AUGUST 17, 2023

FULLY FUNCTIONAL SPI TRANSMITTER AND RECEIVER

AREEBA MEHBOOB
BCUBE PVT LIMITED

Verilog Task: Fully Functional SPI Transmitter and Receiver

Design Engineer: Areeba Mehboob

Technical Manager: Mr. Usama

Supervisor: Mr. Asad

Table of Contents:

List of Figures:

Problem Statement:

Verilog Task: Fully Functional SPI Transmitter and Receiver

This Task is divided into further sub tasks, which are listed below.

1. Study the SPI Transmitter and Receiver Working with signal waveforms.
2. Clock generator Module (Input 100MHz to 10MHz output).
3. Parallel to serial Shift Register (8 / 16 / 32 bits) using FSM.
4. Combine Sub-Task 2 and 3 with Control logic to initiate Process and raise a Done flag after completion of process.
5. Serial to parallel Shift Register (8 / 16 / 32 bits) using FSM.
6. Combine Sub-Task 2 and 5 with Control logic to initiate Process and raise a Done flag after completion of process.
7. Combine the tested module in task 2 & 3 and 4 for SPI Transmitter and Loop back data in tested module in task 2 & 5 and 6 for SPI Receiver. (Looped data should be matched).

1. Introduction:

This document serves as a comprehensive and approachable resource to grasp the concept of Serial Peripheral Interface (SPI) communication. The document covers three main aspects: clock division, individual data transmission, and data reception. By breaking down the process into these core components, it simplifies the overall understanding of SPI communication.

Firstly, the clock division aspect focuses on managing the timing of data exchange. It provides insights into how to generate different clock frequencies for optimal synchronization between devices. This is crucial for maintaining consistent communication and ensuring that data is transferred accurately.

Secondly, the individual data transmission module demonstrates how to convert parallel data into serial data suitable for SPI communication. It highlights the importance of mode selection, where a signal determines whether the system is ready to transmit data. The module's functionality clearly explained how data is serialized and transmitted effectively.

Lastly, the data reception module covers the reverse process—converting serial data back into parallel data. It emphasizes the role of flags indicating when data transmission is complete and when it's safe to read the received data.

This document provides a comprehensive breakdown of SPI communication, addressing clock division, data transmission, and data reception. By following the explained concepts and modules, users can gain a solid foundation in SPI communication.

2. SPI:

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol that enables high-speed, full-duplex communication between a master device (usually a microcontroller) and multiple slave devices (such as peripherals, sensors, or other microcontrollers). It offers a straightforward and cost-effective way to connect and control various hardware components. Unlike UART, which employs asynchronous communication over RX and TX lines, SPI employs synchronous communication with a clock signal. This ensures controlled data transfer and consistent data rates between the transmitting and receiving devices.

Key features of SPI:

1. Master-Slave Relationship: SPI operates in a master-slave configuration. The master device (typically a microcontroller) controls the communication process, while the slave devices respond to the master's commands.

2. Single Master, Multiple Slaves: An SPI bus supports a single master but can connect to multiple slave devices. Each slave is selected using a Chip Select (CS) or Slave Select (SS) line.

3. Four Signal Lines: The SPI bus employs four signal lines: - Master Out/Slave In (MOSI): Data is transmitted from the master to the slave. - Master In/Slave Out (MISO): Data is transmitted from the slave to the master. - Serial Clock (SCLK): This clock signal synchronizes data transmission between devices. - Chip Select (CS)/Slave Select (SS): Used by the master to select the target slave device for communication.

4. Shift Registers and Data Latches: Both master and slave devices include shift registers and data latches. The master device initiates data transmission by shifting data into the MOSI line, while the slave responds by shifting data into the MISO line. Data is then latched for further processing.

5. Loop Connection: The shift registers of both master and slave devices are interconnected, forming a loop. This loop ensures that data can be continuously transmitted between the devices.

6. Data Sizes: The typical size of the shift registers is 8 bits, but larger sizes, such as 16 bits, are also common. Overall, SPI provides a versatile and efficient means of connecting various hardware components, including memory modules, sensors, displays, and more. Its synchronous nature, precise clocking, and support for multiple slaves make it suitable for applications requiring high-speed data exchange and precise control over communication.

The following image shows the minimal system requirements for both the devices.

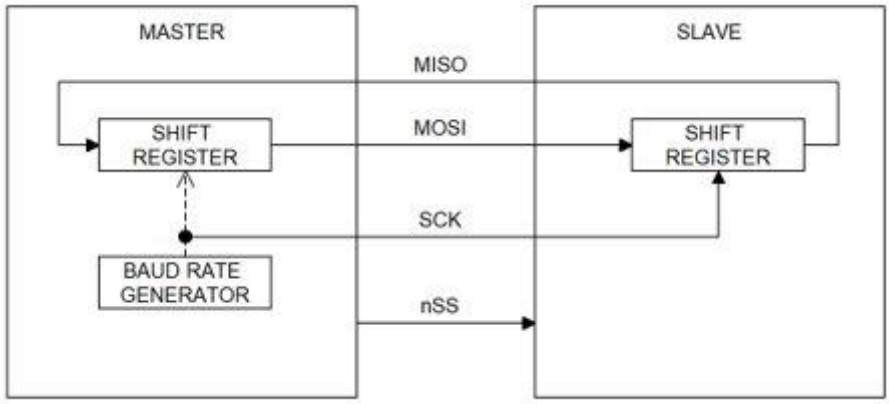


Figure 2.1 SPI signals 1

From the image, the Master device consists of a Shift Register, a data latch, and a clock generator. The slave consists of similar hardware: a shift registers and a data latch. Both the shift registers are connected to form a loop. Usually, the size of the register is 8 – bits but higher size registers of 16 – bits are also common.

During the positive edge of the clock signal, both the devices (master and slave) read input bit into LSB of the register. During the negative cycle of the clock signal, both the master and slave place a bit on its corresponding output from the MSB of the shift register.

Hence, for each clock cycle, a bit of data is transferred in each direction i.e., from master to slave and slave to master. So, for a byte of data to be transmitted from each device, it will take 8 clock cycles.

SPI modes of operation:

In SPI, the main can select the clock polarity and clock phase. The CPOL bit sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitioning to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission. The CPHA bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data. The main must select the clock polarity and clock phase, as per the requirement of the sub node. Depending on the CPOL and CPHA bit selection, four SPI modes are available.

Mode 0:

In this mode, clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase in this mode is 0, which indicates that the data is sampled on the rising edge and the data is shifted on the falling edge of the clock signal.

Mode 1

In this mode, clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase in this mode is 1, which indicates that the data is sampled on the falling edge

(shown by the orange dotted line) and the data is shifted on the rising edge (shown by the dotted blue line) of the clock signal.

Mode 2:

In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 0, which indicates that the data is sampled on the rising edge and the data is shifted on the falling edge of the clock signal.

Mode 3:

In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 1, which indicates that the data is sampled on the falling edge and the data is shifted on the rising edge of the clock signal.

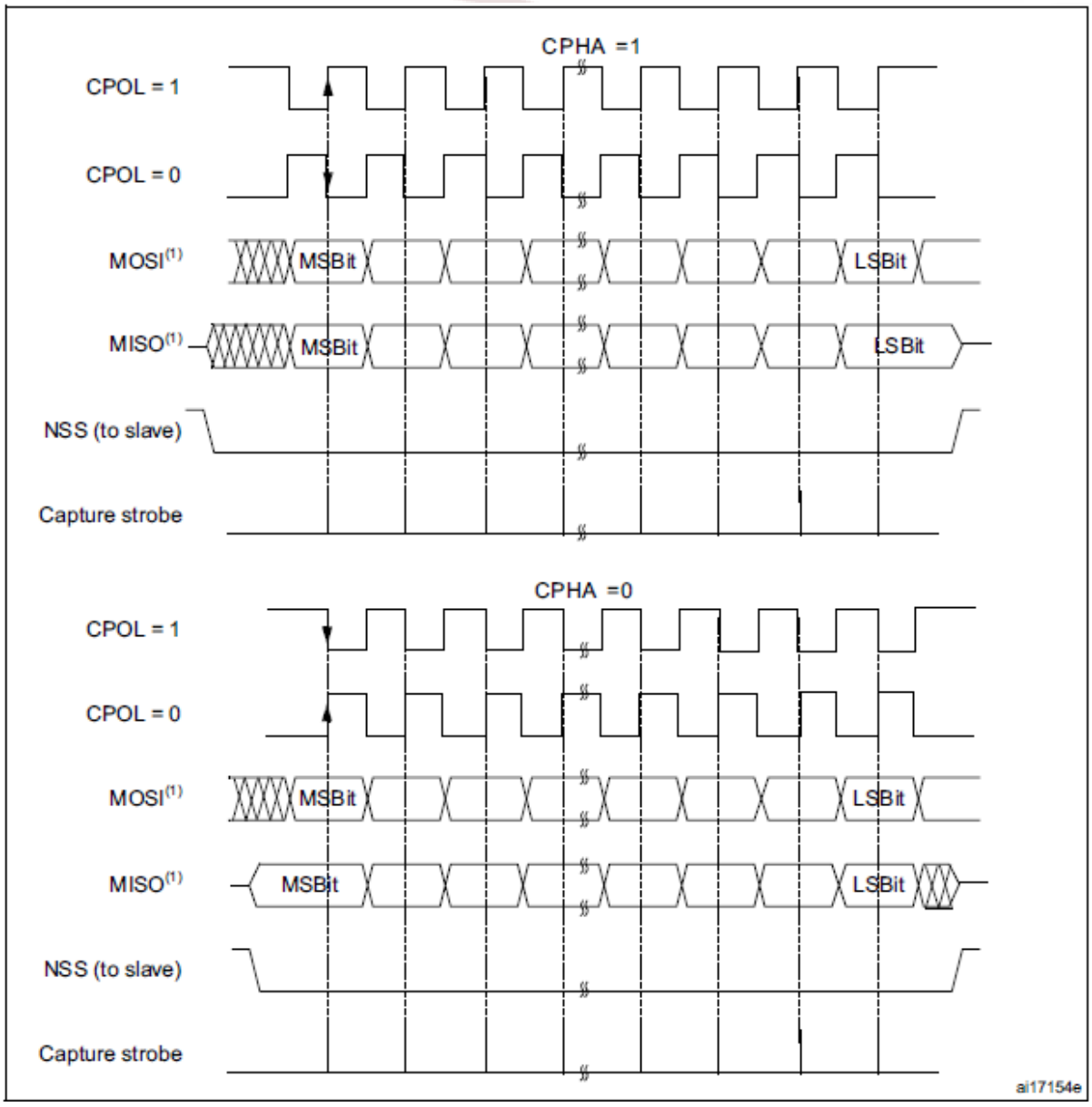


Figure 2.2 SPI Modes 1

Advantages of SPI

1. The main advantage of the SPI is to transfer the data without any interruption.
2. It is simple hardware.
3. It provides full-duplex communication.
4. There is no need for a unique address of the slave in this protocol.
5. This protocol does not require precise oscillation of slave devices because it uses the master's clock.
6. In this, software implementation is very simple.
7. It provides high transfer speeds.
8. Signals are unidirectional.
9. It has separate lines of MISO and MOSI, so the data can be sent and received at the same time.

Disadvantages of SPI

1. Usually, it supports only one master.
2. It does not check the error like the UART.
3. It uses more pins than the other protocol.
4. It can be used only from a short distance.
5. It does not give any acknowledgment that the data is received or not.

Applications of SPI

1. Memory: SD Card, MMC, EEPROM, and Flash.
2. Sensors: Temperature and Pressure.
3. Control Devices: ADC, DAC, digital POTS, and Audio Codec.
4. Others: Camera Lens Mount, Touchscreen, LCD, RTC, video game controller, e

3. Workflow & Block Diagram:

Workflow:

Below given diagram is workflow of my project:

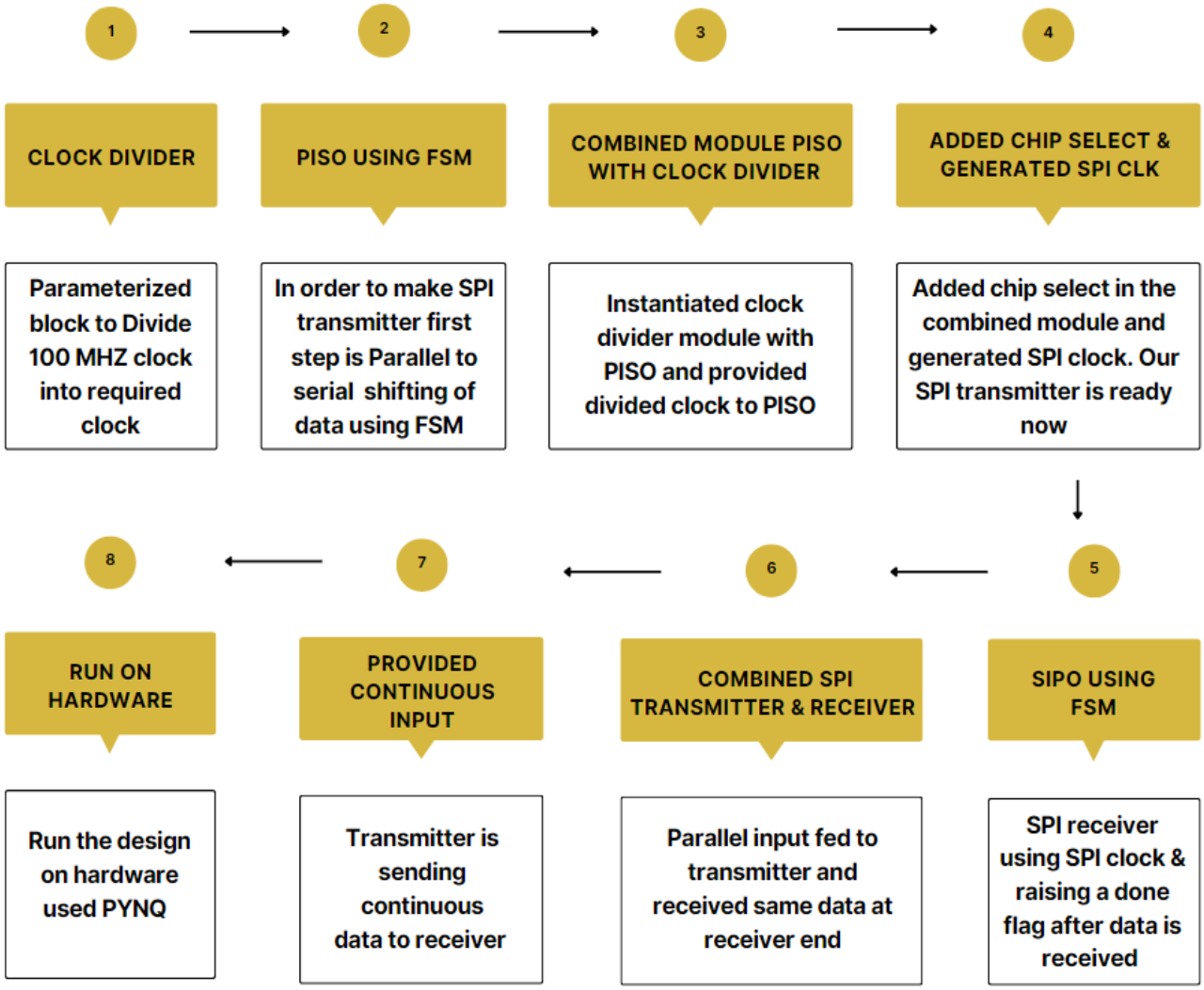


Figure 3.1 Workflow 1

BLOCK DIAGRAM:

Below I have attached block diagram of my project:

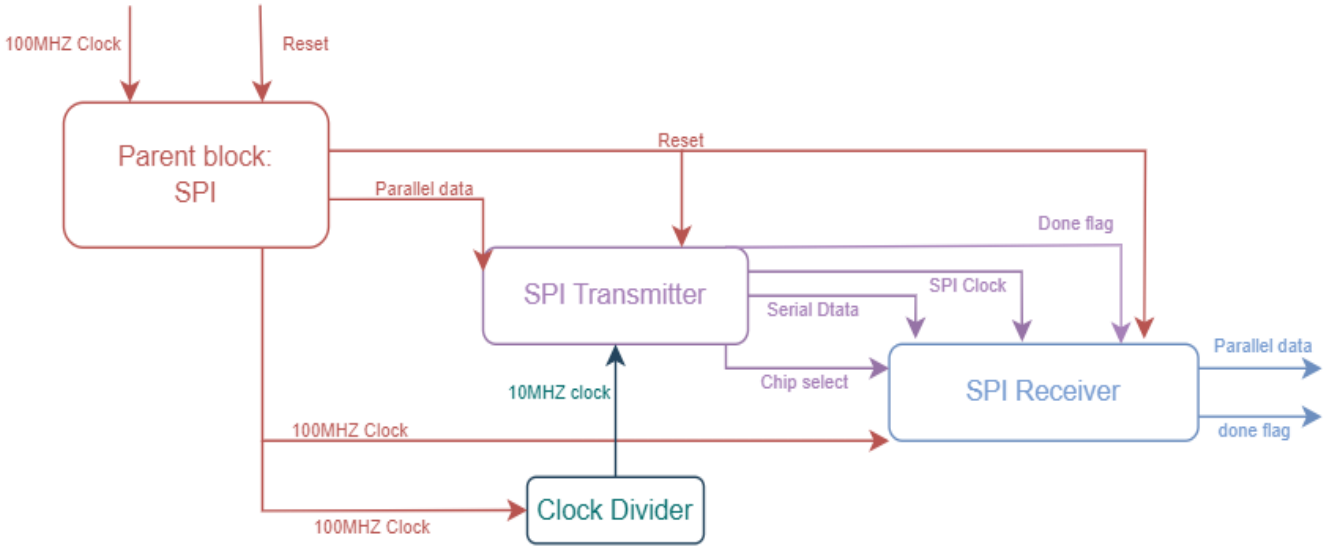


Figure 3.2 Block Diagram 1

4. CLOCK DIVIDER

A clock divider is a digital circuit that takes an input clock signal and produces an output clock signal with a lower frequency. It achieves this by dividing the frequency of the input clock signal by a fixed integer value. Clock dividers are commonly used in digital systems to generate slower clock signals that synchronize with higher-frequency clock domains, enable precise timing control, and facilitate interactions between different components operating at varying speeds.

Design module:

In this specific scenario, where we have an input signal of 100 MHz and we want to generate an output signal of 10 MHz, a clock divider can be employed to create an output pulse for every 10 pulses of the input clock. This ensures that the output signal maintains the desired frequency relationship with the input while accommodating different clock domains and timing requirements within our digital system. The clock divider's function is essential for controlling timing-critical operations, enabling synchronization, and maintaining efficient data processing in various digital applications.

- This section of code defines a Verilog mo dule named clk_divider.
- It takes two ports: clk_in (input clock) and clk_out (output clock).
- The module also has a parameter clk_count with a default value of 4.
- An 8-bit wide register named counter is declared. It is used as a counter to track the number of clock cycles. By changing the value of clk_count we can change after how many cycles we want to toggle our clk_out.

For code reference see appendix A.1

- In this section the counter value is incremented by 1 on every positive edge of the clk_in signal.
- It checks if the value of counter has reached the specified clk_count.
- If true, it toggles the clk_out signal and resets the counter back to 0.
- This division of frequency effectively generates the desired clock frequency at clk_out.

For code reference see appendix A.2

Output of clock divider:

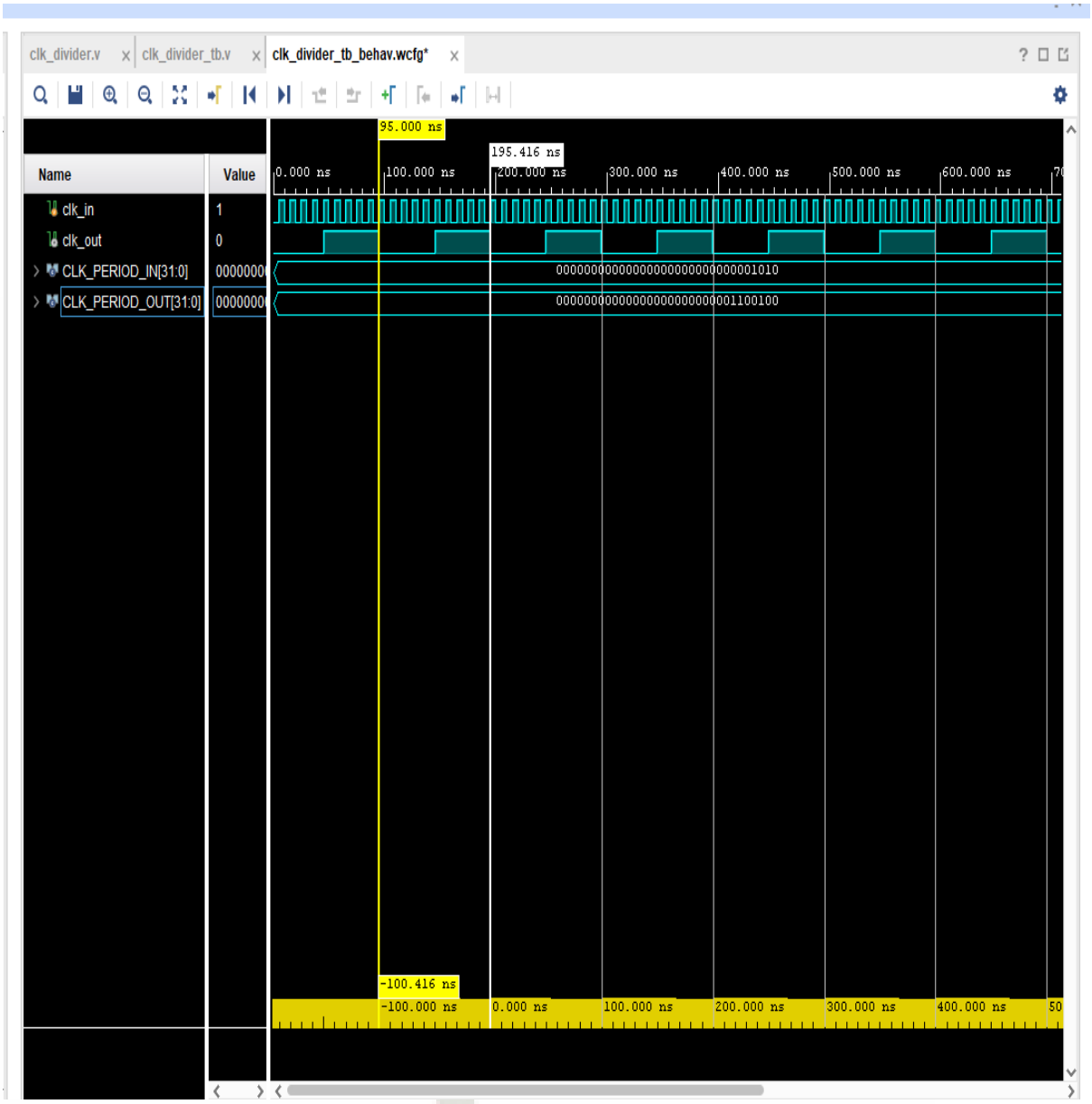


Figure 4.1 Clock Divider Output

5. SPI TRANSMITTER

Combine Clock Divider module with Parallel to Serial shift register and raise a Done flag after completion of process, use state machine.

Parallel to Serial Shift Register:

A parallel-to-serial shift register is a digital circuit that takes multiple parallel input data bits and converts them into a single serial output data stream. It's commonly used in digital systems for various purposes such as data transmission, data serialization, and reducing the number of output pins required.

The parallel-to-serial shift register consists of a chain of flip-flops, with each flip-flop representing one bit of the parallel input data. The data is loaded into the flip-flops in parallel and then shifted out serially. By shifting the data bit by bit, the parallel data is converted into a sequential bit stream.

The circuit is controlled by clock signals and sometimes control signals to initiate loading and shifting operations. Parallel-to-serial shift registers are often used in communication protocols like Serial Peripheral Interface (SPI) and Universal Asynchronous Receiver-Transmitter (UART), where data needs to be transmitted serially over a single wire or communication channel.

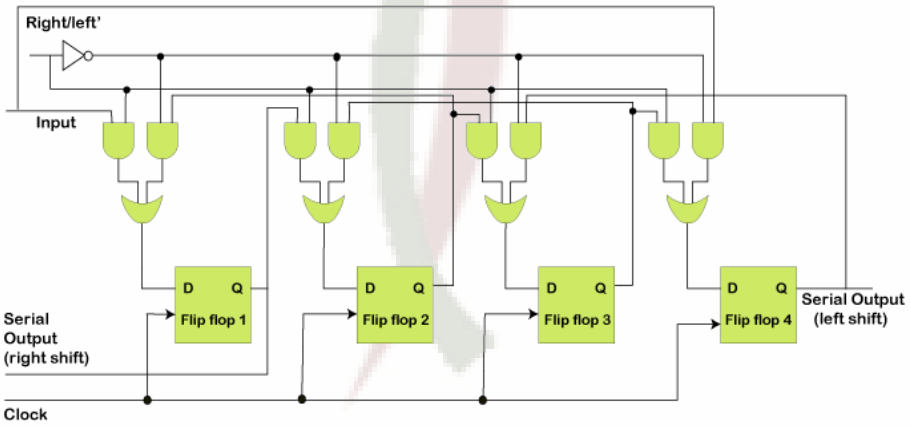


Figure 5.1 PISO

Design module:

This module can be used for scenarios where parallel data needs to be efficiently transmitted over a serial communication channel. The Clock Divider guarantees accurate timing synchronization, and the state machine arranges the entire process, ensuring that the conversion from parallel to serial is executed correctly. Once the transmission is finished, the shift_done flag provides a clear indication of the completion, making it an essential component for reliable serial data communication.

In this section of code all the ports and registers are defined.

For above explained code reference see appendix B.1

In this section we are making instance of the clock divider module so that we can provide divided clock to our transmitter.

For above explained code reference see appendix B.2

- In this code section, a clock signal clk is generated based on the falling edge of the input clock clk_in.
- When the latch_t signal is high (1), the clock signal clk is set to low (0), effectively halting its operation.
- When latch_t is low, the clock signal clk takes on the value of clkout, which is the output clock of the clock divider.
- Transmitter is generating clock for receiver which it will provide along with data to receiver.

For above explained code reference see appendix B.3

State Machine for SPI Transmitter:

In the Control path of state machine our state will be idle (state 0) when we reset our transmitter otherwise our state will be next_state.

For above explained code reference see appendix B.4

This section of code is the next state control path of our state machine which is thoroughly explained in the figure below:

In this state machine we have 3 states:

1. **IDLE (state 0)**
2. **SHIFT_DATA (state 1)**
3. **SHIFT_DONE (state 2)**

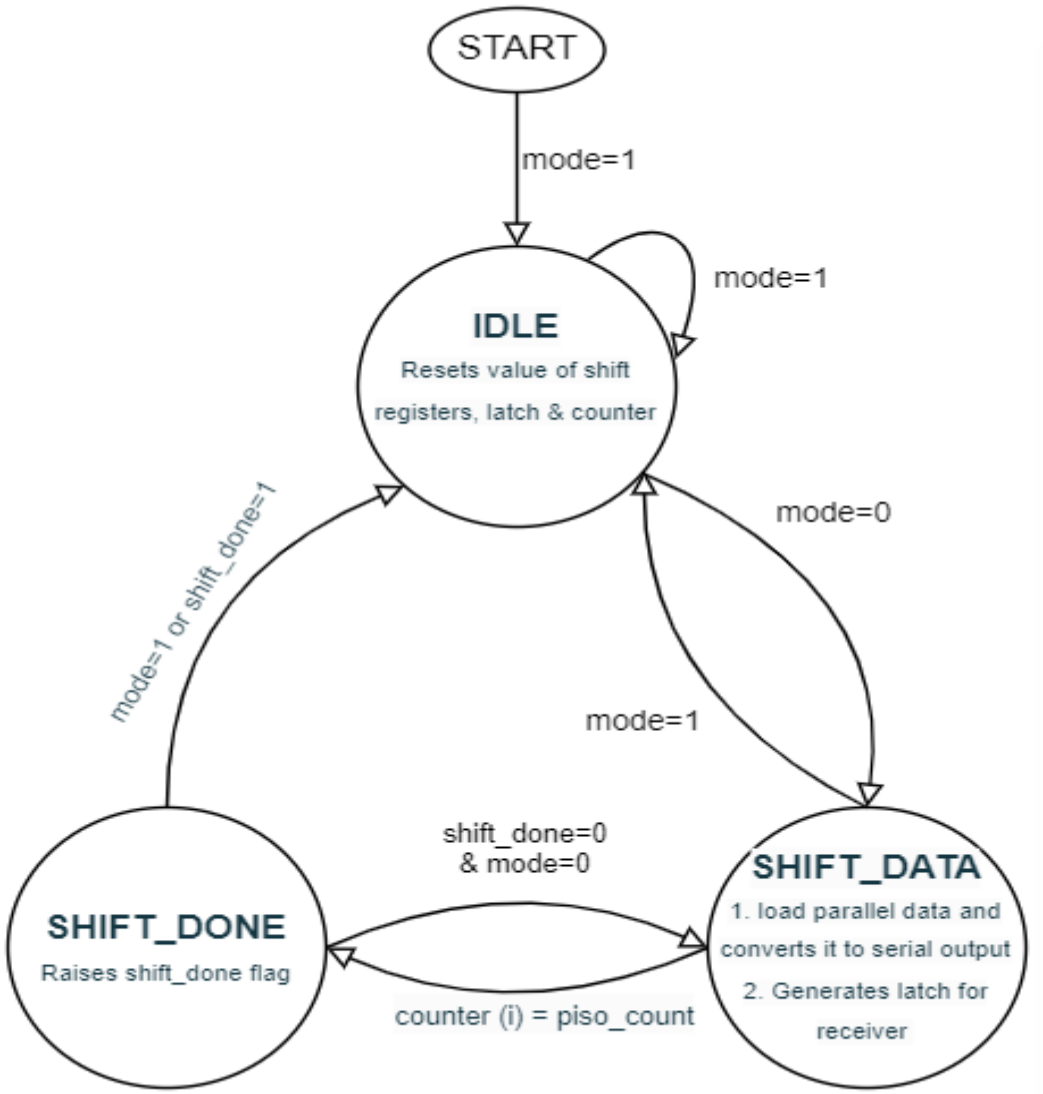


Figure 5.2 SPI Transmitter State Machine

- IDLE: In this state if MODE=1 it will remain in IDLE state. Otherwise, if MODE=0 it will go to state 1 which is SHIFT_DATA.
- SHIFT_DATA: In this state if our counter 'i' has reached the value of parameter 'piso_count' then it will go to state SHIFT_DONE otherwise, it will remain in state SHIFT_DATA.

- SHIFT_DONE: In this state if our done flag has raised (shift_done=1) or MODE=1 then it will go to state IDLE but if done flag has not raised (shift_done=0) then it will go back to state SHIFT_DATA.

For above explained code reference see appendix B.5

State Machine Data Path:

The data path in the state machine is responsible for processing and managing the data during different states of the module. Here's how the data path works:

IDLE State:

When the mode is 1 (reset condition), the module is in the IDLE state. The shift_done flag is cleared (set to 0). The parallel input pi is loaded into the parallel-to-serial shift register m. The counter i is initialized to 0. The shift_done flag is cleared (set to 0). The latch signal latch_t is set to 1 to ensure that no data is shifted during the IDLE state.

SHIFT_DATA State:

When the mode is 0, the module is in the SHIFT_DATA state. The parallel input pi is continuously loaded into the shift register ‘m’. The latch signal latch_t is set to 0, allowing data to be shifted out. If the counter ‘i’ is less than the specified piso_count, the shift register ‘m’ is shifted left, and its MSB is assigned to the serial out ‘so’ output. The counter ‘i’ is incremented.

SHIFT_DONE State:

In this state, the module waits until the counter ‘i’ reaches piso_count. Once the counter reaches the desired count, the shift_done flag is set to 1 indicating that the shift operation is complete. The counter ‘i’ is reset to 0, and the latch signal latch_t is set to 1 to prepare for the next transmission.

This data path efficiently manages the shift register's loading, shifting, and the control of the shift_done flag.

For above explained code reference see appendix B.6

OUTPUT of SPI Transmitter:

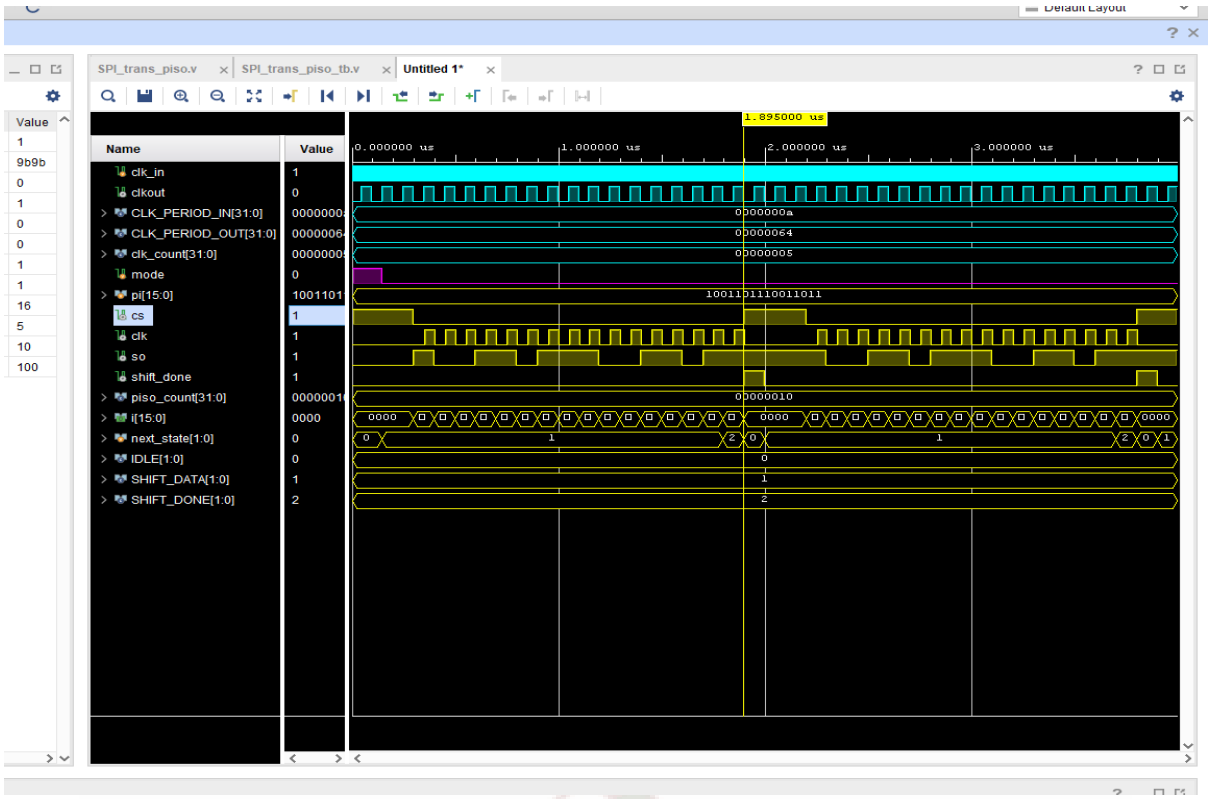


Figure 5.3 SPI Transmitter Output 1

6. SPI RECEIVER:

Serial to Parallel Shift Register:

A Serial-to-Parallel Shift Register is a digital circuit that takes a serial input data stream and converts it into parallel output data. It operates by shifting in each bit of the serial input sequentially, and once a complete parallel data word is accumulated, it is made available at the parallel output ports. This type of shift register is commonly used in various applications, such as data communication, data storage, and parallel processing.

In the context of communication protocols like SPI (Serial Peripheral Interface) and UART (Universal Asynchronous Receiver-Transmitter), a serial-to-parallel shift register is often used at the receiver end to convert the received serial data stream into parallel bytes or words that can be processed by the receiving system. This enables efficient and synchronized data transfer between devices operating at different clock speeds or with varying data widths.

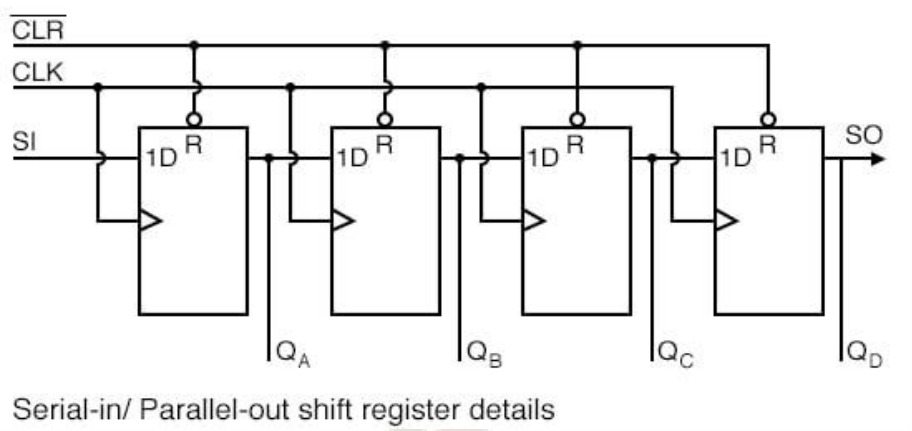


Figure 6.1 SIPO 1

Design Module of SPI Receiver:

Overall, the SIPO module operates as a serial-to-parallel shift register with control logic driven by a state machine. It converts serial input data into parallel data and raises a shift_done_r flag when the conversion is complete. The control path manages state transitions, while the data path handles data shifting and output storage.

In the above section all the ports and registers are defined for SPI Receiver module.

For above explained code reference see appendix C.1

State Machine:

Control path for state machine that our state will be idle_r (state 0) when we reset our transmitter otherwise our state will be next_state_r (state 1).

For above explained code reference see appendix C.2

In this section of code is the next state control path of our state machine which is thoroughly explain in the figure below:

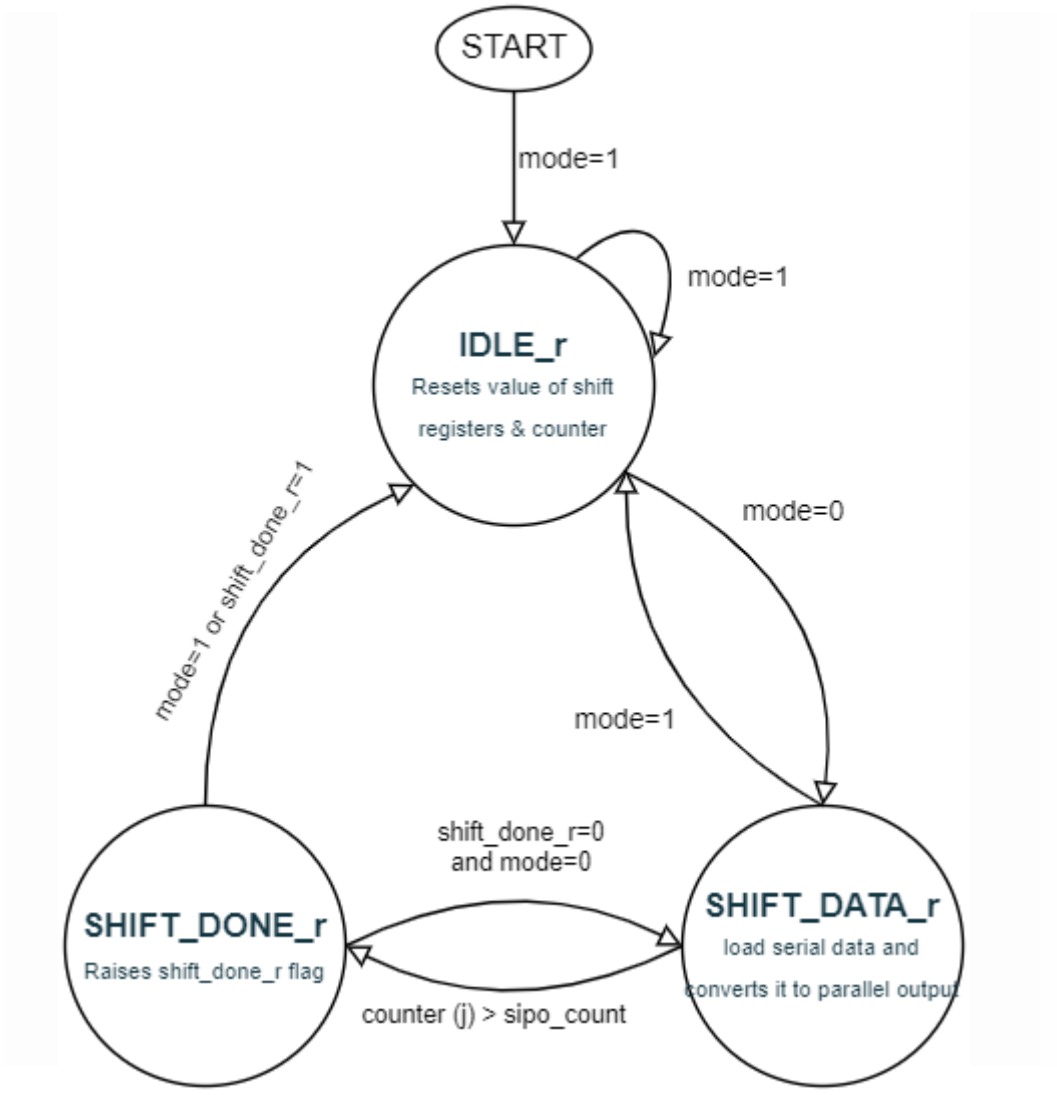


Figure 6.2 SPI Receiver State Machine 1

In this state machine we have 3 states:

1.IDLE_r (state 0)

2.SHIFT_DATA_r (state 1)

3.SHIFT_DONE_r (state 2)

IDLE_r: In this state if mode=1 it will remain in IDLE_r state. Otherwise, if mode=0 it will go to state 1 which is SHIFT_DATA_r.

SHIFT_DATA_r: In this state if our counter 'j' is greater than the value of parameter 'sipo_count' then it will go to state SHIFT_DONE_r otherwise, it will remain in state SHIFT_DATA_r.

SHIFT_DONE_r: In this state if our done flag has raised (shift_done_r=1) or mode=1 then it will go to state IDLE_r but if done flag has not raised (shift_done=0) and mode=0 then it will go back to state SHIFT_DATA_r.

For above explained code reference see appendix C.3

State Machine Data Path:

The data path in the state machine is responsible for processing and managing the data during different states of the module. Here's how the data path works:

1.IDLE State:

If mode is 1 (reset condition), the module is in the IDLE_r state. The shift_done_r flag is cleared (set to 0). Parallel output (po) sets to zero and counter (j) sets to zero.

2.SHIFT_DATA State:

When the mode is 0, the module is in the SHIFT_DATA_r state. If the counter 'j' is less than or equal to the specified sipo_count serial input pi is continuously loaded into the LSB of shift register p_o and shift register is shifted left. The counter 'j' is incremented. If the counter is greater than sipo_count, the shift_done_r flag is set to 1, indicating that all data has been received and stored in the form of parallel data.

3.SHIFT_DONE State:

In this state, the module waits until the counter 'j' is greater than sipo_count. Once the counter reaches the desired count, the shift_done_r flag is set to 1 to indicate that the shift operation is complete. The counter 'j' is reset to 0 to receive the next coming data.

This data path efficiently manages the shift register's loading, shifting, and the control of the shift_done_r flag.

For above explained code reference see appendix C.4

In the above given code parallel data from p_o register will be assigned to output 'po' only when a done flag is raised, and this done flag is raised after all the serial data has been successfully stored in p_o register

For above explained code reference see appendix C.5

Output of SPI Receiver:

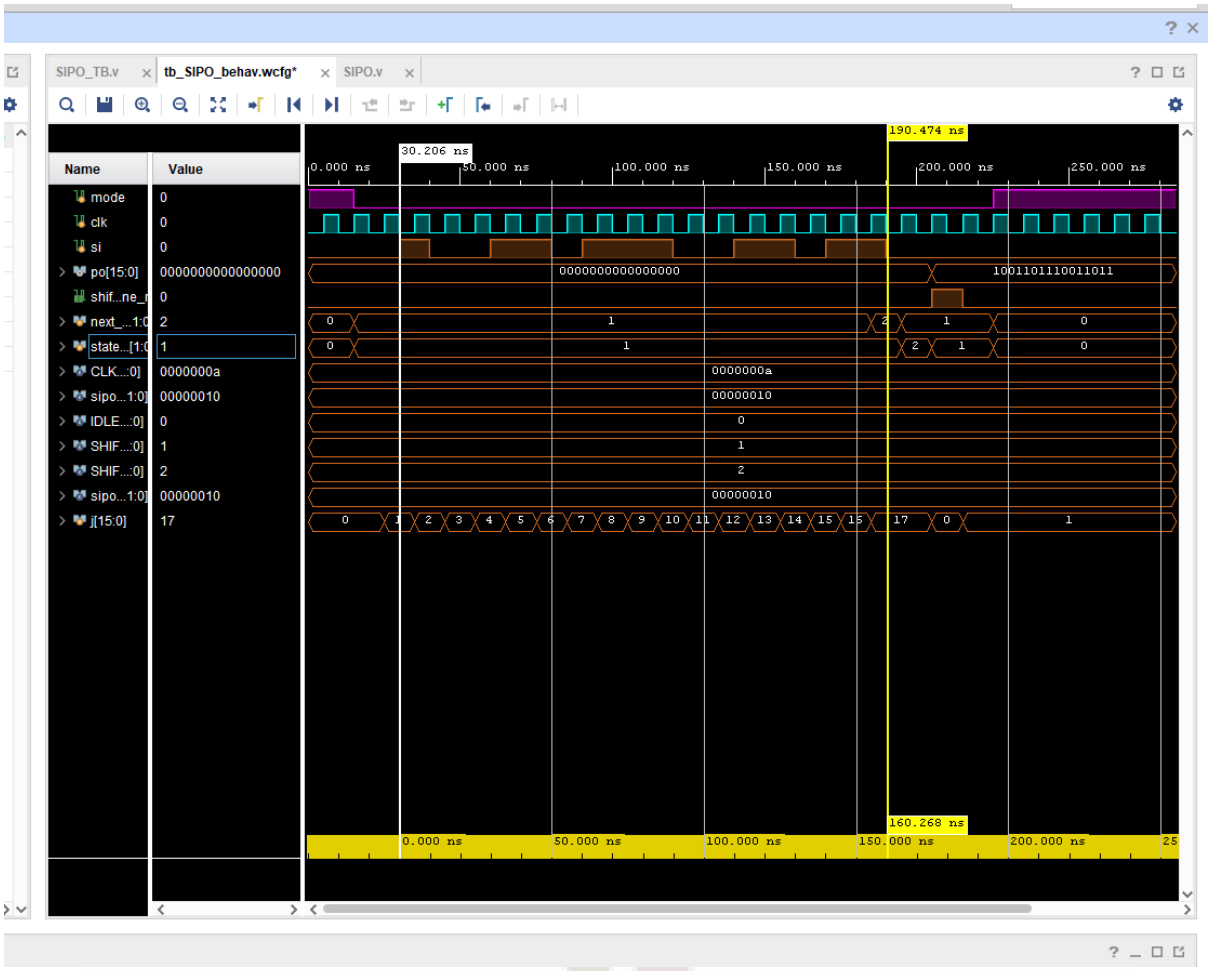


Figure 6.3: SPI Receiver Output 1

7. SPI TRANSMISSION

For SPI transmission on hardware instead of test bench we will use Integrated Logic Analyzer.

ILA (Integrated Logic Analyzer) is a debugging and analysis tool provided by Xilinx that allows you to monitor and capture signals within your FPGA design while it's running on the hardware. It helps you analyze the behavior of your design by capturing signal values, waveforms, and data at specific points in time.

Parent Design module:

This module combines SPI transmitter and receiver, with additional logic for managing the parallel input to the transmitter. The clkout signal generated by the transmitter is transmitted to the receiver along with latch ‘cs’ and serial data ‘s_d’.

- All the ports and registers are defined in this section.
- It takes various inputs including mode (control signal), clk_in (input clock), and others.
- It has a set of outputs including pi (parallel input to transmitter), s_d (serial data line), cs (chip select signal), po (parallel output from receiver), shift_done_r (receiver shift done flag), next_state_r (receiver's next state), and state_r (receiver's current state).
- It also has two registers for counter and to store that counter’s value in parallel input ‘pi’.

For above explained code reference see appendix E.1

SPI transmitter and SPI receiver modules are instantiated here.

For above explained code reference see appendix E.2

In this always block if value of counter ‘i’ of transmitter has reached piso_count and mode=0 it means transmission has been done and transmitter is ready for another data to be transmitted only then ‘pi’ has been assigned value of counter ‘n’ until ‘n’ is less than piso_count.

For above explained code reference see appendix E.3

Daughter Modules:

Following are the daughter modules for SPI transmission.

Transmitter module:

For explanation refer to Section 5 ‘SPI Transmitter’.

For code reference see appendix B

Receiver module:

For explanation refer to Section 6 ‘SPI Receiver’.

For code reference see appendix C

Clock Divider module:

For explanation refer to Section 4 ‘Clock Divider’.

For code reference see appendix A

Vivado simulation:

Reset=0:

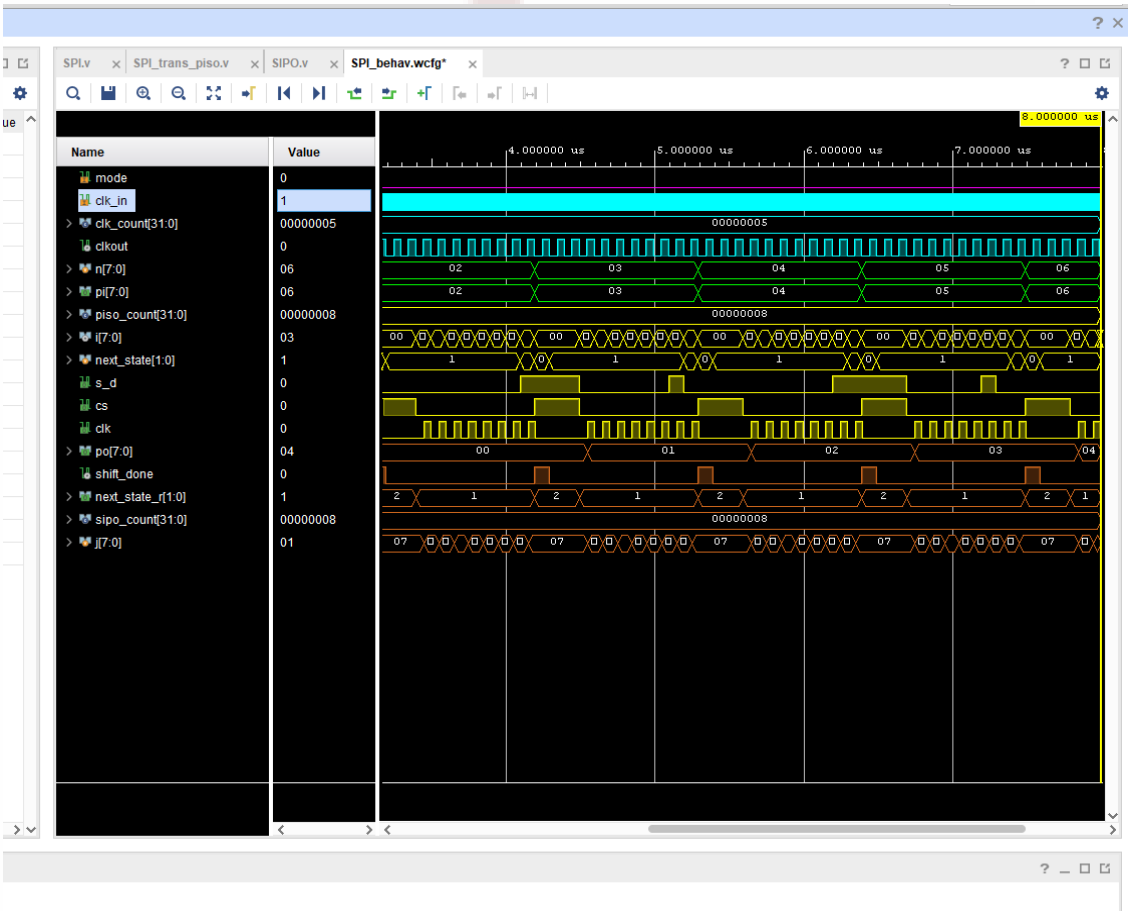


Figure 9.1 SPI Transmission vivado, simulation with continuous data when reset is 0.

Reset=1:

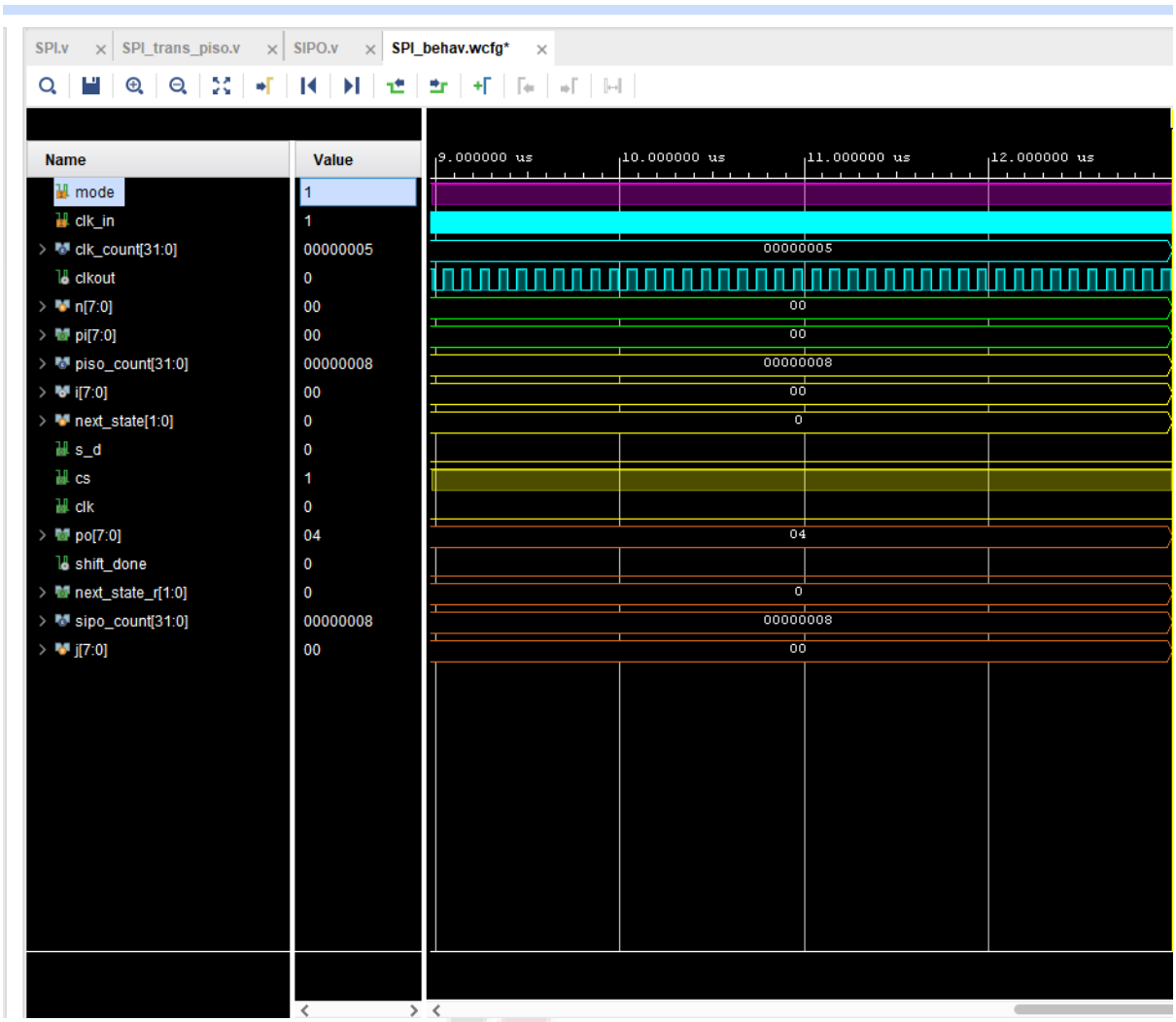
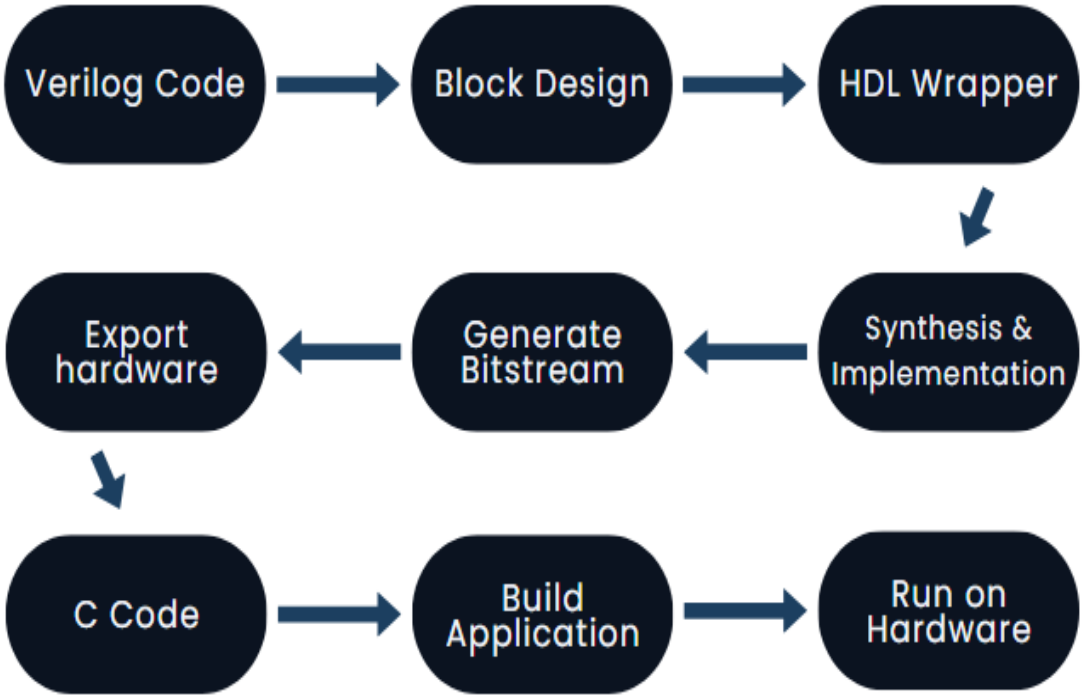


Figure 9.2 SPI Transmission, vivado simulation with continuous data when reset is 1.

8. DESIGN FLOW:

Below Provided figure is the design flow before running on hardware using clock and reset provided by processing system.



PYNQ development board

In this project we are using PYNQ development board for processing system and programmable logic communication.

PYNQ boards, such as the PYNQ-Z1 or PYNQ-Z2, are designed for use with Xilinx Zynq SoCs and provide an accessible platform for FPGA development. These boards are equipped with various peripherals and interfaces, making them suitable for a wide range of embedded system development and FPGA prototyping tasks. Developers can use PYNQ boards in conjunction with the PYNQ framework to leverage the power of the Zynq SoC while using Python and Jupiter notebooks for programming and development.

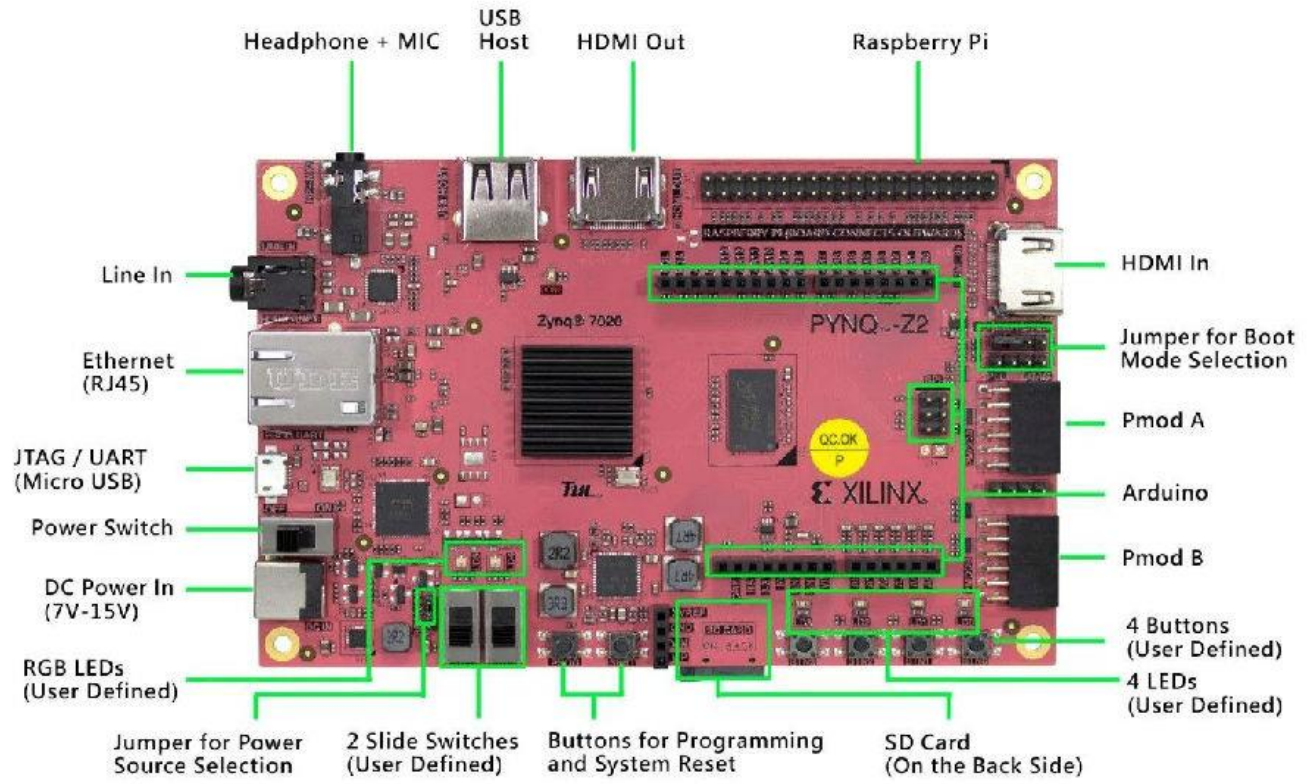


Figure 1.7 PYNQ Z2

Key features of PYNQ:

SOC

- Zynq-7000 SoC XC7Z020-1CLG400C

I/O interfaces

- USB-JTAG Programming circuitry
- USB OTG 2.0

- USB-UART bridge
- One 10/100/1G Ethernet
- HDMI Input
- HDMI Output
- I2S interface with 24bit DAC with 3.5mm TRRS jack
- Line-in with 3.5mm jack

Memory

- 512 Mbyte DDR3 with 16-bit bus @ 1050 Mbps
- 128 Mbit Quad-SPI Flash
- Micro SD card connector

Switches and LEDs

- 2 Slide switches
- 2 RGB LEDs
- 4 LEDs
- 4 Pushbuttons

Clocks

- One 125 MHz for PL
- One 50 MHz for PS

Expansion ports

- 2 Pmod ports
    - 16 Total FPGA I/O (8 shared pins with Raspberry Pi connector)
- 1 Arduino Shield connector
    - 24 Total FPGA I/O
    - 6 Single-ended 0-3.3V Analog inputs to XADC
- Raspberry Pi connector
    - 28 Total FPGA I/O (8 shared pins with Pmod A port)

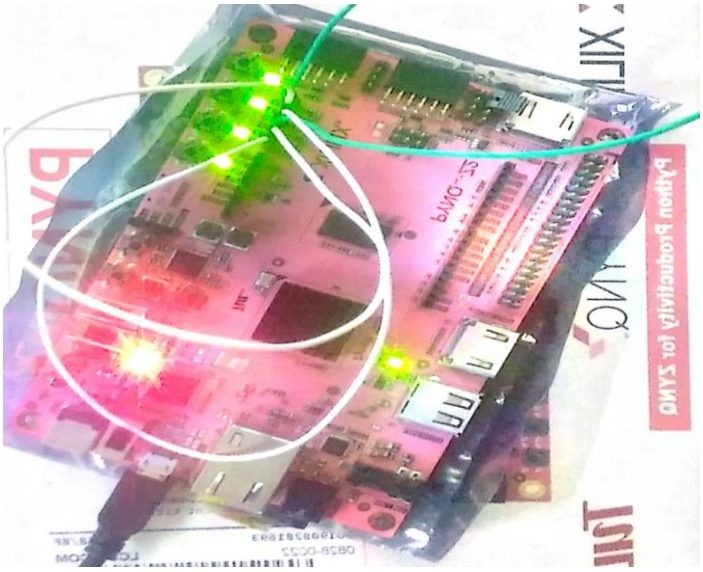
8. HARDWARE SETUP:

Our project setup includes a PYNQ device, a USB cable to connect hardware to PC, and 3 male-to-male jumper wires.

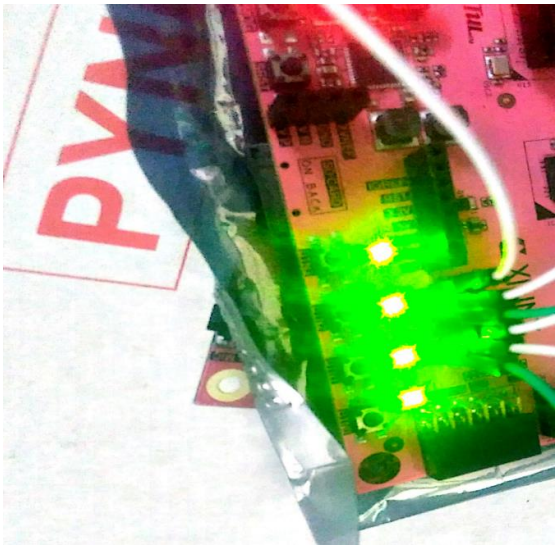
1 For providing clock to SPI receiver from SPI transmitter.

2 For providing Chip Select from SPI receiver to SPI transmitter.

3 For transmitting serial data from transmitter to receiver.



(a)



(b)

ILA Simulation:

Reset=0:

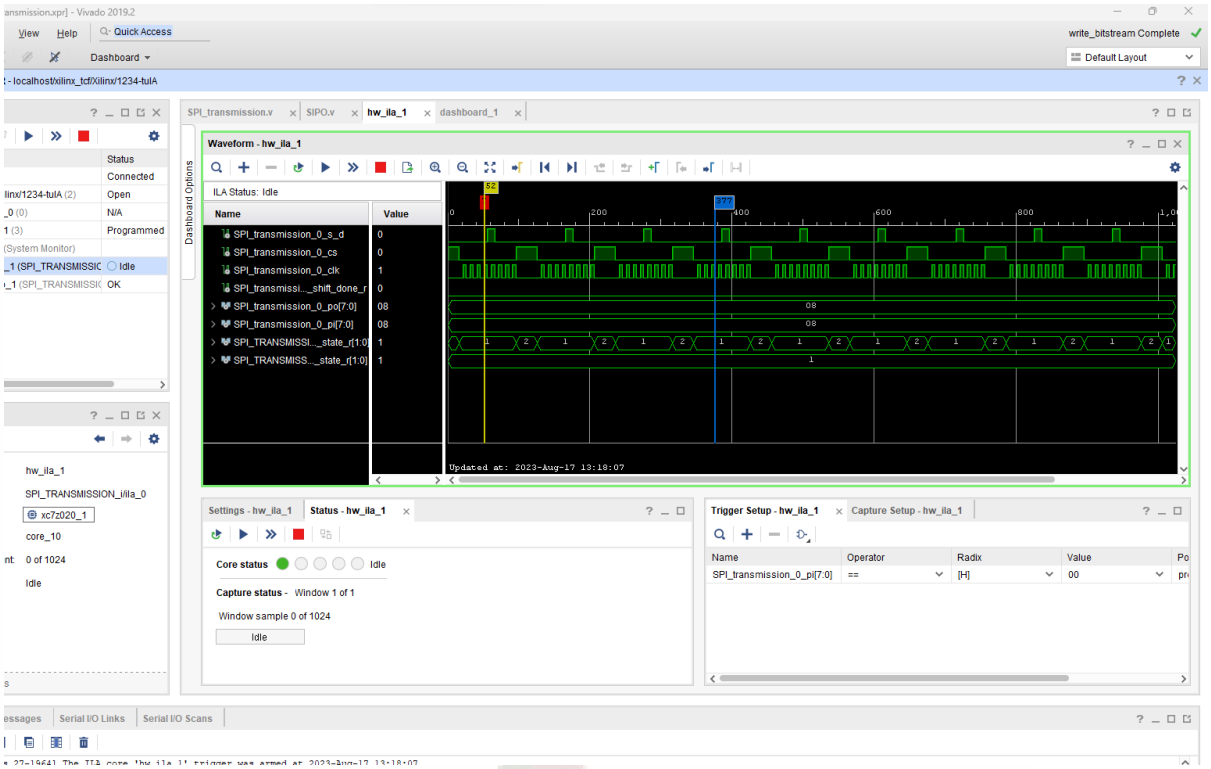


Figure 9.3 SPI Transmission, ILA simulation when reset is 0.

Reset=1:

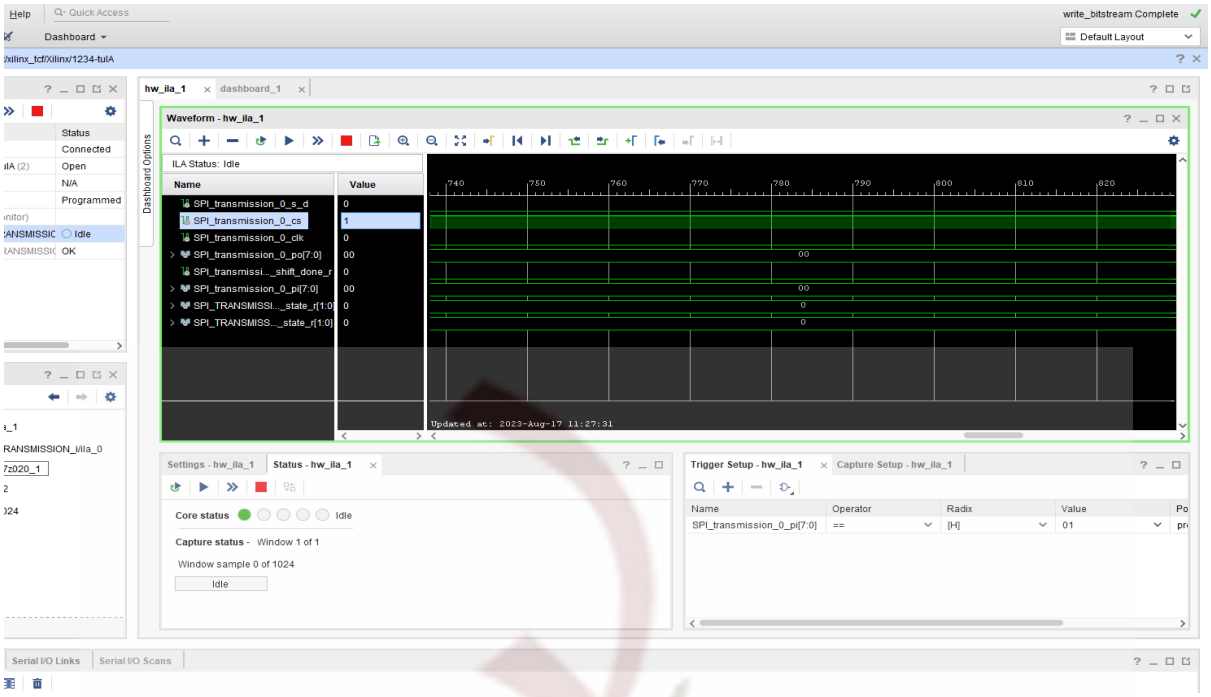


Figure 9.4 SPI Transmission, ILA simulation when reset is 1.

10.Resource Utilization report:

Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

| Tool Version : Vivado v.2019.2 (win64) Build 2708876 Wed Nov 6 21:40:23 MST 2019
| Date : Thu Aug 17 12:10:30 2023
| Host : Areeba running 64-bit major release (build 9200)
| Command : report_utilization -file SPI_TRANSMISSION_SPI_transmission_0_0_utilization_synth.rpt -pb SPI_TRANSMISSION_SPI_transmission_0_0_utilization_synth.pb
| Design : SPI_TRANSMISSION_SPI_transmission_0_0
| Device : 7z020clg400-1
| Design State : Synthesized

Utilization Design Information

Table of Contents

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*			53200	0.06
LUT as Logic			53200	0.06
LUT as Memory			17400	0.00
Slice Registers			106400	0.04
Register as Flip Flop			106400	0.04
Register as Latch			106400	<0.01
F7 Muxes			26600	0.00
F8 Muxes			13300	0.00

2. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	140	0.00
RAMB36/FIFO*	0	0	140	0.00
RAMB18	0	0	280	0.00

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	220	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	0	0	125	0.00
Bonded IPADs	0	0	2	0.00
Bonded IOPADs	0	0	130	0.00
PHY_CONTROL	0	0	4	0.00
PHASER_REF	0	0	4	0.00
OUT_FIFO	0	0	16	0.00
IN_FIFO	0	0	16	0.00
IDELAYCTRL	0	0	4	0.00
IBUFDS	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	200	0.00
ILOGIC	0	0	125	0.00
OLOGIC	0	0	125	0.00

5. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	0	0	32	0.00
BUFIO	0	0	16	0.00
MMCME2_ADV	0	0	4	0.00
PLLE2_ADV	0	0	4	0.00
BUFMRCE	0	0	8	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	16	0.00

6. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	40	Flop & Latch
LUT2	11	LUT
LUT6	9	LUT
LUT3	9	LUT
LUT5	7	LUT
LUT4	7	LUT
LDCE	4	Flop & Latch
LUT1	2	LUT

Conclusion:

In this document, we have explored the design and implementation of individual components for SPI communication, including a clock divider, SPI transmitter, and SPI receiver. The clock divider ensures precise clock synchronization, while the transmitter converts parallel data to serial format and generates control signals. The receiver module converts received serial data back to parallel and raises a "done" flag upon completion. Finally, we've combined these components into a comprehensive SPI transmission module. This integrated solution showcases how to achieve reliable and efficient SPI communication for digital systems, enhancing their flexibility and interaction capabilities.

Appendix:

Appendix A:

Appendix B:

Appendix C:

Appendix D:

References:

https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

https://www.elprocus.com/piso-shift-register/

https://techmasterplus.com/programs/verilog/verilog-shiftregister.php

https://www.allaboutcircuits.com/textbook/digital/chpt-12/serial-in-parallel-out-shift-register/

https://www.edaboard.com/threads/clock-divider-in-verilog.303639/

https://www.realdigital.org/doc/9c21eab4a0f85c50486858a87380d1f6

https://www.design-reuse.com/articles/52504/serial-peripheral-interface-spi.html

https://www.electronicshub.org/basics-serial-peripheral-interface-spi/

https://fastbitlab.com/spi-cpol-cpha-discussion/

https://www.javatpoint.com/spi-protocol