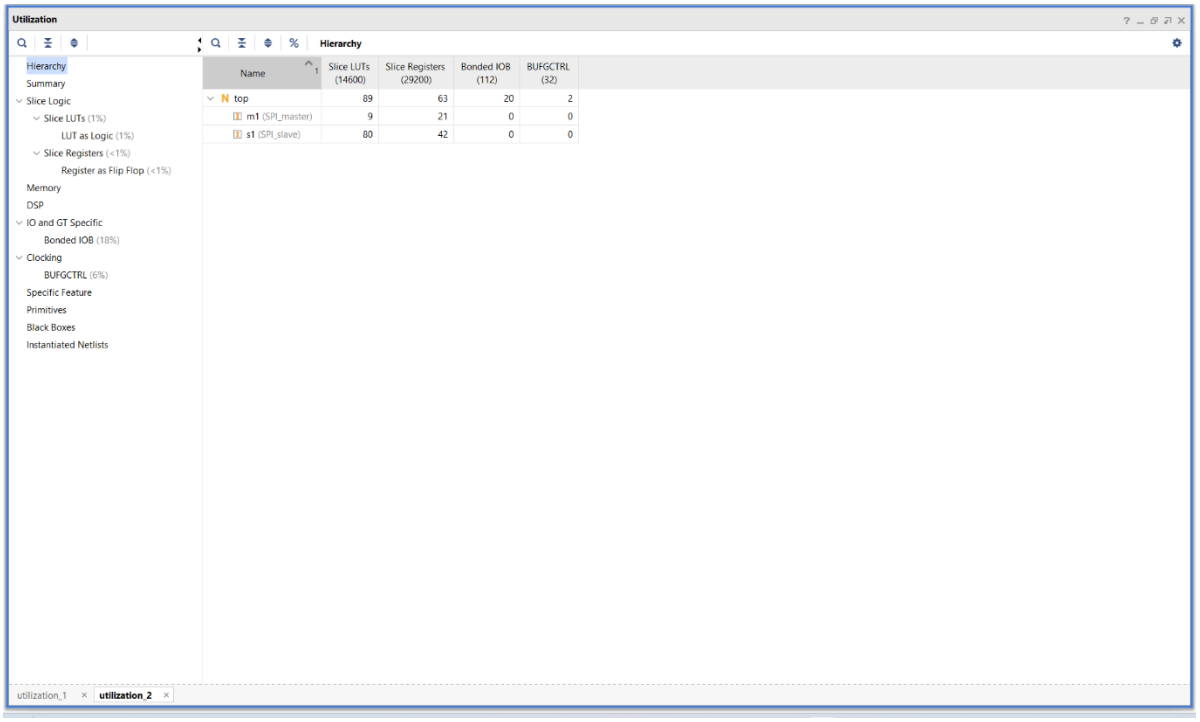

Fig – 36 : Resource utilization report

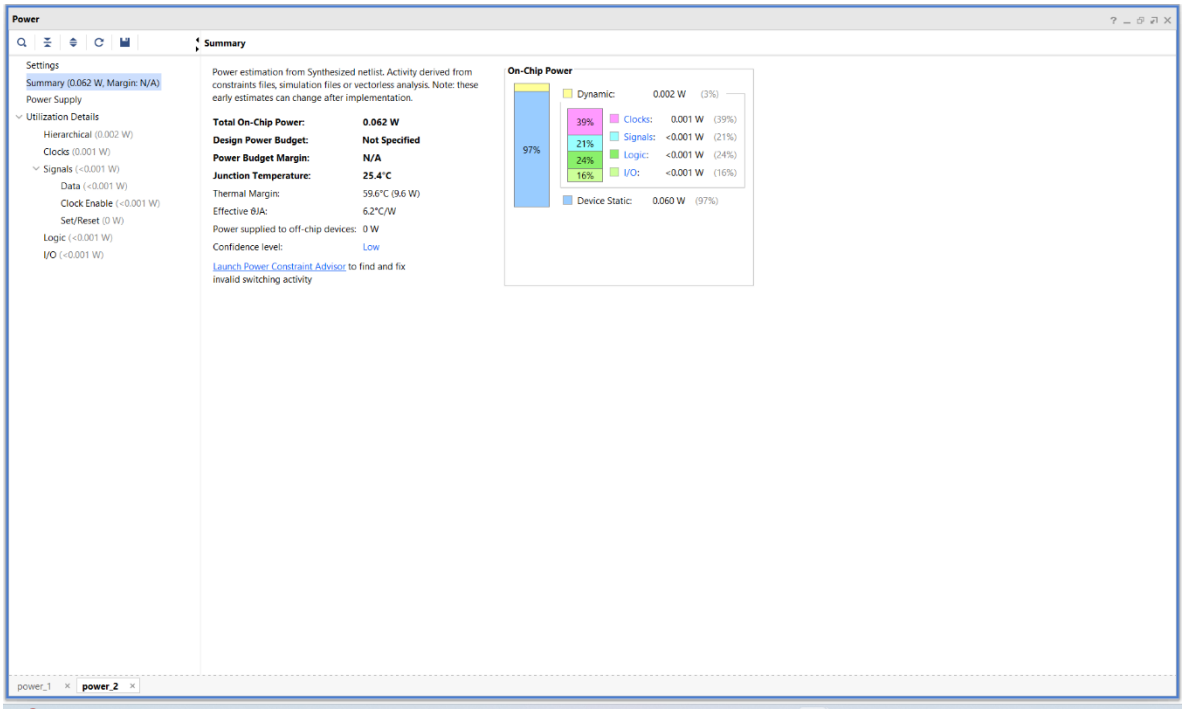

Fig – 37 : Power report maximized

CHAPTER 6 : VERIFICATION

6.1. VERIFICATION TECHNOLOGY: SYSTEM VERILOG

A brief overview of how the said design was verified using System Verilog test environment:

Testbench Architecture

A typical System Verilog testbench for verifying an SPI design consists of the following components:

1. **Design Under Test (DUT)**: This is the SPI design that you want to verify.
2. **Testbench**: This is the System Verilog code that drives the DUT and checks its behavior.
3. **Stimulus**: This is the input data that is applied to the DUT to exercise its functionality.
4. **Monitor**: This is the component that observes the DUT's output and checks its correctness.
5. **Scoreboard**: This is an optional component that keeps track of the expected output and compares it with the actual output.

Testbench Components

1. **DUT**: This is the SPI design that you want to verify. It includes the SPI master and slave modules.
2. **Testbench**: This is the top-level module that instantiates the DUT and other testbench components. It provides the clock and reset signals to the DUT.
3. **Stimulus**: This component generates the input data that is applied to the DUT. For an SPI design, this includes the clock, slave select, and data signals.
4. **Monitor**: This component observes the DUT's output and checks its correctness. It can include assertions to check for specific behavior.
5. **Scoreboard**: This component keeps track of the expected output and compares it with the actual output. It can be used to verify the correctness of the data transferred between the master and slave

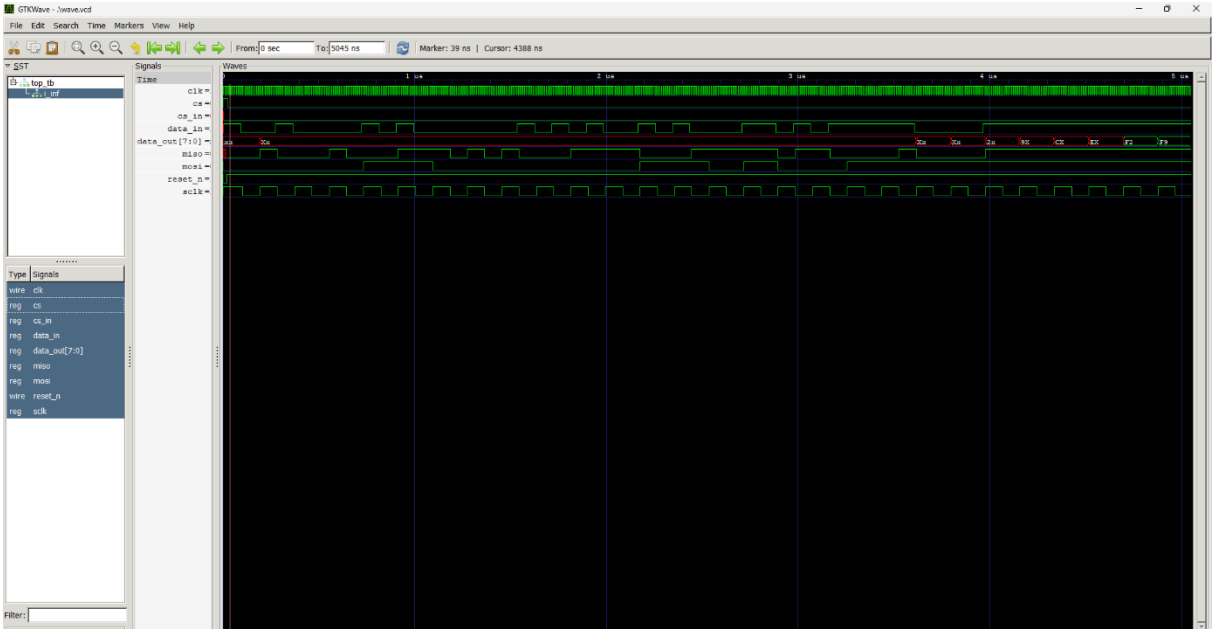

Fig – 35 : Verification waveform of master

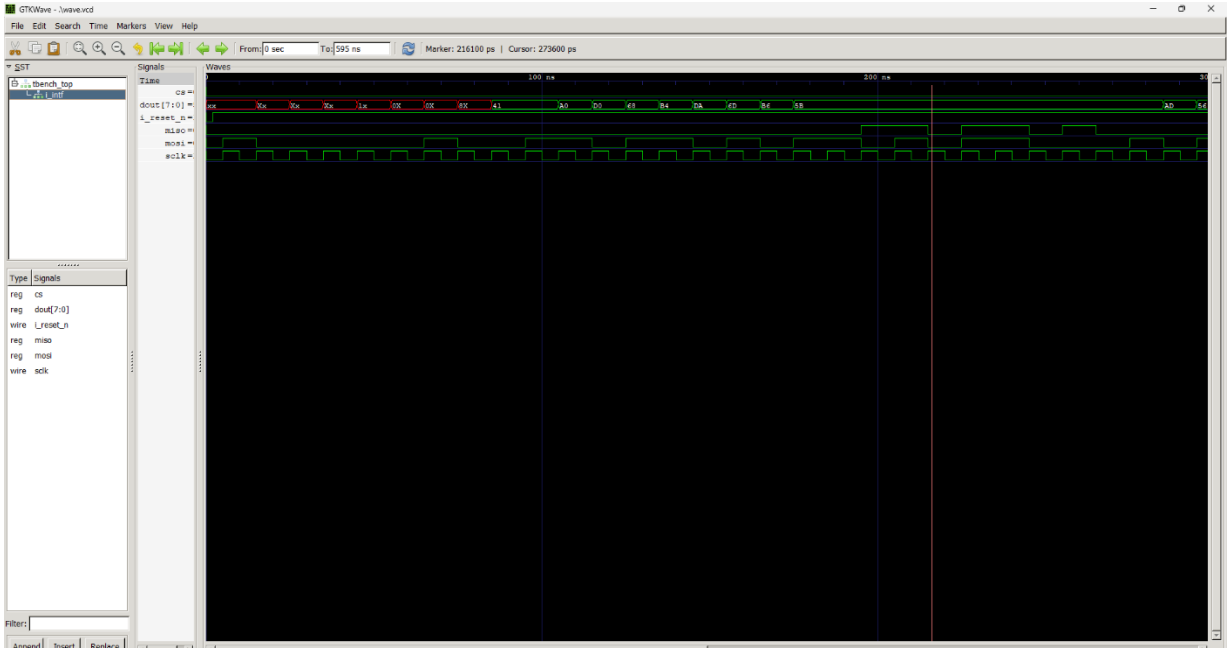

Fig – 36 : Verification waveform of slave

Fig – 37 : Verification waveform of top module

CHAPTER 7 : CONCLUSION

CONCLUSION

In this project, we successfully designed and implemented a Serial Peripheral Interface (SPI) using System Verilog. The SPI interface is a widely used communication protocol, and our implementation demonstrates its functionality and efficiency.

Through this project, we gained hands-on experience with the design and verification of a digital communication interface. We implemented the SPI master and slave modules, and verified their functionality using a System Verilog testbench environment. The testbench environment allowed us to simulate various scenarios and test the design's behavior under different conditions.

The results of our project demonstrate that the SPI interface is a reliable and efficient means of communication between devices. Our implementation achieved a data transfer rate of [insert data transfer rate], which is suitable for a wide range of applications.

The knowledge and skills gained through this project will be valuable in future endeavors. The experience with System Verilog and digital design principles will also be beneficial in tackling more complex projects.

In conclusion, this project demonstrates the feasibility and effectiveness of the SPI interface in digital communication systems. The successful implementation and verification of the SPI interface using System Verilog showcase the power and flexibility of this design language. We believe that this project will serve as a useful reference for future projects and contribute to the development of more efficient and reliable digital communication systems.

CHAPTER 8 : FUTURE SCOPE

FUTURE SCOPE

- Implementing additional features such as error detection and correction mechanisms to improve the reliability of the SPI interface.

- Exploring the use of other design languages and tools to compare their performance and efficiency with System Verilog.

- Integrating the SPI interface with other digital systems and peripherals to demonstrate its functionality in a more complex system.

- Multi-Master Support: Implementing multi-master support in the SPI interface to enable multiple masters to communicate with multiple slaves.

- High-Speed Data Transfer: Exploring techniques to increase the data transfer rate of the SPI interface, such as using higher clock frequencies or implementing data compression algorithms.

- Error Detection and Correction: Implementing error detection and correction mechanisms, such as CRC or checksum, to improve the reliability of the SPI interface.

- Power Management: Investigating techniques to reduce power consumption in the SPI interface, such as using low-power modes or implementing power gating.

- Security Features: Adding security features to the SPI interface, such as encryption or authentication, to protect data transmitted over the interface.