COURSE: VLSI design methodologies

Project Report

Done by:

V.Veerraju choudhary

21BEC1652

Serial peripheral interface (SPI)

Introduction

Serial Peripheral Interface (SPI) is a master – slave type protocol that provides a simple and low cost interface between a microcontroller and its peripherals ICs such as sensors, ADC's,DAC's, shift register, SRAM and others. In SPI protocol, there can be only one master but many slave devices.

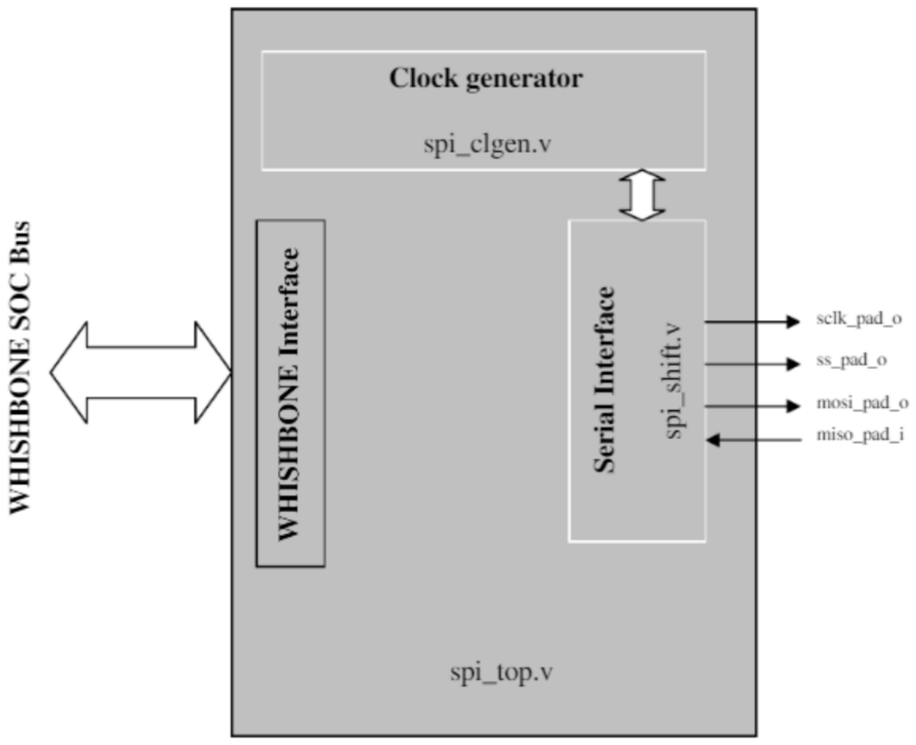
The SPI bus consists of 4 signals or pins. They are

- Master – Out / Slave – In (MOSI)
- Master – In / Slave – Out (MISO)
- Serial Clock (SCLK) and
- Slave Select (SS)

Data is exchanged simultaneously between devices in a serial manner (shifted out serially and shifted in serially). Serial clock line synchronizes shifting and sampling of information on two serial data lines. A slave select line allows individual selection of slave SPI devices

Architecture of SPI Master Core

The SPI Master core consists of three parts shown in the following figure:

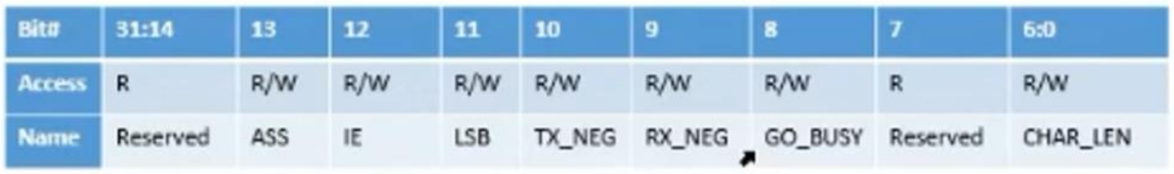


Features of SPI Master Core:

- Full duplex synchronous serial data transfer
- Variable length of transfer word up to 128 bits
- MSB or LSB first data transfer
- Rx and Tx on both rising or falling edge of serial clock independently
- 8 slave select lines
- Fully static synchronous design with one clock domain
- Technology independent Verilog
- Fully synthesizable

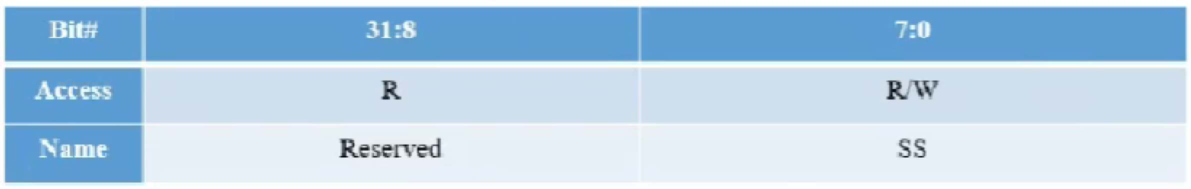
Module “spi.defines.v” is used to initialize all the variable values related to registers (CTRL, SS, DIVIDER and SPI Registers).

Control and Status Register bus is as follows:



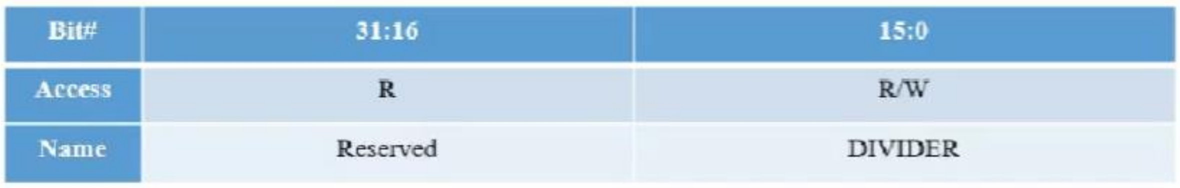
Bit#	31:14	13	12	11	10	9	8	7	6:0
Access	R	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Name	Reserved	ASS	IE	LSB	TX_NEG	RX_NEG	GO_BUSY	Reserved	CHAR_LEN

Slave Select Register bus is as follows:



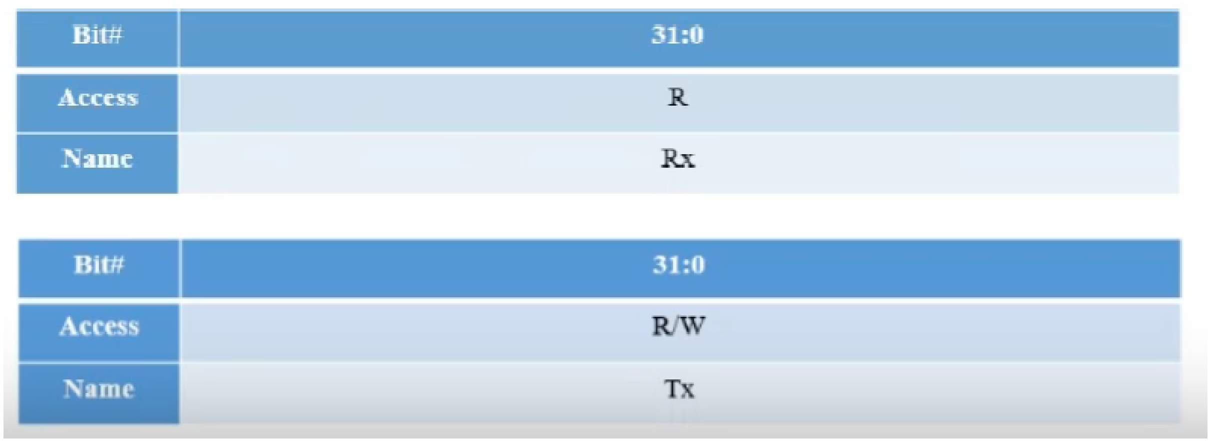
Bit#	31:8	7:0
Access	R	R/W
Name	Reserved	SS

Divider Register bus is as follows:



Bit#	31:16	15:0
Access	R	R/W
Name	Reserved	DIVIDER

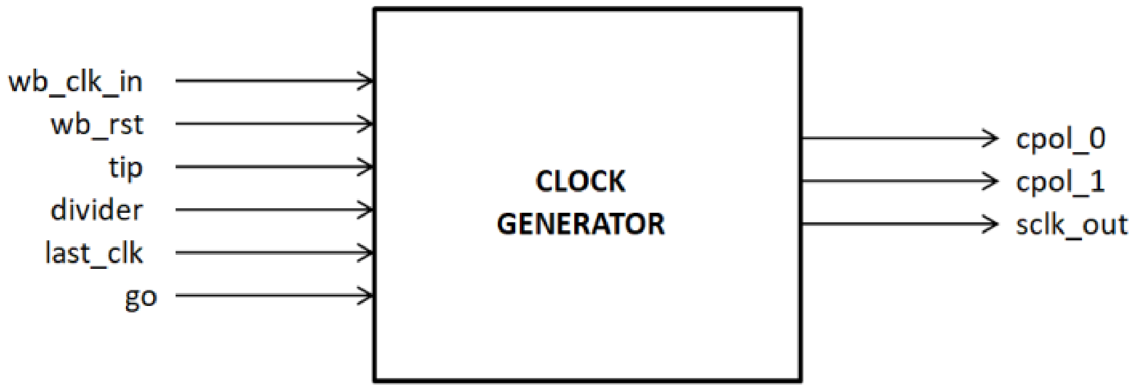
TxX and RxX Register bus are as follows:



Bit#	31:0
Access	R
Name	Rx

Bit#	31:0
Access	R/W
Name	Tx

Clock generator



Functionality:

In an SPI (Serial Peripheral Interface) project, a clock generator plays a pivotal role in the reliable transmission of data between master and slave devices. This component is responsible for generating the crucial clock signal, commonly known as SCLK (Serial Clock), which synchronizes the data exchange. Let's delve into the various functions and significance of a clock generator in an SPI project:

Module Declaration: Defines the module spi_clgen with input and output ports.

● Inputs:

- wb_clk_in: Input clock signal.
- wb_rst: Reset signal.
- go: Signal indicating the start of the transfer.
- tip: Signal indicating whether a transfer is in progress.
- last_clk: Signal indicating the last clock edge of the transfer.
- divider: Input value determining the clock divider for generating the SPI clock.

● Outputs:

- sclk_out: Output SPI clock signal.
- cpol_0: Output signal indicating the pulse marking the positive edge of the SPI clock.
- cpol_1: Output signal indicating the pulse marking the negative edge of the SPI clock.

The clock generator block generates two clock signals: rx_clk and tx_clk. These clock signals are used to control the receive and transmit operations of the SPI shift register. The generation of these clocks is based on the input signals rx_negedge, tx_negedge, last, sclk, cpol_0, and cpol_1

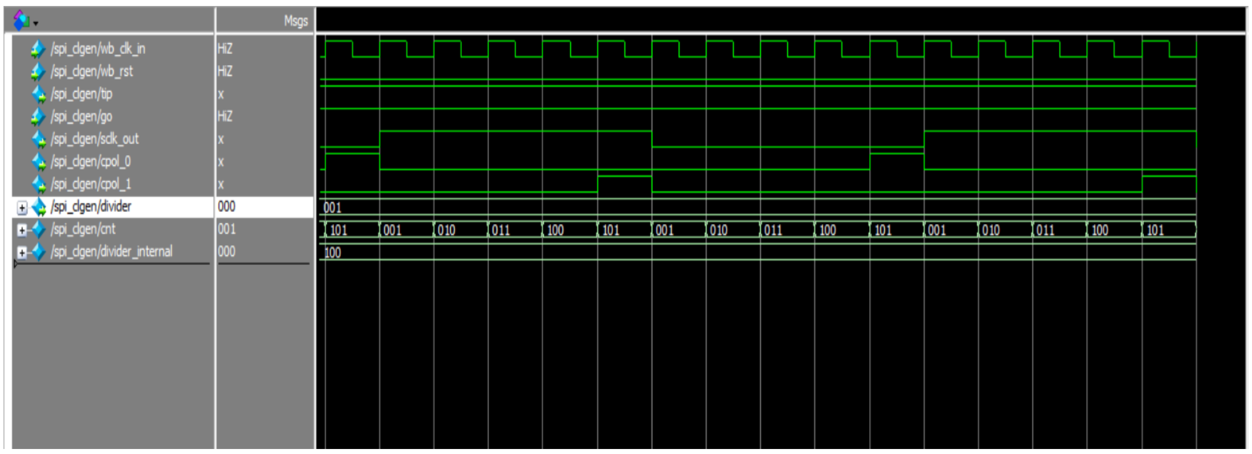
The value given to divider field is the frequency divider of the system clock wb_clk_i to generate the serial clock on the output sclk_pad_o. The desired frequency is obtained according to the following equation:

$$f_{sclk} = \frac{f_{wb\_clk}}{(DIVIDER+1)*2}$$

The rx_clk signal is generated by combining the rx_negedge input signal and the logical OR of last and sclk. It is used as the receive clock enable. The tx_clk signal is generated by combining the tx_negedge input signal and the logical AND of last and the selected clock polarity (cpol_0 or cpol_1). It is used as the transmit clock enable.

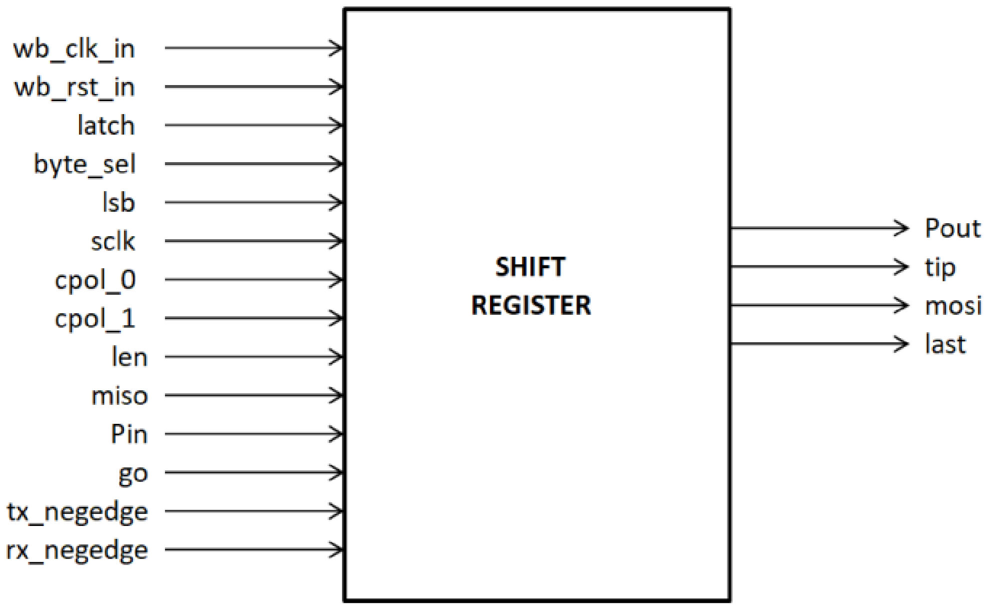
The clock generator block ensures that the clock signals are generated appropriately based on the specified clock polarities and the transfer status (last). This enables proper synchronization and timing control for the SPI shift register module during data transmission and reception.

output



```
                          Msgs
/spi_clgen/wb_clk_in      HiZ
/spi_clgen/wb_rst         HiZ
/spi_clgen/tip            x
/spi_clgen/go             HiZ
/spi_clgen/sclk_out       x
/spi_clgen/cpol_0         x
/spi_clgen/cpol_1         x
/spi_clgen/divider        000   001
/spi_clgen/cnt            001   101 001 010 011 100 101 001 010 011 100 101 001 010 011 100 101
/spi_clgen/divider_internal 000 100
```

Shift register



Module Declaration: Defines the module spi_shift_register with input and output ports.

● Input:

- rx_negedge: Input signal, indicates the negative edge of the receive clock.
- tx_negedge: Input signal, indicates the negative edge of the transmit clock.
- byte_sel: 4-bit input signal used for byte selection.
- latch: Input signal used for latch control.
- len: 32-bit input signal representing the length of the character.
- p_in: 32-bit input signal for the incoming data.
- wb_clk_in: Input clock signal (system clock).
- wb_rst: Input reset signal.
- go: Input signal to initiate the transfer.
- miso: Input signal for the master input slave output.
- lsb: Input signal indicating the least significant bit.
- cpol_0: Input signal for the pulse marking the positive edge of the sclk_out.
- cpol_1: Input signal for the pulse marking the negative edge of the sclk_out.

● Output:

- p_out: Output signal representing the shifted data from the shift register. It is SPI_MAX_CHAR bits wide.
- last: Output signal indicating whether the last character is being transmitted or received.
- mosi: Output signal representing the serial output data from the shift register.
- tip: Output signal indicating if a transfer is in progress or not.
- rx_clk: Output signal representing the receive clock enable.
- tx_clk: Output signal representing the transmit clock enable.

● Internal Registers:

- char_count: Register used for counting the number of bits in a character.
- master_data: Register representing the shift register holding the data.
- tx_bit_pos: Register indicating the next bit position for transmission.
- rx_bit_pos: Register indicating the next bit position for reception.

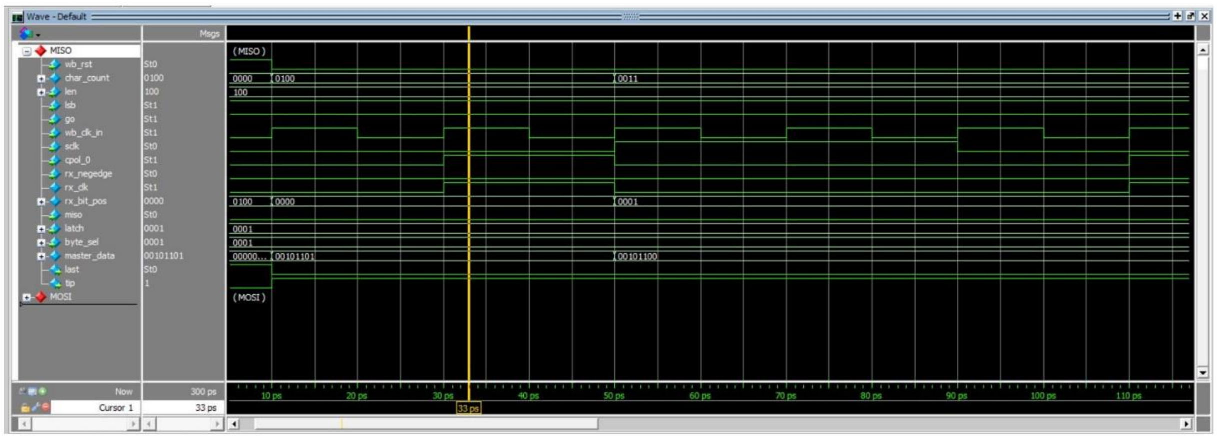
The shift register module takes various inputs such as clock signals, data selection, data input, and control signals. The module includes internal registers for storing data, bit positions for data shifting, and a counter for character bit tracking.

It calculates the transfer in progress and the last character indicator. It also calculates the serial output based on the clock signals and the current bit position.

The module handles the calculation of the bit positions for both transmission and reception based on the configuration. The output is the shifted data from the shift register. The module also supports data latching.

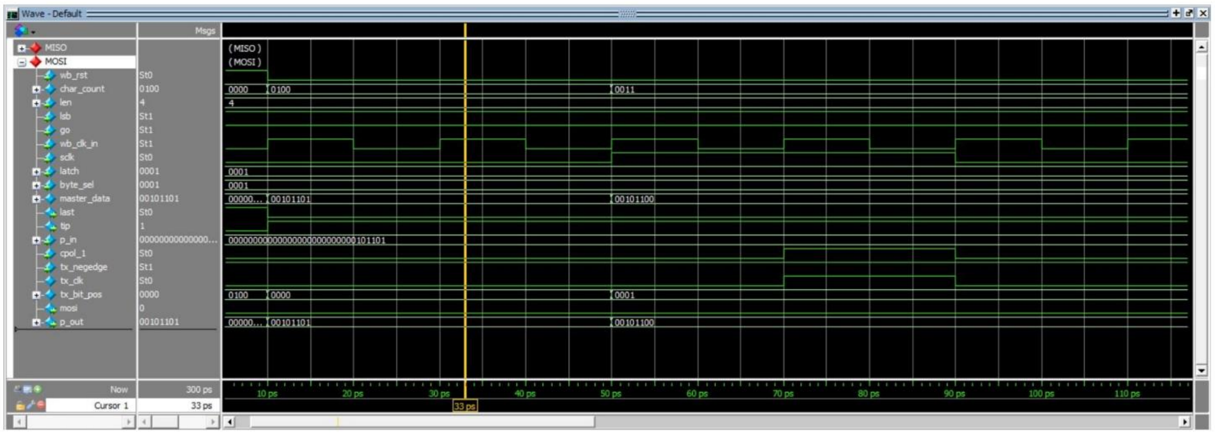
Shift register waveforms

MISO



```
Wave - Default
                      Msgs
MISO
  wb_rst              St0
  char_count          0100
  len                 100
  lsb                 St1
  go                  St1
  wb_clk_in           St1
  sclk                St0
  cpol_0              St1
  rx_negedge          St0
  rx_clk              St1
  rx_bit_pos          0000
  miso                St0
  latch               0001
  byte_sel            0001
  master_data         00101101
  last                St0
  tip                 1
MOSI
Now                   300 ps
Cursor 1              33 ps
```

MOSI



```
Wave - Default
                      Msgs
MISO
MOSI
  wb_rst              St0
  char_count          0100
  len                 4
  lsb                 St1
  go                  St1
  wb_clk_in           St1
  sclk                St0
  latch               0001
  byte_sel            0001
  master_data         00101101
  last                St0
  tip                 1
  p_in                000000000000000...
  cpol_1              St0
  tx_negedge          St1
  tx_clk              St0
  tx_bit_pos          0000
  mosi                0
  p_out               00101101
Now                   300 ps
Cursor 1              33 ps
```

SPI Top Block

It interfaces with a Wishbone bus and provides communication with a master device using the SPI protocol.

● Inputs:

- wb_clk_in: Clock input for the Wishbone bus.
- wb_rst_in: Reset input for the Wishbone bus.
- wb_adr_in: Address input for the Wishbone bus.
- wb_dat_in: Data input for the Wishbone bus.
- wb_sel_in: Select input for the Wishbone bus.
- wb_we_in: Write enable input for the Wishbone bus.
- wb_stb_in: Strobe input for the Wishbone bus.
- wb_cyc_in: Cycle input for the Wishbone bus.
- miso: Master Input Slave Output (MISO) signal

● Outputs:

- wb_dat_o: Data output for the Wishbone bus.
- wb_ack_out: Acknowledge output for the Wishbone bus.
- wb_int_o: Interrupt output for the Wishbone bus.
- sclk_out: Clock output for the SPI interface.
- mosi: Master Output Slave Input (MOSI) signal.
- ss_pad_o: Slave select output for the SPI interface.

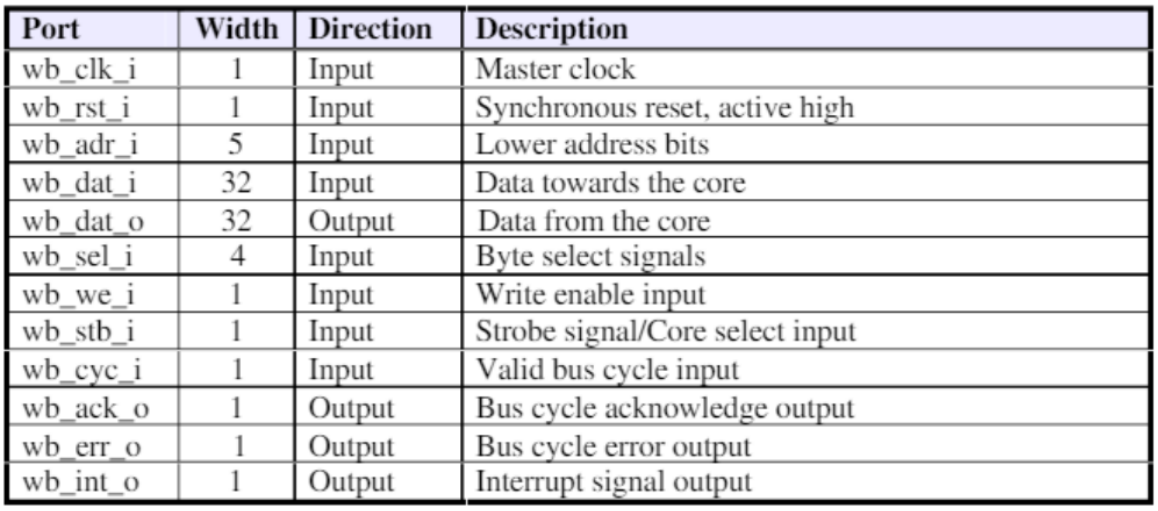
The code instantiates two sub-modules: "spi_clgen" and "spi_shift_reg".

➢ "spi_clgen" generates the SPI clock signal based on the input clock and control signals.
➢ "spi_shift_reg" handles the data shifting and synchronization for SPI communication.

It decodes addresses to read from specific registers and provides output data and acknowledgment signals. It also supports interrupt generation and slave device selection. Overall, it serves as a complete SPI controller for data transfer between a master and multiple slave devices.

Wishbone Master Module Interface

SPI Master acts as a slave to Wishbone Interface. Wishbone master communicates with the host. Bus signals of Wishbone master are as follows:



Port	Width	Direction	Description
wb_clk_i	1	Input	Master clock
wb_rst_i	1	Input	Synchronous reset, active high
wb_adr_i	5	Input	Lower address bits
wb_dat_i	32	Input	Data towards the core
wb_dat_o	32	Output	Data from the core
wb_sel_i	4	Input	Byte select signals
wb_we_i	1	Input	Write enable input
wb_stb_i	1	Input	Strobe signal/Core select input
wb_cyc_i	1	Input	Valid bus cycle input
wb_ack_o	1	Output	Bus cycle acknowledge output
wb_err_o	1	Output	Bus cycle error output
wb_int_o	1	Output	Interrupt signal output

Table 1: Wishbone interface signals

All output WISHBONE signals are registered and driven on the rising edge of wb_clk_i. All

input WISHBONE signals are latched on the rising edge of wb_clk_i.

Module Declaration: Defines the module wishbone_master with input and output ports.

● Inputs:

- clk_in: The clock input for the module.
- rst_in: The reset input for the module.
- ack_in: The acknowledgment input from the slave device.
- err_in: The error input from the slave device.
- dat_in: The data input for write operations.

● Outputs:

- adr_o: The address output for the Wishbone master.
- cyc_o: The cycle output for initiating a Wishbone transaction.
- stb_o: The strobe output for indicating the start of a Wishbone transaction.
- we_o: The write enable output for write operations.
- dat_o: The data output for read operations.
- sel_o: The select output for selecting the active byte lanes.

The module contains a task called “initialize” for initializing the internal signals of the module. It also contains a task called single_write for performing a single write operation. It sets the internal signals adr_temp, sel_temp, we_temp, dat_temp, cyc_temp, and stb_temp based on the provided inputs. It waits for the acknowledgment signal (ack_in) to go low before resetting the internal signals.

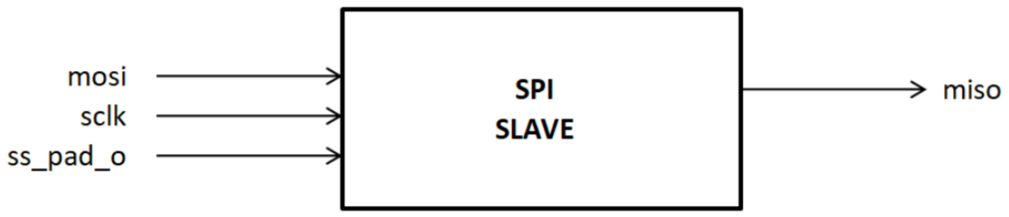
The module uses sequential blocks (always@(posedge clk_in)) to assign values to the output signals based on the internal signals. The rst_in signal is used to handle reset conditions for cyc_o and stb_o.

Overall, the wishbone_master module provides control and data signals for interacting with a Wishbone slave device, allowing for read and write operations.

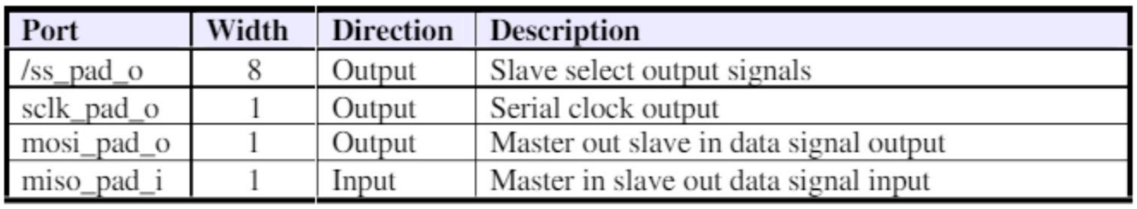
Slave Module Interface

The master can choose which slave it wants to talk to by setting the slave's SS line to a low voltage level. In the idle, non-transmitting state, the slave select line is kept at a high voltage level. Multiple SS pins may be available on the master, which allows for multiple slaves to be wired in parallel. If only one SS pin is present, multiple slaves can be wired to the master by daisy-chaining.

SPI Slave module here just behaves as a "slave" to test for the code results. It is not an actual slave.



Bus Signals for SPI slave module are as follows:



Port	Width	Direction	Description
/ss_pad_o	8	Output	Slave select output signals
sclk_pad_o	1	Output	Serial clock output
mosi_pad_o	1	Output	Master out slave in data signal output
miso_pad_i	1	Input	Master in slave out data signal input

Table 2: SPI external connections

Module Declaration: Defines the module spi_slave with input and output ports.

● Inputs:

- sclk: The serial clock input for synchronization.
- mosi: The input signal representing the data received from the SPI master.
- ss_pad_o: A multi-bit output signal representing the chip select lines for the SPI slave.

● Output:

- miso: The output signal representing the data to be transmitted from the SPI slave.

● Internal signals:

- rx_slave: A register indicating whether the SPI slave is in receive mode.
- tx_slave: A register indicating whether the SPI slave is in transmit mode.
- temp1: A 128-bit register used for temporary storage of received data.
- temp2: A 128-bit register used for temporary storage of received data.

On the posedge of sclk, if the chip select lines are not all high (ss_pad_o != 8'b1111111) and the slave is in receive mode (rx_slave is high) and not in transmit mode (tx_slave is low), the received data mosi is concatenated with the existing data in temp1.

On the negedge of sclk, if the slave is in receive mode (rx_slave is high) and not in transmit mode (tx_slave is low), the value in temp1 is assigned to miso1.

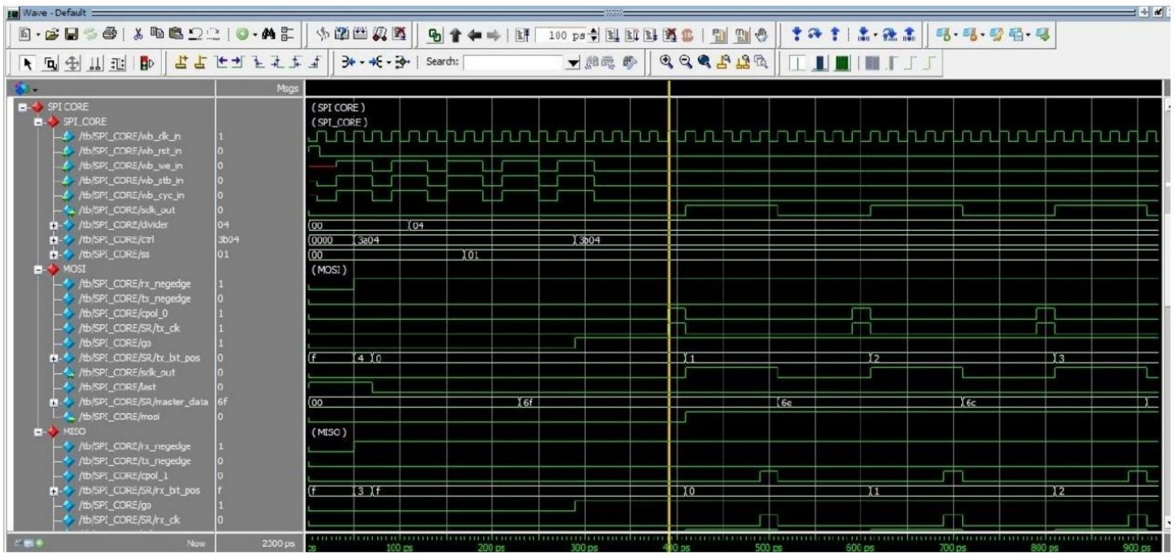
On the posedge of sclk, if the slave is in receive mode (rx_slave is high) and not in transmit mode (tx_slave is low), the value in temp2 is assigned to miso2.

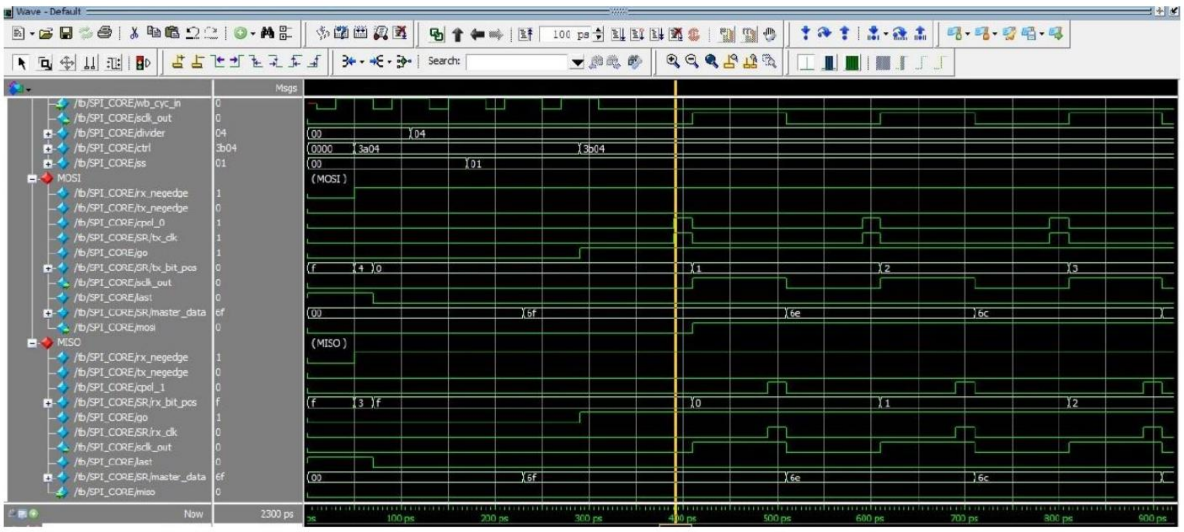
The miso signal is assigned as the logical OR of miso1 and miso2, which represents the data to be transmitted from the SPI slave.

Overall, the spi_slave module receives data from the SPI master on the mosi line, processes and stores it in temp1 and temp2, and provides the output miso for transmitting data back to the SPI master. The chip select lines (ss_pad_o) are used to control the slave operation

Test Case Module Waveforms

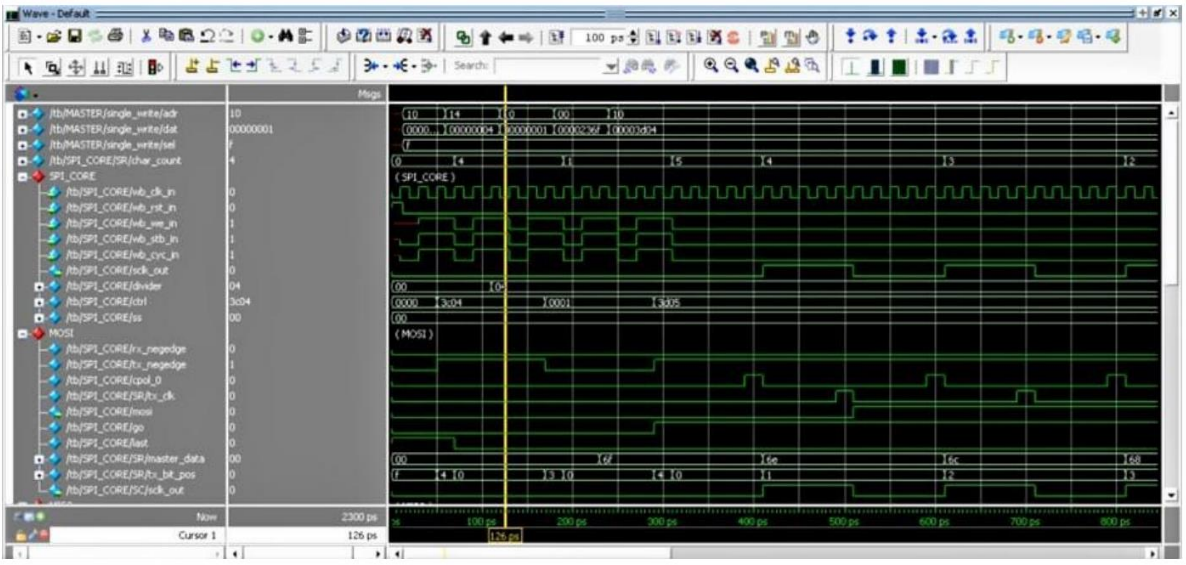
a. RX negedge = 1 and TX negedge = 0


```
SPI CORE
  SPI_CORE
    /tb/SPI_CORE/wb_clk_in           1
    /tb/SPI_CORE/wb_rst_in           0
    /tb/SPI_CORE/wb_we_in            0
    /tb/SPI_CORE/wb_stb_in           0
    /tb/SPI_CORE/wb_cyc_in           0
    /tb/SPI_CORE/sclk_out            0
    /tb/SPI_CORE/divider             04
    /tb/SPI_CORE/ctrl                3b04
    /tb/SPI_CORE/ss                  01
  MOSI
    /tb/SPI_CORE/rx_negedge          1
    /tb/SPI_CORE/tx_negedge          0
    /tb/SPI_CORE/cpol_0              1
    /tb/SPI_CORE/SR/tx_clk           1
    /tb/SPI_CORE/go                  1
    /tb/SPI_CORE/SR/tx_bit_pos       0
    /tb/SPI_CORE/sclk_out            0
    /tb/SPI_CORE/last                0
    /tb/SPI_CORE/SR/master_data      6f
    /tb/SPI_CORE/mosi                0
  MISO
    /tb/SPI_CORE/rx_negedge          1
    /tb/SPI_CORE/tx_negedge          0
    /tb/SPI_CORE/cpol_1              0
    /tb/SPI_CORE/SR/rx_bit_pos       f
    /tb/SPI_CORE/go                  1
    /tb/SPI_CORE/SR/rx_clk           0
Now                                  2300 ps
```

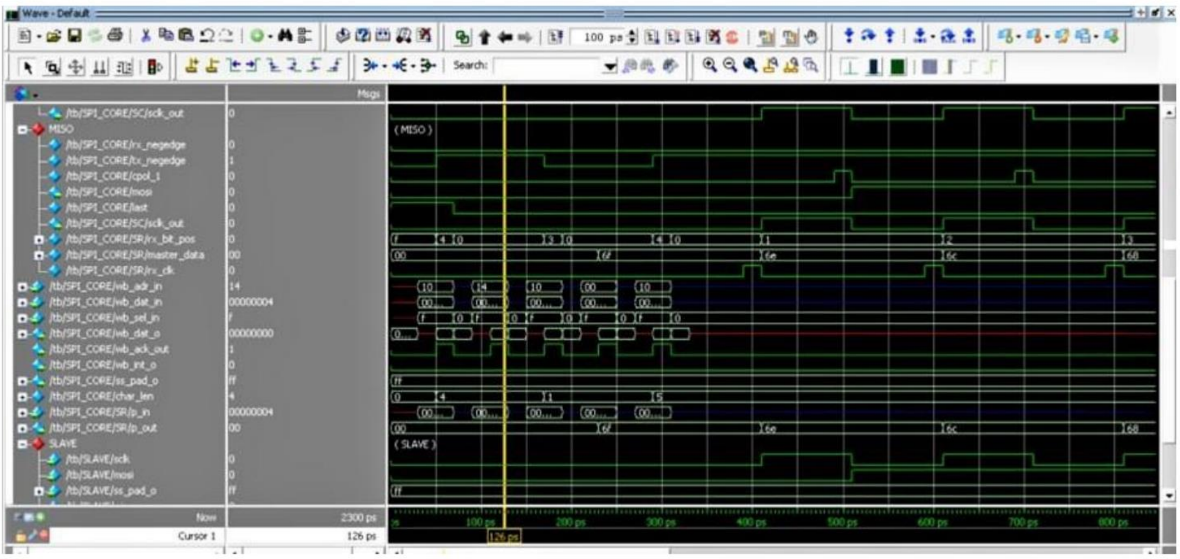


```
Wave - Default
                                          Msgs
    /tb/SPI_CORE/wb_cyc_in                0
    /tb/SPI_CORE/sclk_out                 0
+   /tb/SPI_CORE/divider                  04
+   /tb/SPI_CORE/ctrl                     3b04
+   /tb/SPI_CORE/ss                       01
- MOSI
    /tb/SPI_CORE/rx_negedge               1
    /tb/SPI_CORE/tx_negedge               0
    /tb/SPI_CORE/cpol_0                   1
    /tb/SPI_CORE/SR/tx_clk                1
    /tb/SPI_CORE/go                       1
+   /tb/SPI_CORE/SR/tx_bit_pos            0
    /tb/SPI_CORE/sclk_out                 0
    /tb/SPI_CORE/last                     0
+   /tb/SPI_CORE/SR/master_data           ef
    /tb/SPI_CORE/mosi                     0
- MISO
    /tb/SPI_CORE/rx_negedge               1
    /tb/SPI_CORE/tx_negedge               0
    /tb/SPI_CORE/cpol_1                   0
+   /tb/SPI_CORE/SR/rx_bit_pos            f
    /tb/SPI_CORE/go                       1
    /tb/SPI_CORE/SR/rx_clk                0
    /tb/SPI_CORE/sclk_out                 0
    /tb/SPI_CORE/last                     0
+   /tb/SPI_CORE/SR/master_data           6f
    /tb/SPI_CORE/miso                     0
Now                                       2300 ps
```

b. RX negedge =0 and TX negedge = 1



```
Wave - Default
                                          Msgs
+ /tb/MASTER/single_write/adr             10
+ /tb/MASTER/single_write/dat             00000001
+ /tb/MASTER/single_write/sel             f
+ /tb/SPI_CORE/SR/char_count              4
- SPI_CORE
    /tb/SPI_CORE/wb_clk_in                0
    /tb/SPI_CORE/wb_rst_in                0
    /tb/SPI_CORE/wb_we_in                 1
    /tb/SPI_CORE/wb_stb_in                1
    /tb/SPI_CORE/wb_cyc_in                1
    /tb/SPI_CORE/sclk_out                 0
+   /tb/SPI_CORE/divider                  04
+   /tb/SPI_CORE/ctrl                     3c04
+   /tb/SPI_CORE/ss                       00
- MOSI
    /tb/SPI_CORE/rx_negedge               0
    /tb/SPI_CORE/tx_negedge               1
    /tb/SPI_CORE/cpol_0                   0
    /tb/SPI_CORE/SR/tx_clk                0
    /tb/SPI_CORE/mosi                     0
    /tb/SPI_CORE/go                       0
    /tb/SPI_CORE/last                     0
+   /tb/SPI_CORE/SR/master_data           00
+   /tb/SPI_CORE/SR/tx_bit_pos            0
    /tb/SPI_CORE/SC/sclk_out              0
Now                                       2300 ps
Cursor 1                                  126 ps
```



```
                                      Msgs
  /tb/SPI_CORE/SC/sclk_out            0
MISO
  /tb/SPI_CORE/rx_negedge             0
  /tb/SPI_CORE/tx_negedge             1
  /tb/SPI_CORE/cpol_1                 0
  /tb/SPI_CORE/mosi                   0
  /tb/SPI_CORE/last                   0
  /tb/SPI_CORE/SC/sclk_out            0
  /tb/SPI_CORE/SR/rx_bit_pos          0
  /tb/SPI_CORE/SR/master_data         00
  /tb/SPI_CORE/SR/rx_clk              0
/tb/SPI_CORE/wb_adr_in                14
/tb/SPI_CORE/wb_dat_in                00000004
/tb/SPI_CORE/wb_sel_in                f
/tb/SPI_CORE/wb_dat_o                 00000000
/tb/SPI_CORE/wb_ack_out               1
/tb/SPI_CORE/wb_int_o                 0
/tb/SPI_CORE/ss_pad_o                 ff
/tb/SPI_CORE/char_len                 4
/tb/SPI_CORE/SR/p_in                  00000004
/tb/SPI_CORE/SR/p_out                 00
SLAVE
  /tb/SLAVE/sclk                      0
  /tb/SLAVE/mosi                      0
  /tb/SLAVE/ss_pad_o                  ff
Now                                   2300 ps
Cursor 1                              126 ps
```