KLE TECH. KLE Technological University
Creating Value
Leveraging Knowledge

RISC V Based SoC Design-SPI

Team Details

Team Number:01 (EEE)

Sl No	Name	USN	Roll No	Division
1	Rahul G Teli	01FE21BEE008	6	A
2	Malhar Kulkarni	01FE21BEE016	12	A
3	Chandru Thomare	01FE21BEE022	17	A
4	Chandrashekar R Angadi	01FE21BEE031	25	A

GUIDE:

Dr Saroja V Siddamal

Prof. Vijay

Prof. Suhas

Prof. Jayashree Mallidu

Outline of Report

Introduction:

The SPI is a synchronous serial interface in which data in a 32-bit byte can be shifted in and/or out one bit at a time.

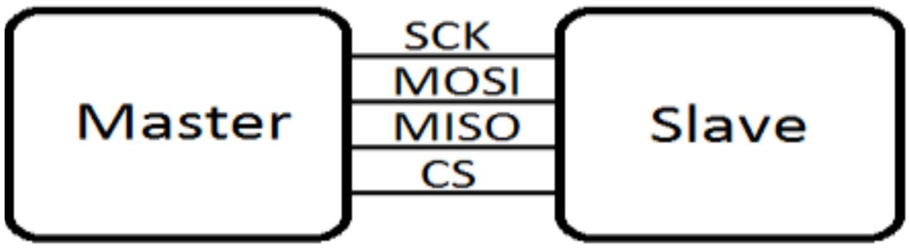
USES: It can be used to communicate with a serial peripheral device or with another microcontroller.

Table 1: SPI Signals

SL.NO	SPI Signal	Name
1	MISO	Master-In-Slave-Out
2	MOSI	Master-Out-Slave-In
3	SCLK	Serial Clock
4	SS	Slave Select

Table 2: SPI Registers

NAME	Register Address	Description
SP0CR1	00D0	SPI Control Register
SP0CR2	00D1	SPI Control Register 2
SP0BR	00D2	SPI Baud Rate Register
SP0SR	00D3	SPI Status Register
SP0DR	00D5	SPI Data Register



SPI with Single Master& Single Slave

A) Master In Slave Out (MISO):

Input of master and output of slave was configured as MISO line. It transfers serial data in only one direction, wherein MSB is sent first.

(B) Master Out Slave In (MOSI):

Input to slave and output to master is done by MOSI line. Data is transmitted in single direction serially, with the MSB.

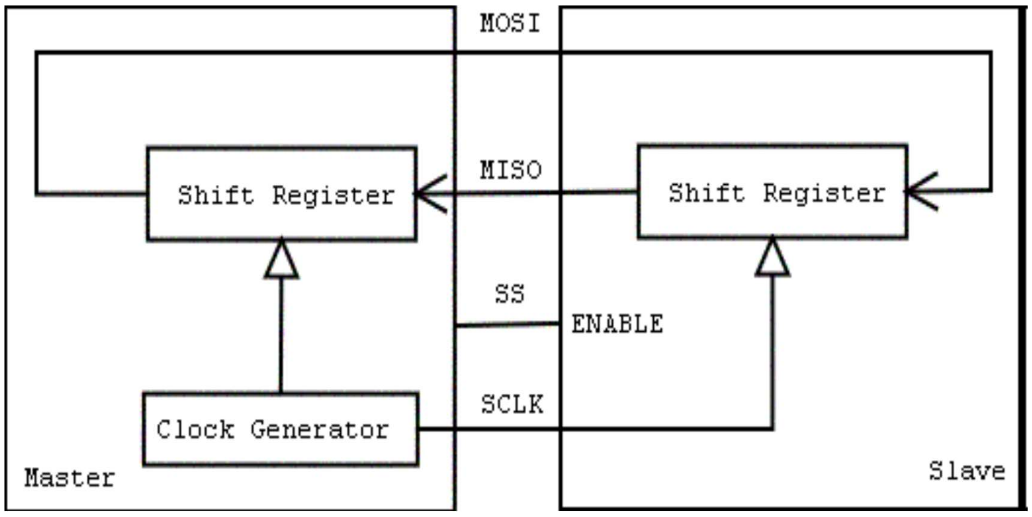
(C) Slave Select (SS Bar):

The device of slave is selected using the select input line of slave, which is active low during data transfer and must stay low throughout the data transfer.

(D) Serial Clock (SCLK):

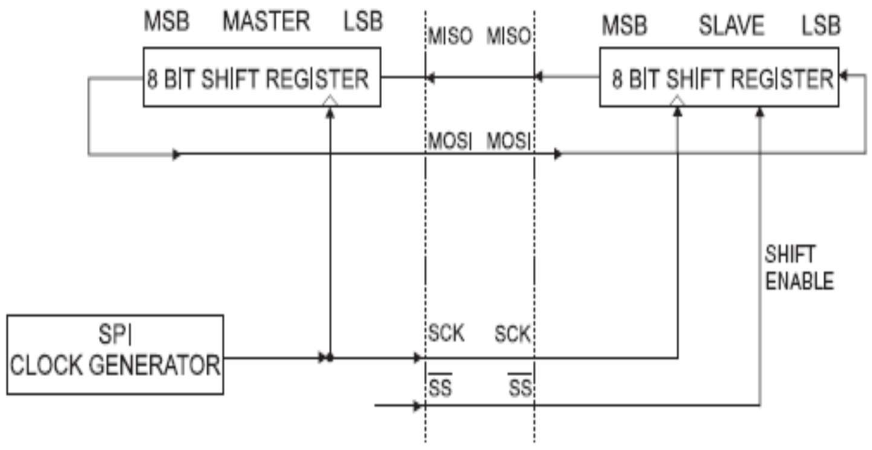
The synchronization of data to and from the device is done using the serial clock, through MOSI & MISO signals. The duration of eight clock cycles, slave and master device are capable in exchanging the data a byte.

Diagram:



Master/Slave Transfer Block Diagram

Synchronous Serial Data Transfer



Master SPI device

Slave SPI device

Register Descriptions

1) SPI Control Register 1 [8 bit] [Read/Write]:

	Bit 7	6	5	4	3	2	1	Bit 0
R / W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset:	0	0	0	0	0	1	0	0

- **SPIE** — SPI Interrupt Enable Bit
  This bit enables SPI interrupt requests, if SPIF or MODF status flag is set.
  1 = SPI interrupts enabled.
  0 = SPI interrupts disabled.

- **SPE** — SPI System Enable Bit
  This bit enables the SPI system and dedicates the SPI port pins to SPI system functions.
  1 = SPI enable, port pins are dedicated to SPI functions.
  0 = SPI disabled (lower power consumption).

- **SPTIE** — SPI Transmit Interrupt Enable
  This bit enables SPI interrupt requests, if SPTEF flag is set.
  1 = SPTEF interrupt enabled. 0 = SPTEF interrupt disabled.

- **MSTR** — SPI Master/Slave Mode Select Bit
  This bit selects, if the SPI operates in master or slave mode. Switching the SPI from master to slave or vice versa forces the SPI system into idle state.
  1 = SPI is in Master mode.
  0 = SPI is in Slave mode.

- **CPOL** — SPI Clock Polarity Bit
  This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values.
  1 = Active-low clocks selected. In idle state SCK is high.
  0 = Active-high clocks selected. In idle state SCK is low.

- **CPHA** — SPI Clock Phase Bit
  This bit is used to select the SPI clock format. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.
  1 = Sampling of data occurs at even edges (2,4,6,...16) of the SCK clock
  0 = Sampling of data occurs at odd edges (1,3,5,...,15) of the SCK clock

- **LSBFE** — LSB-First Enable
  This bit does not affect the position of the MSB and LSB in the data register. Reads and writes of the data register always have the MSB in bit 7.
  1 = Data is transferred least significant bit first.
  0 = Data is transferred most significant bit first.

- **SSOE** — Slave Select Output Enable

The SS output feature is enabled only in master mode, if MODFEN is set, by asserting the SSOE as shown in **Table 3.**

Table 3: SS Input / Output Selection

MOD FEN	**SSOE**	**Master Mode**	**Slave Mode**
0	0	SS not used by SPI	SS input
0	1	SS not used by SPI	SS input
1	0	SS input with MODF feature	SS input
1	1	SS is slave select output	SS input

2) SPI Control Register 2 [8 bit] [Read/Write]:

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W	▨	▨	▨			▨		
Reset:	0	0	0	0	0	0	0	0

▨ = Reserved

- **MODFEN** — Mode Fault Enable Bit
  This bit allows the MODF failure being detected. If the SPI is in Master mode and MODFEN is cleared, then the SS port pin is not used by the SPI. In Slave mode, the SS is available only as an input regardless of the value of MODFEN.
  1 = SS port pin with MODF feature
  0 = SS port pin is not used by the SPI

- **BIDIROE** — Output enabled in the Bidirectional mode of operation
  This bit controls the MOSI and MISO output buffer of the SPI, when in bidirectional mode of operation (SPC0 is set). In master mode this bit controls the output buffer of the MOSI port, in slave mode it controls the output buffer of the MISO port.
  1 = Output buffer enabled
  0 = Output buffer disabled
- **SPISWAI** — SPI Stop in Wait Mode Bit
  This bit is used for power conservation while in wait mode.
  1 = Stop SPI clock generation when in wait mode.
  0 = SPI clock operates normally in wait mode.

- **SPC0** — Serial Pin Control Bit 0
  This bit enables bidirectional pin configurations as shown in **Table 4**.

Table 4 Bidirectional Pin Configurations

Pin Mode	SPC0	BIDIROE	MISO	MOSI
Master Mode of Operation				
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
Slave Mode of Operation				
Normal	0	X	Slave Out	SlaveIn
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

3) SPI Baud Rate Register [8 bit]:

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
W	▨				▨			
Reset:	0	0	0	0	0	0	0	0

▨ = Reserved

SPPR2–SPPR0 — SPI Baud Rate Preselection Bits
SPR2–SPR0 — SPI Baud Rate Selection Bits
These bits specify the SPI baud rates as shown in the table below. In master mode, a change of these bits will abort a transmission in progress and force the SPI system into idle state.

4) SPI Status Register [8 bit] [Only Read & Not Write]:

	Bit 7	6	5	4	3	2	1	Bit 0
R	SPIF	0	SPTEF	MODF	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	1	0	0	0	0	0

▨ = Reserved

- **SPIF** — SPIF Interrupt Flag
  This bit is set after a received data byte has been transferred into the SPI Data Register. This bit is cleared by reading the SPISR register (with SPIF set) followed by a read access to the SPI Data Register.
  1 = New data copied to SPIDR
  0 = Transfer not yet complete
- **SPTEF** — SPI Transmit Empty Interrupt Flag
  If set, this bit indicates that the transmit data register is empty. To clear this bit and place data into the transmit data register, SPISR has to be read with SPTEF=1, followed by a write to SPIDR.
  1 = SPI Data register empty.
  0 = SPI Data register not empty.

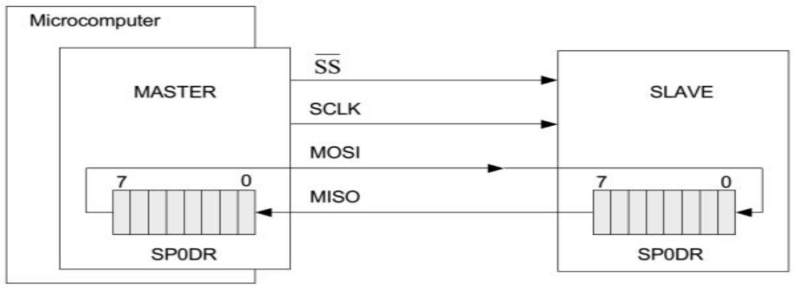
- **MODF** — Mode Fault Flag
  This bit is set if the SS input becomes low while the SPI is configured as a master and mode fault detection is enabled, MODFEN bit of SPICR2 register is set. The flag is cleared automatically by a read of the SPI Status Register (with MODF set) followed by a write to the SPI Control Register 1.
  1 = Mode fault has occurred.
  0 = Mode fault has not occurred.

5) SPI Data Register [8 bit]:

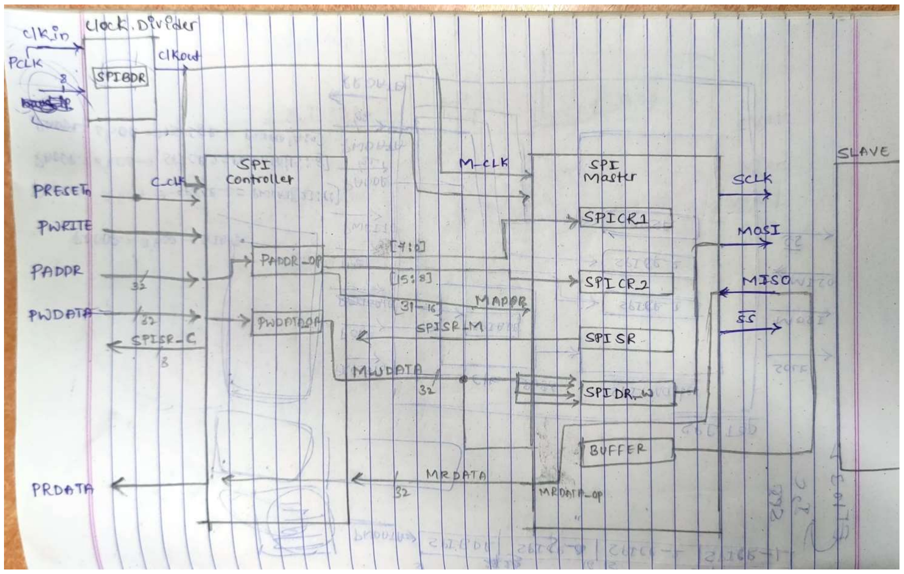
	Bit 7	6	5	4	3	2	1	Bit 0
R / W	Bit 7	6	5	4	3	2	2	Bit 0
Reset:	0	0	0	0	0	0	0	0

- The SPI Data Register is both the input and output register for SPI data. A write to this register allows a data byte to be queued and transmitted. For a SPI configured as a master, a queued data byte is transmitted immediately after the previous transmission has completed.
- Received data in the SPIDR is valid when SPIF is set.
- If SPIF is cleared and a byte has been received, the received byte is transferred from the receive shift register to the SPIDR and SPIF is set.
- If SPIF is set and not serviced, and a second byte has been received, the second received byte is kept as valid byte in the receive shift register until the start of another transmission.

Two SPI modules Connected in a Master-Slave Configuration.



SPI DIAGRAM:



Controller Block

- It is a configurable SPI controller that allows setting parameters like data width, address width, clock polarity/phase, baud rate etc. through register writes.
- The SPICR_1, SPICR_2 and SPIBDR registers allow configuring the SPI mode parameters. Writing to these registers saves the configuration.
- The MWDATA register is used to write the data to be transmitted. Reading the MRDATA gives the received SPI data.
- It generates the SPI clock using a clock divider circuit. The baud rate is configurable by writing to the SPIBDR register.
- The SPI state machine, transmission and reception logic is not shown here. Only the configuration registers, clock generation and interface to access the registers is defined.
- It provides interfaces like chip select signals, interrupts etc. along with the SPI lines SCK, MOSI, MISO.
- this is a configurable SPI controller design that allows setting various SPI communication parameters through register access. The module provides the necessary interfaces and a programmable clock divider to generate the SCK. The state machine and data transfer logic for SPI transmission/reception is likely implemented separately.

Master Block

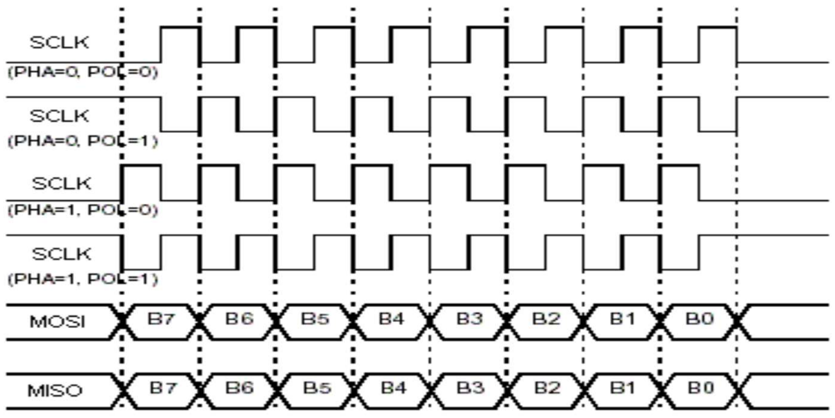
- It is a configurable SPI master that allows setting parameters like data width, clock polarity/phase, LSB/MSB first etc. through the SPICR registers.
- The core logic involves SPI data transmission in 4 8-bit chunks. Transmit and receive shift registers (SPIDR_W, SPIDR_R) are used to shift out TX data and capture RX data.
- The state machine has 4 states - idle, setup, write, read. Setup state configures the transmission, write state handles MOSI data shifting out, while read state shifts in MISO data.
- The sclk generation logic creates the serial clock based on configurable parameters like polarity, phase. The baud rate can also be configured.
- Interface signals include ss, sclk, mosi, miso along with registers like SPICR, SPISR for control and status.
- this is a parameterized and configurable SPI master with support for varied data widths. It can handle both transmission and reception through inbuilt shift register based logic. The state machine orchestrates the SPI operations based on parameters set in configuration registers. Overall, this allows customizing SPI communication for different external slave devices.

SPI Top Block

- It instantiates an SPI controller module (spi_controller) and an SPI master module (spi_master).
- The controller module handles the register interface, clock generation and overall SPI parameters configuration.
- The master module has the actual SPI data transmission and reception logic.
- Configuration registers like SPICR_1, SPICR_2 are used to set SPI modes and are common for both modules.
- The controller sets these registers and passes to the master module. It also handles the clock generation for SPI communication.
- Master module uses these configurations to control the SPI transfer. It also has access to data registers for transmission and reception.
- The MISI, MOSI, SS, SCLK signals are driven from the master module based on data transfers controlled by the state machine.
- this creates a configurable SPI system with flexible interfaces. The controller and master separation allows reusing modules and customizing setups. The registers form the backbone to configure operational modes. Overall, this implements an SPI architecture following standard building blocks.

SPI Testbench

- It instantiates the SPI top module (spi_top) and drives stimulus to it.
- Stimulus includes the control signals like clock, reset, register writes to configure SPI. Also, data to transmit and simulate MISO input.
- It first writes the configuration registers SPICR_1, SPICR_2 to set parameters like clock polarity, phase etc.
- It writes data to the transmit register, then toggles clock to complete an SPI transfer.
- To simulate reception, it is driving MISO signal with dummy data, which gets captured.
- Finally, it reads the received data and control registers to check proper functioning.
- this testbench automates the process of configuring the SPI module, conducting full data transfers by toggling clocks and finally reading back status registers and data. It verifies transmission and reception through interacting with interface signals. The spi_top interfaces are appropriately stimulated here**.**

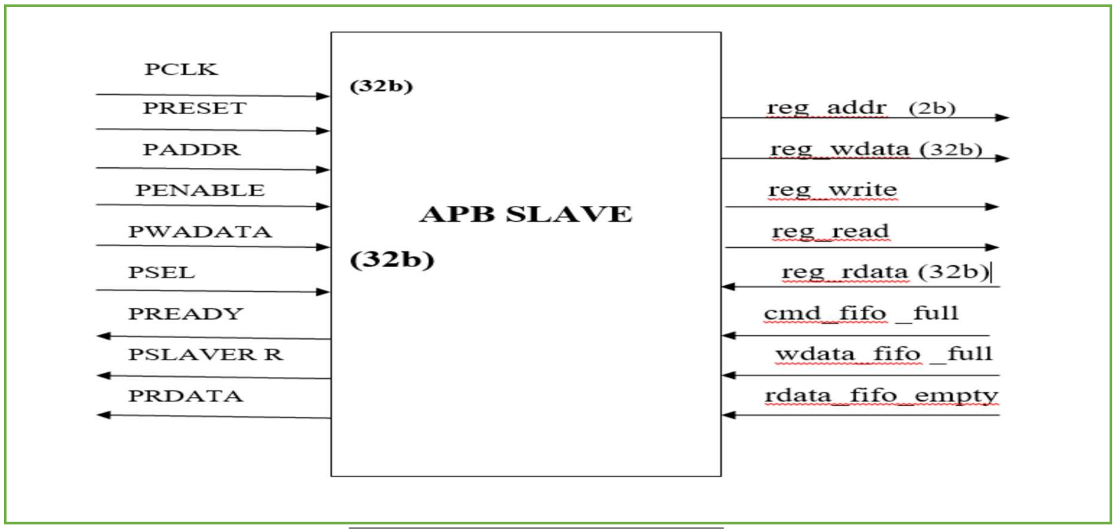


SPI Modes based on CPOL and CPHA

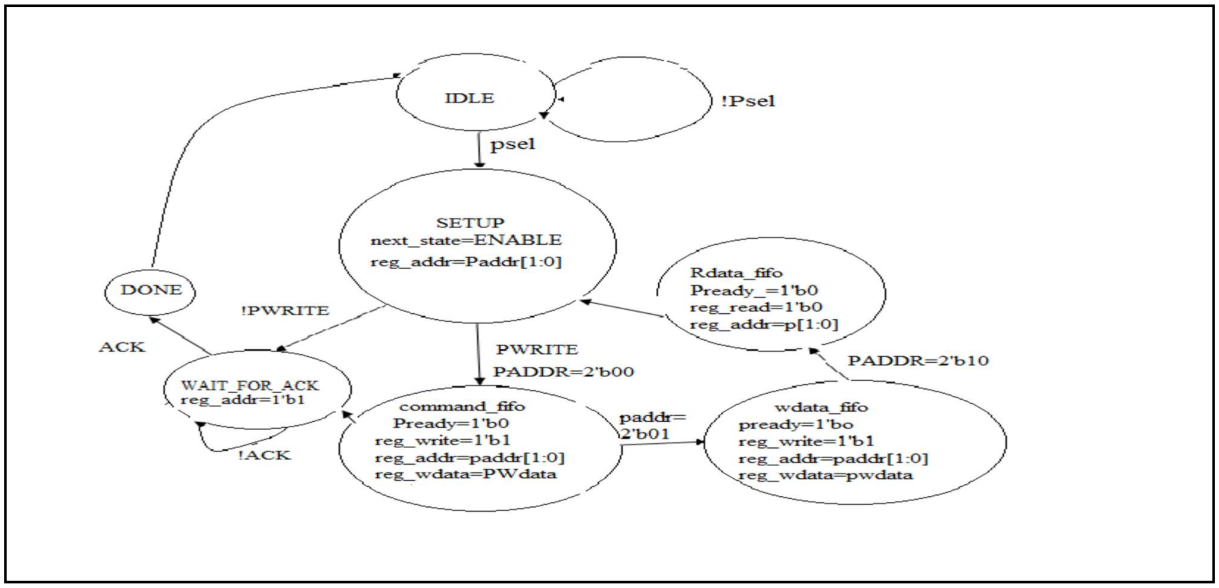
Signal Descriptions:

SL.NO	Signal	Source	Width	Description
1	**PCLK**	Clock	1	Clock. **PCLK** is a clock signal. All APB signals are timed against the rising edge of **PCLK**
2	**PRESET(n)**	System bus reset	1	Reset. **PRESET(n)** is the reset signal and is active-LOW. **PRESET(n)** is normally connected directly to the system bus reset signal
3	**PADDR**	Requester	ADDR_WIDTH	Address. **PADDR** is the APB address bus. **PADDR** can be up to 32 bits wide.
4	**PPROT**	Requester	3	Protection type. **PPROT** indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
5	**PNSE**	Requester	1	Extension to protection type.
6	**PSEL x**	Requester	1	Select. The Requester generates a **PSELx** signal for each Completer. **PSELx** indicates that the Completer is selected and that a data transfer is required.
7	**PENABLE**	Requester	1	Enable. **PENABLE** indicates the second and subsequent cycles of an APB transfer.
8	**PWRITE**	Requester	1	Direction. **PWRITE** indicates an APB write access when HIGH and an APB read access when LOW.
9	**PWDATA**	Requester	DATA_WIDTH	Write data. The **PWDATA** write data bus is driven by the APB bridge Requester during write cycles when **PWRITE** is HIGH. **PWDATA** can be 8 bits, 16 bits, or 32 bits wide
10	**PSTRB**	Requester	DATA_WIDTH/8	Write strobe. **PSTRB** indicates which byte lanes to update during a write transfer. There is one write strobe for each 8 bits of the write data bus. **PSTRB[n]** corresponds to **PWDATA [(8n + 7): (8n)]**. **PSTRB** must not be active during a read transfer.
11	**PREADY**	Completer	1	Ready. **PREADY** is used to extend an APB transfer by the Completer
12	**PRDATA**	Completer	DATA_WIDTH	Read data. The **PRDATA** read data bus is driven by the selected Completer during read cycles when **PWRITE** is LOW. **PRDATA** can be 8 bits, 16 bits, or 32 bits wide
13	**PSLVERR**	Completer	1	Transfer error. **PSLVERR** is an optional signal that can be asserted HIGH by the Completer to indicate an error condition on an APB transfer.
14	**PWAKEUP**	Requester	1	Wake-up. **PWAKEUP** indicates any activity associated with an APB interface.

15	**PAUSER**	Requester	USER_REQ_WIDTH	User request attribute. **PAUSER** is recommended to have a maximum width of 128 bits
16	**PWUSER**	Requester	USER_DATA_WIDTH	User write data attribute. **PWUSER** is recommended to have a maximum width of DATA_WIDTH/2.
17	**PRUSER**	Completer	USER_DATA_WIDTH	User read data attribute. **PRUSER** is recommended to have a maximum width of DATA_WIDTH/2.
18	**PBUSER**	Completer	USER_RESP_WIDTH	User response attribute. **PBUSER** is recommended to have a maximum width of 16 bits



APB Slave block diagram



State Diagram for APB Slave

APB Access Constraints:

- APB (Advanced Peripheral Bus) only supports single-word 32-bit accesses.
- Bits [1:0] of the PADDR signal are not used, resulting in byte and half-word accesses being treated as word accesses.

Memory Controller Operation:

- APB interface enables the memory controller state of operation.
- It allows programming the memory controller with correct timings and settings for the connected memory type.
- Initializes connected memory devices during the initialization phase.

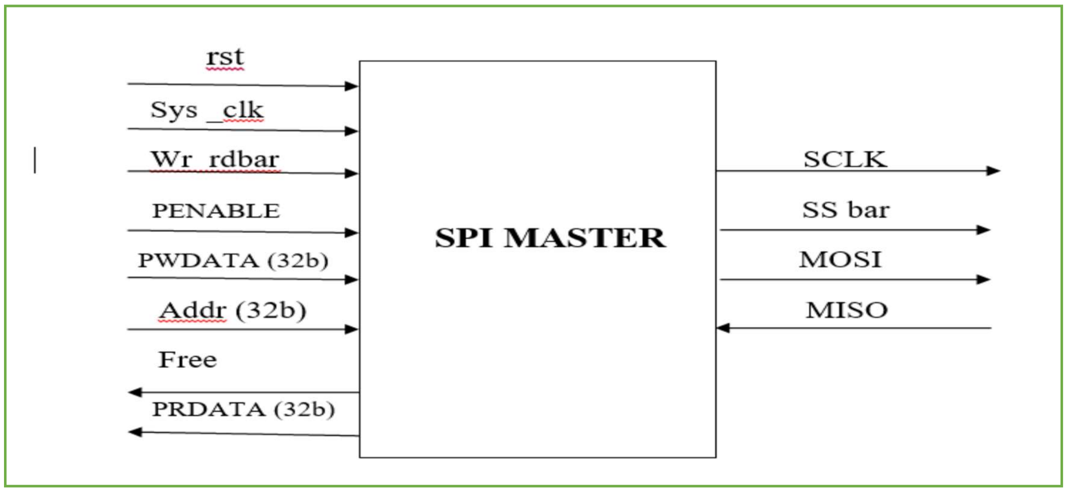
- **Clocking and Control:**
- APB interface is clocked by the same clock as the AXI domain clock (ack).
- The interface has a clock enable, allowing it to be slowed down to execute at an integer divisor of ack.

- **PREADY Signal and Wait States:**
- PREADY signal is used to enable a clean registered interface to the external infrastructure.
- The APB interface adds a wait state for all reads and writes by driving PREADY low.
- Delay of more than one wait state can be generated in specific instances.

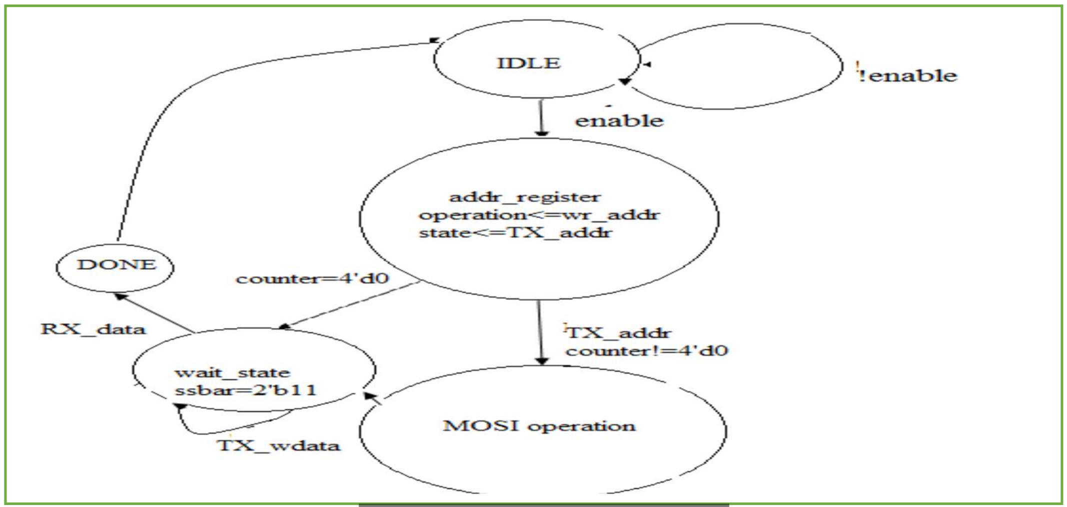
- **PSLVERR Output:**
- PSLVERR output is included for completeness.
- DMC (Dynamic Memory Controller) permanently drives PSLVERR low.

- **State Diagram (Fig) for APB Slave:**
- IDLE is the normal state of the APB.
- When a transfer is necessary, the bus transitions into the SETUP state, where the suitable select signal PSEL is asserted.
- The bus waits in the SETUP state for one clock cycle and moves to the ACCESS state on the next rising edge of the clock.
- ACCESS state is controlled by the PWRITE signal from the slave.
- Conditions to exit the ACCESS state include PWRITE being selected for three commands (command FIFO, write operation, read operation), or if PWRITE is not selected by the slave.
- After exiting ACCESS state, the bus returns to the IDLE state if no more transfers are required, starting the same cycle.

- **APB Slave Block Diagram (Fig):**
- Inputs include PCLK, PRESET, PADDR, PENABLE, PSEL, and REGRDATA.
- Outputs include PEREADY, PSLAVERR, and PRDATA.



APB Master block diagram



State Diagram for APB Master

➢ SPI Bus Overview:

- SPI is a full-duplex serial communication link for short-range communication between devices.
- Originally developed by Motorola, it has become a de facto standard for communication between master and slave devices in embedded systems.
- Widely used in devices such as sensors and SD cards.

➢ SPI Signals:

- SCLK (Serial Clock): Output from the master to the slave, controlling the rate of data transfer.
- MOSI (Master Output Slave Input): Data is sent from the master to the slave.
- MISO (Master Input Slave Output): Data is sent from the slave to the master.
- SS (Slave Select): Signal from the master enabling the slave device for serial data transfer.

➢ SPI Master Block Diagram (Fig):

- IDLE is the normal state of the SPI.
- When a transfer is necessary, the bus transitions into the SETUP state with the ENABLE signal asserted.
- The SETUP state lasts for one clock cycle, then moves to the ACCESS state on the next rising edge of the clock.
- In the ACCESS state, the TX address signal is asserted.

➢ Transition from SETUP to ACCESS State:

- Write, write data signals, select, and address must remain stable during the transition.
- ACCESS state is controlled by the counter signal from the master.

➢ Conditions for Exiting ACCESS State:

- If the counter is held LOW by the master, the peripheral bus remains in the ACCESS state.
- If the counter is driven HIGH by the slave, the ACCESS state is exited, and the bus returns to the IDLE state.
- The bus returns to IDLE if no more transfers are required, starting the same cycle.

➢ SPI Master State Diagram (Fig. 6.1):

- Describes the states of the SPI master, including IDLE, SETUP, and ACCESS states.
- Defines the conditions for state transitions, including the assertion of the ENABLE signal, and control by the counter signal.

Conclusion:

1. The series of SPI controller, SPI master and SPI testbench modules implemented demonstrate a fully functional SPI communication system in Verilog. The controller configures the SPI modes and parameters through register settings. The master module handles the actual data transfers, transmission and reception via shift register based logic controlled by a state machine. The interfaces conform to standard SPI protocols.
2. The testbench module provides stimulus to validate transmission and reception of multiple data packets. It shows the complete working of the configurable SPI modules for different bit orders, clock polarity/phases etc. Thus, this reusable and flexible SPI architecture with parametrized data widths fulfills common communication
3. The configurable SPI controller and SPI master modules developed here showcase common building blocks required for implementing SPI communication protocols. The register interfaces allow easy integration with processors for configuration and data transfer. Modularizing the control and data path allows mixing different SPI modes easily based on application requirements.
4. Overall, these set of modules can enable rapid prototyping of SPI based systems. For example, the SPI master can interface with external SPI flash memory or ADCs/DACs to quickly build a sensor monitoring solution. The flexibility in data width parameter allows interfacing with peripherals of different resolutions easily. The testbench module provides a starting point for comprehensive verification through simulation tests. Both master and slave configurations are possible for end applications by tweaking the control registers.

In conclusion, by following standard SPI interfaces and a modular design approach, this reference implementation covers the key functionality needed in typical embedded applications. Built over this framework, custom SPI centric solutions for sensors, touchscreen, audio, and IoT systems can be rapidly realized. The parameterized structure makes the modules portable across different FPGAs as well. The modular architecture is amenable for future enhancements also discussed in the prior section.