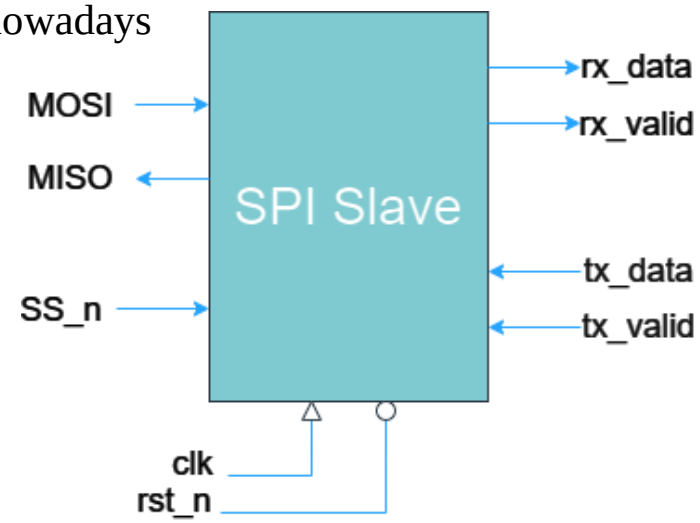
IEEE
2024 –

SPI Protocol

IEEE CAIRO
UNIVERSITY STUDENT
BRANCH

Part 1: SPI Slave Interface

- One of the most popular Interfaces nowadays
- Stands for Serial-Peripheral Interface
- High Data Rates

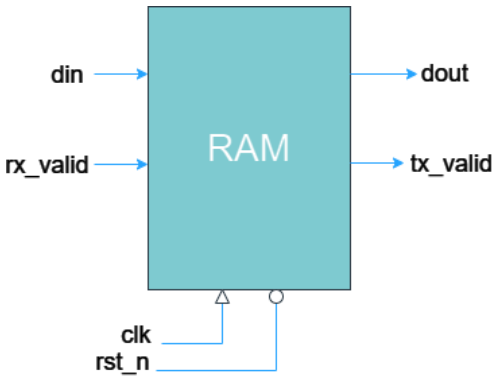


- Ports

Name	Type	Size	Description
MOSI	Input	1 bit	Master Out Slave In
tx_valid		1 bit	Control for Input Data
tx_data		8 bits	Input Data for SPI Slave
rst_n		1 bit	Active low asynchronous reset
clk		1 bit	Clock
MISO	Output	1 bit	Data Output
rx_valid		1 bit	Control for Output Data
rx_data		10 bits	Output Data for SPI Slave

Part 2: Single-port Async RAM

- Parameters
  1. MEM_DEPTH, Default: 256
  2. ADDR_SIZE, Default: 8



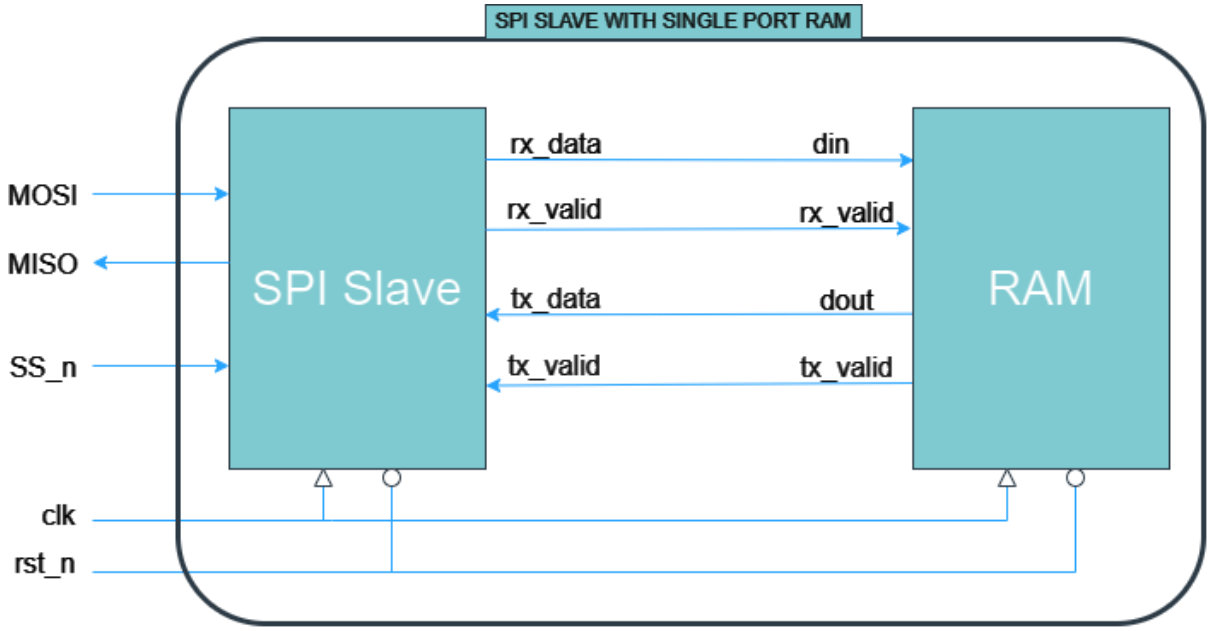
- Ports

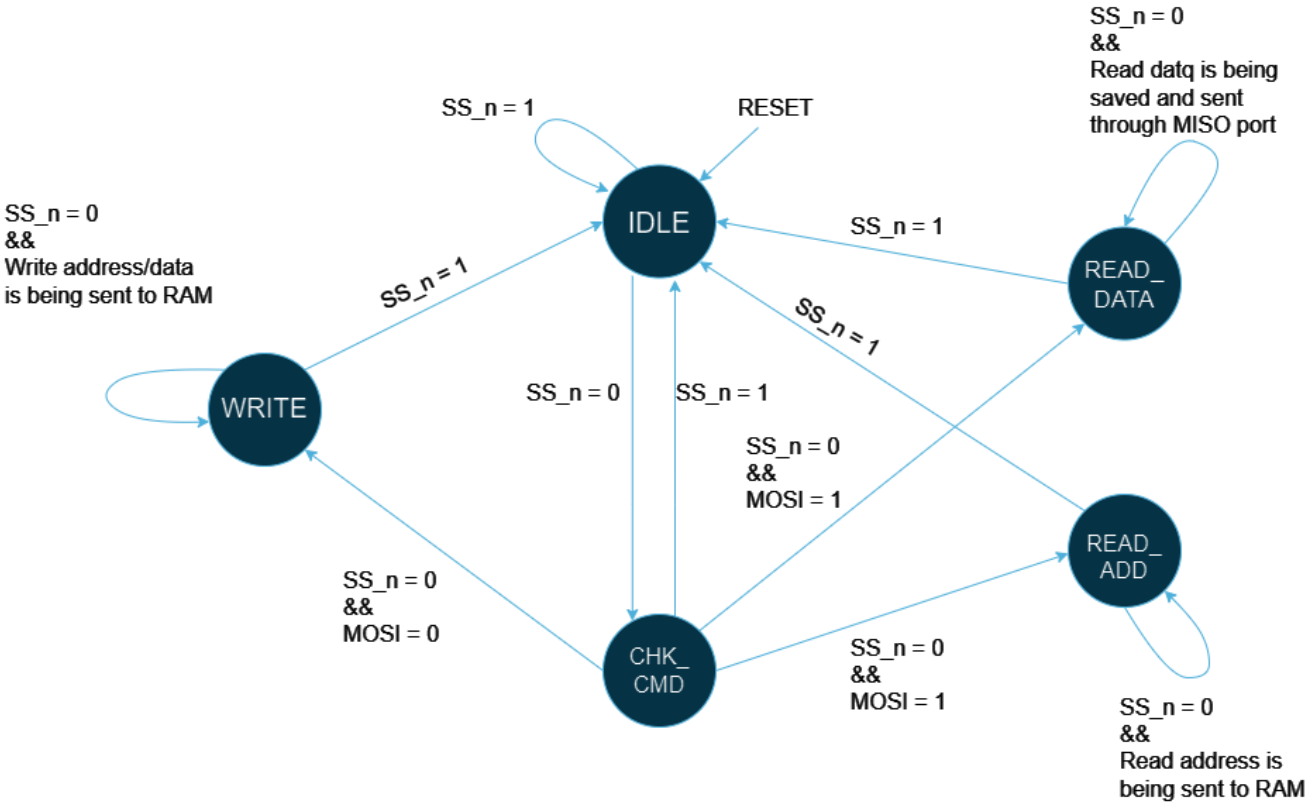
Name	Type	Size	Description
din	Input	10 bits	Data Input
clk		1 bit	Clock
rst_n		1 bit	Active low asynchronous reset
rx_valid		1 bit	If HIGH: accept din[7:0] to save the write/read address internally or write a memory word depending on the most significant 2 bits din[9:8]
dout	Output	8 bits	Data Output
tx_valid		1 bit	Whenever the command is memory read the tx_valid should be High

- Most significant din bit “din[9]” determines if it is a write or read command.

Port	din[9:8]	Command	Description
din	00	Write address	Hold din[7:0] internally as write address
	01	Write data	Write din[7:0] in the memory with write address held previously
	10	Read Address	Hold din[7:0] internally as read address
	11	Read data	Read the memory with read address held previously, tx_valid should be HIGH, dout holds the word read from the memory, ignore din[7:0]

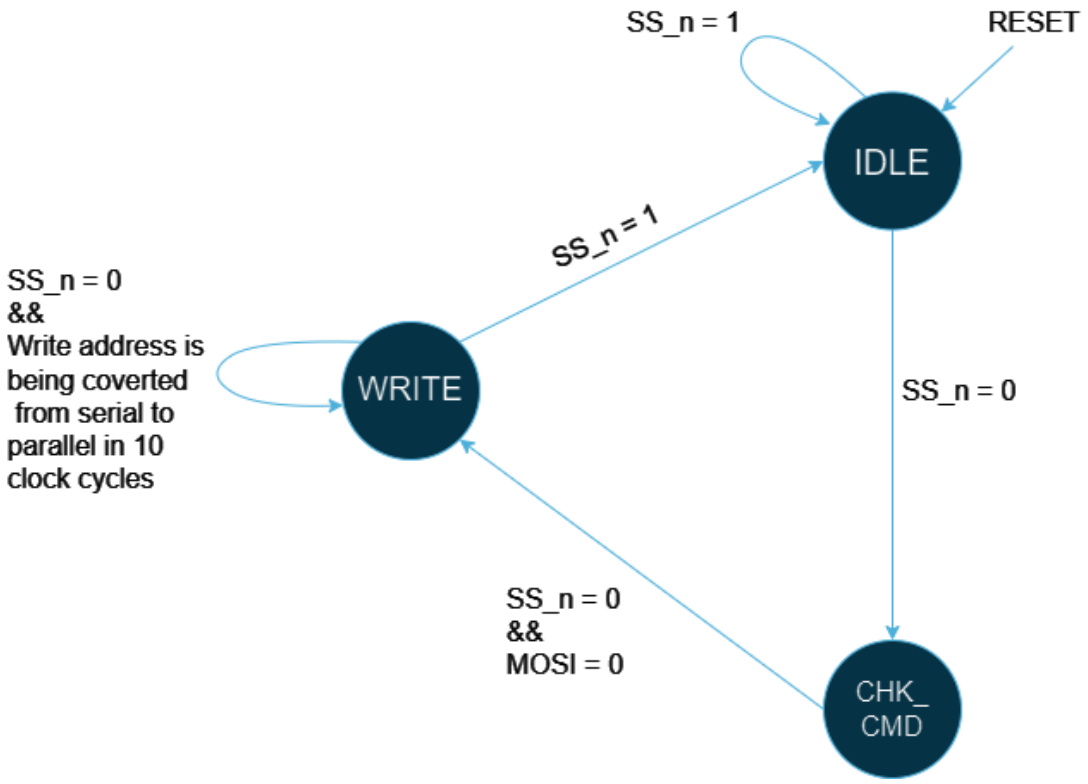
Part 3: SPI Wrapper





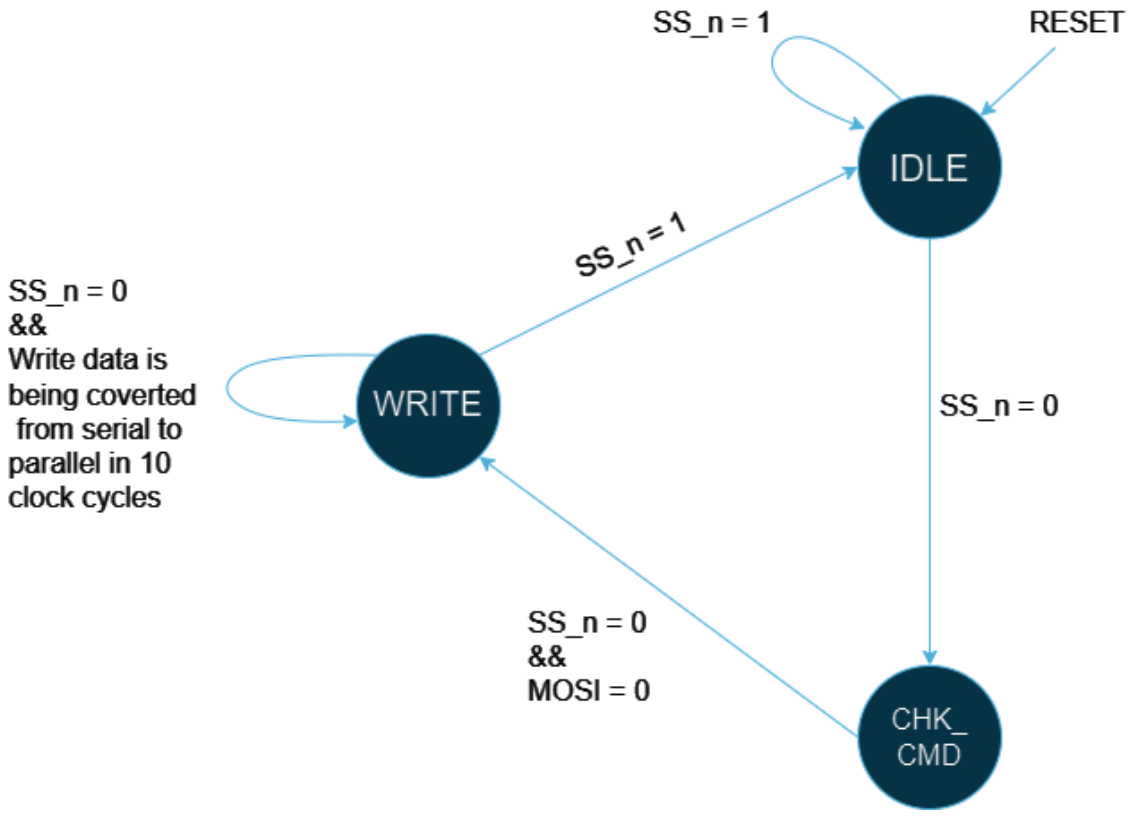
Write Address

1. Master will start the write command by sending the write address value, rx_data[9:8] = din[9:8] = 2'b00.
2. SS_n = 0 to tell the SPI Slave that the master will begin communication.
3. SPI Slave check the first received bit on MOSI port '0' which is a control bit to let the slave determine which operation will take place "write in this case". SPI Slave then expects to receive 10 bits, the first 2 bits are "00" on two clock cycles and then the wr_address will be sent on 8 clock cycles.
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus.
5. rx_valid will be HIGH (only one clock cycle) to inform the RAM that it should expect data on din bus.
6. din takes the value of rx_data.
7. RAM checks on din[9:8] and find that they hold "00".
8. RAM stores din[7:0] in the internal write address bus.
9. SS_n= 1 to end communication from Master side.



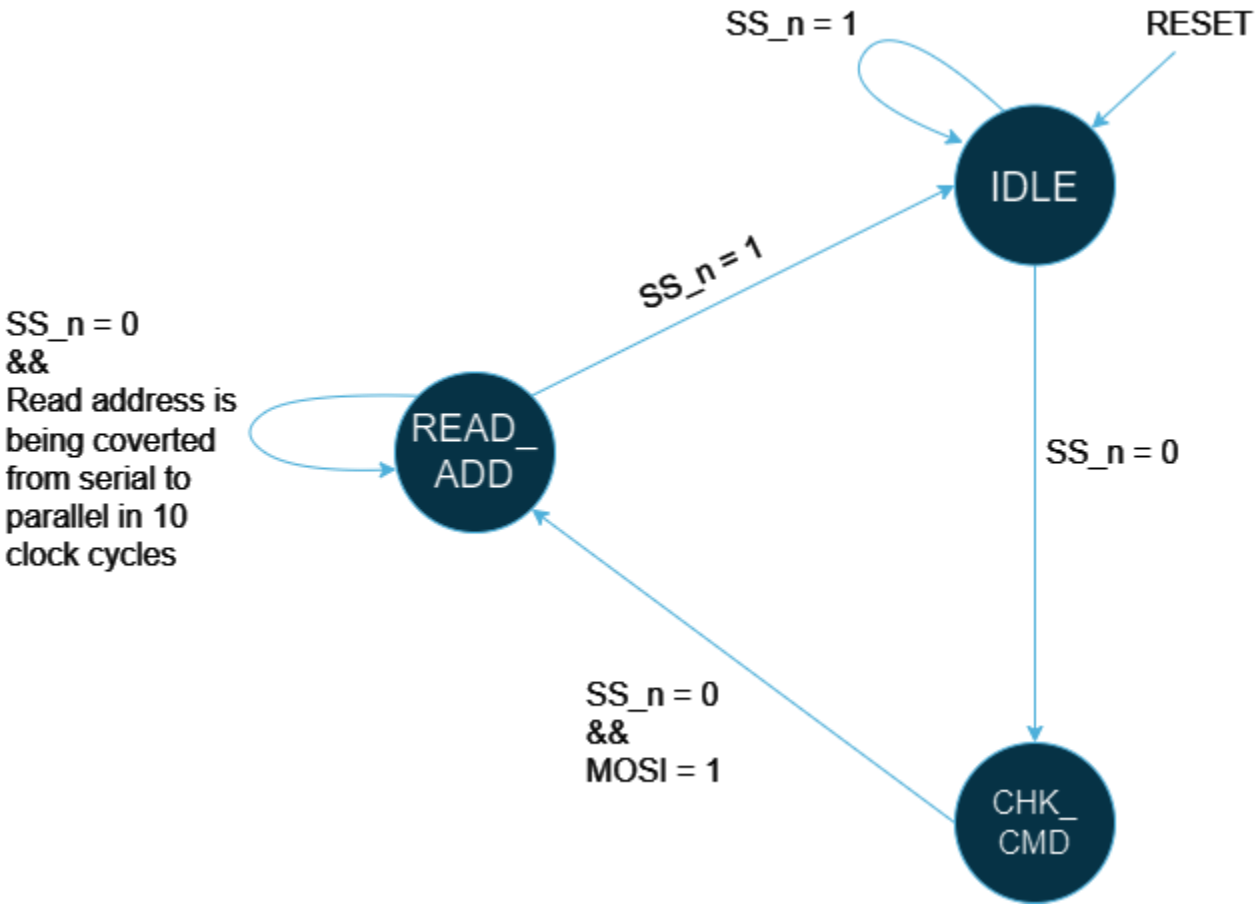
Write Data

1. Master will continue the write command by sending the write data value, rx_data[9:8] = din[9:8] = **2'b01.**
2. SS_n = 0 to tell the SPI Slave that the master will begin communication.
3. SPI Slave check the first received bit on MOSI port 'O' which is a control bit to let the slave determine which operation will take place "write in this case". SPI Slave then expects to receive 10 more bits, the first 2 bits are "01" on two clock cycles and then the **wr_data** will be sent on 8 more clock cycles.
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus.
5. rx_valid will be HIGH (only one clock cycle) to inform the RAM that it should expect data on din bus.
6. din takes the value of rx_data.
7. RAM checks on din[9:8] and find that they hold "01".
8. RAM stores din[7:0] in the RAM with **wr_address** previously held.
9. SS_n= 1 to end communication from Master side.



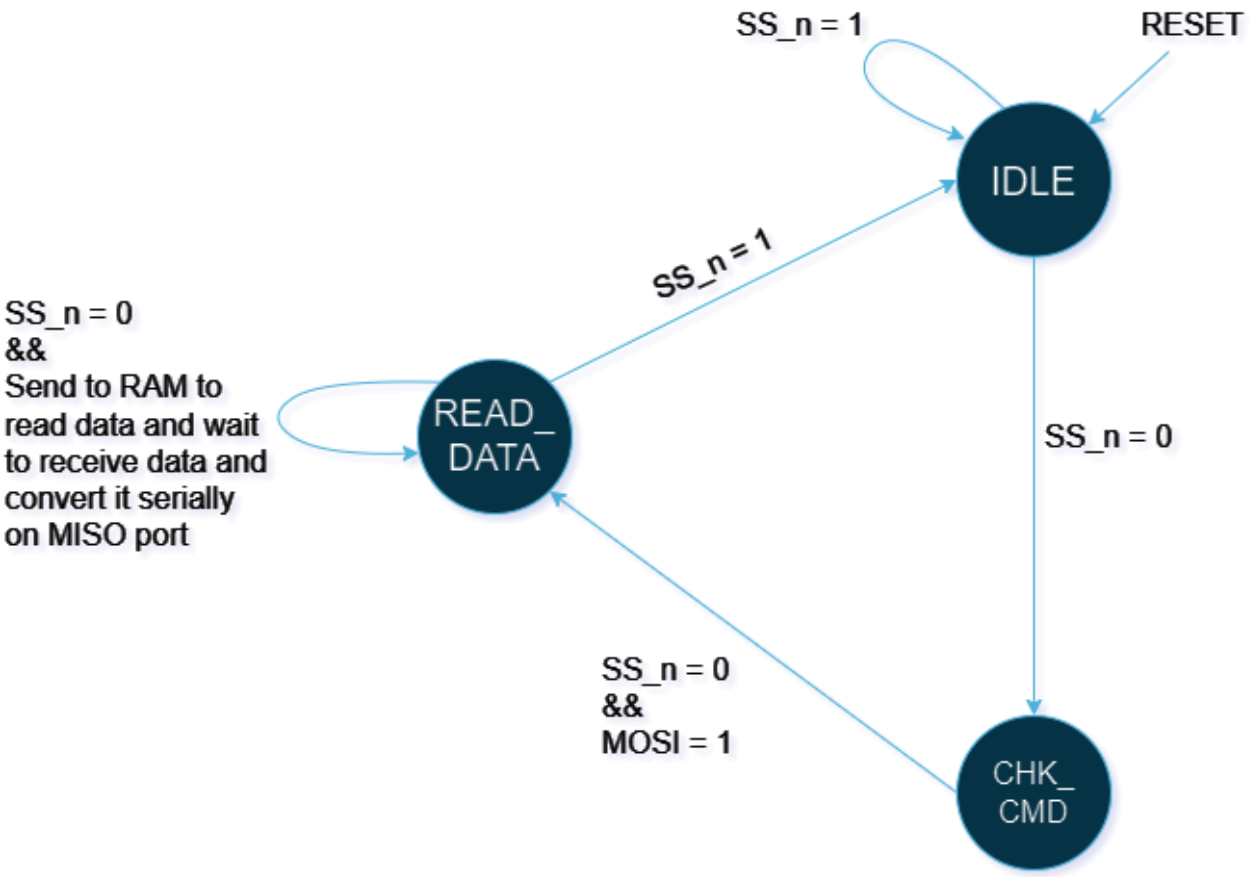
Read Address

1. Master will start the read command by sending the read address value, x_data[9:8] = din[9:8] = 2'b10.
2. SS_n = 0 to tell the SPI Slave that the master will begin communication.
3. SPI Slave check the first received bit on MOSI port '1' which is a control bit to let the slave determine which operation will take place "read in this case". SPI Slave then expects to receive 10 bits, the first 2 bits are "10" on two clock cycles and then the **rd_address** will be sent on 8 more clock cycles.
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus.
5. rx_valid will be HIGH (only one clock cycle) to inform the RAM that it should expect data on din bus.
6. din takes the value of rx_data.
7. RAM checks on din[9:8] and find that they hold "10".
8. RAM stores din[7:0] in the internal read address bus.
9. SS_n = 1 to end communication from Master side.



Read Data

1. Master will continue the read command by sending the read data command, rx_data[9:8] = din[9:8] =2'b11.
2. SS_n = 0 to tell the SPI Slave that the master will begin communication.
3. SPI Slave check the first received bit on MOSI port '1' which is a control bit to let the slave determine which operation will take place "read in this case". SPI Slave then expects to receive 10 bits, the first 2 bits are "11" on two clock cycles and then dummy data will be sent and ignored since the master is waiting for the data to be sent from slave side.
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus.
5. din takes the value of rx_data.
6. RAM reads din[9:8] and find that they hold "11".
7. RAM will read from the memory with rd_address previously held.
8. RAM will assert tx_valid to inform slave that data out is ready.
9. Slave reads tx_data and convert it into serial out data on MISO port.
10. tx_valid will be high 9 clk (1+8).
11. SS_n = 1, Master ends communication after receiving data 8 clock cycles.



Requirements

- SPI SLAVE implementation suggestions
  - Split the SPI Slave design into three always block.
    - ✓ State Memory always block.
    - ✓ Next state Logic always block.
    - ✓ Output Logic always block with posedge clk as the sensitivity.
  - Extra signals that you will need
    - ✓ Counter for serial to parallel and vice versa Conversion.
    - ✓ Internal Signal to allow SPI slave to memorize if the read address is received or not to decide for READ_ADD or READ_DATA transition.

- Requirement
  1. The project should be documented as a PDF file with the following format: <your_name>_Project. For example, Ayman_Mostafa_Sayed_Project. The PDF should contain snippets from the waveforms captured from ModelSim or QuestaSim, showing the design with inputs assigned values and the output values visible.
  2. You should submit Your RTL Code (3 files .v) with this names.
     - RAM.v
     - SPI_SLAVE.v
     - SPI_Wrapper.v

  Notes: The instantiation in SPI_Wrapper module will be like this (SPI, Ram)

```
SPI_SLAVE SPI (
.MOSI(MOSI),
.MISO(MISO),
.SS_n(SS_n),
.clk(clk),
.rst_n(rst_n),
.rx_data(rxdata),
.rx_valid(rx_valid),
.tx_data(txdata),
.tx_valid(tx_valid)
);
```

```
RAM #(MEM_DEPTH,ADDR_SIZE) Ram (
.din(rxdata),
.dout(txdata),
.rx_valid(rx_valid),
.tx_valid(tx_valid),
.clk(clk),
.rst_n(rst_n)
);
```

  3. Write a testbench (SPI_Wrapper_tb.v) to test the following 4 cases after Reset.
     - Write Address (This case takes 13 cycles).
     - Write Data (This case takes 13 cycles).
     - Read Address (This case takes 13 cycles).
     - Read Data (This case takes 22 cycles).
  4. Write a Do file (SPI_Wrapper.do).
  5. Collect all these files into a .zip file and name it <your_name>.zip .For example, Ayman_Mostafa_Sayed.zip.
  6. Submit the .zip file only in the classroom.