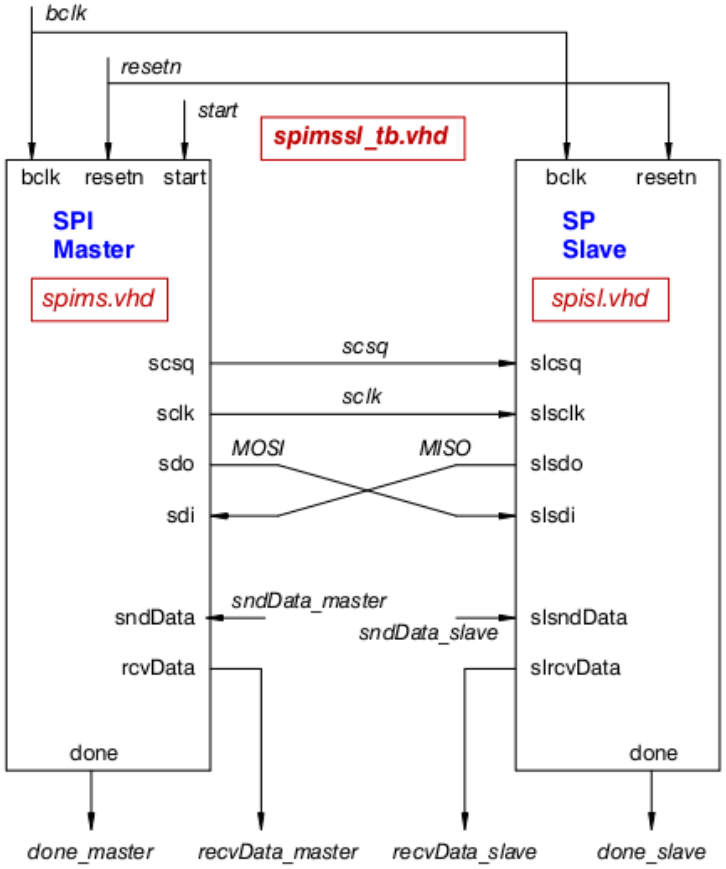
SPI Communication Protocol:

Serial Peripheral Interface (SPI) is a synchronous communication protocol used by many different devices for example, ADC, DAC and RFID modules to communicate with micro-controllers.

Devices communicating via 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.



bclk – bus clock
resetn – active low reset signal
start – initialize the data transmission
SCSQ/SLCSQ – master to select the slave device

SCLK – clock signal
MOSI (Master Output/Slave Input) – master send data to slave

MISO (Master Input/Slave Output) – slave send data to master.

SCSQ/SLCSQ

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

MOSI and MISO

The master sends data to the slave bit by bit, in serial through the MOSI line. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with the MSB first.

The slave can also send data back to the master through the MISO line in serial. The data sent from the slave back to the master is also with the MSB first.

SCLK

SPI communication is always initiated by the master since the master configures and generates the spi clock signal. One bit of data is transferred in each spi clock cycle, so the speed of data transfer is determined by the frequency of the spi clock signal.

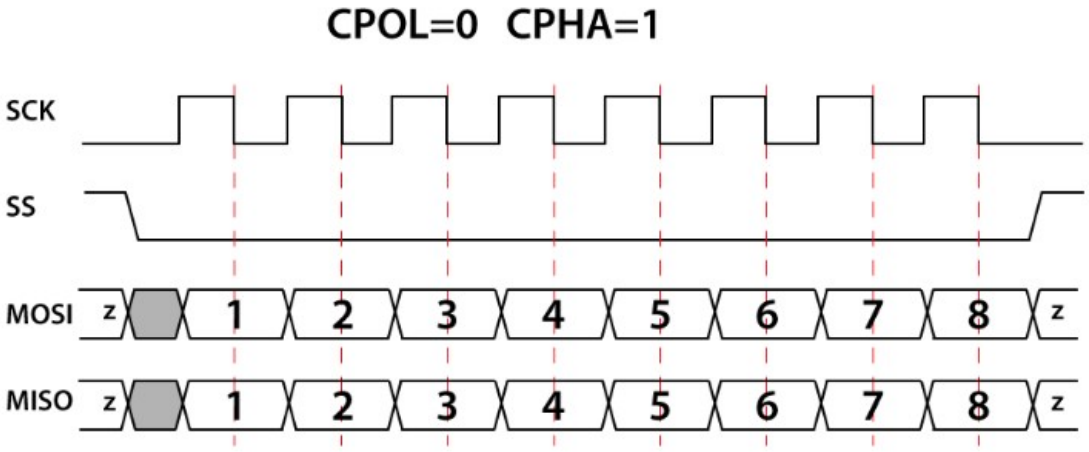
Clock Polarity and Clock Phase

The master configures the clock polarity (CPOL) and clock phase (CPHA) to correspond to slave device requirements. These parameters determine when the data must be stable, when it should be changed according to the clock line and what the clock level is when the clock is not active.

The clock (SCK) signal may be inverted (CPOL=1) or non-inverted (CPOL=0). For the inverted clock signal, the first clock edge is falling. For the non-inverted clock signal, the first clock edge is rising.

The CPHA parameter is used to shift the capturing phase. If CPHA=0, the data are captured on the leading (first) clock edge, regardless of whether that clock edge is rising or falling. If CPHA=1, the data are captured on the trailing (second) clock edge; in this case, the data must be stable for a half cycle before the first clock cycle. Four possible modes are available.

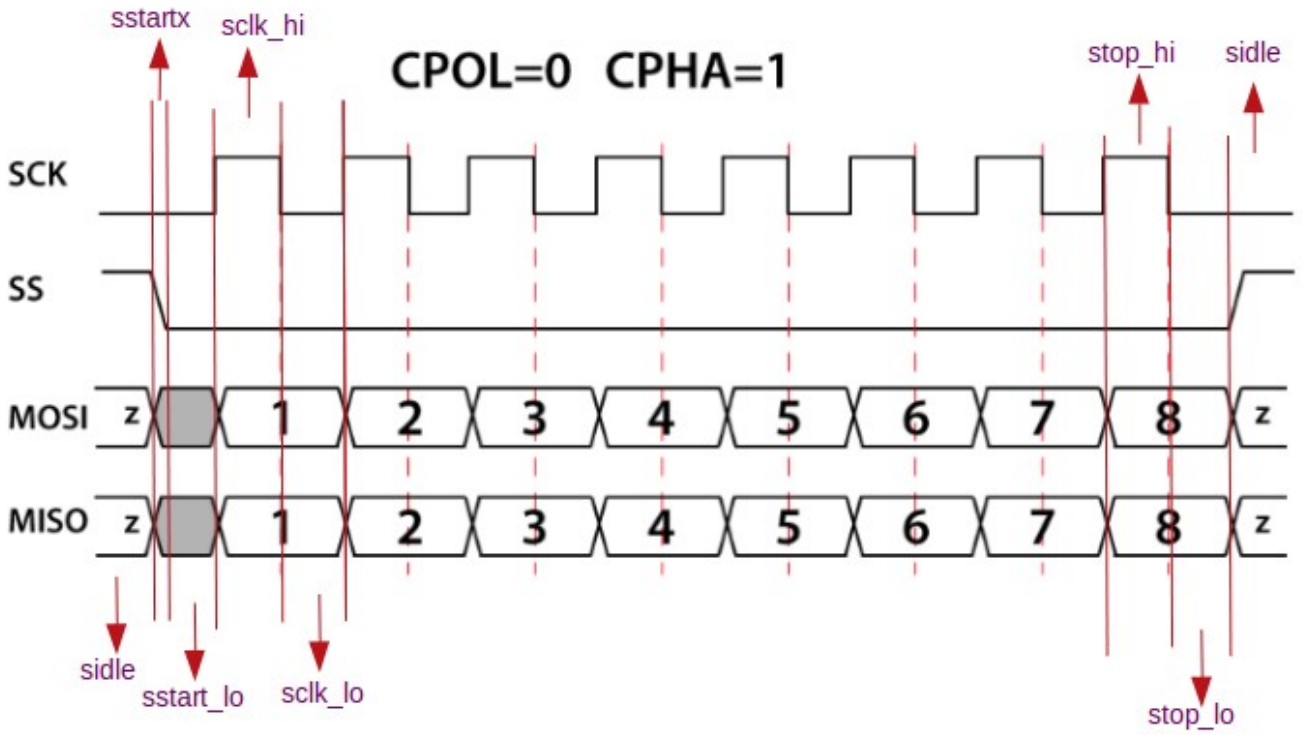
Here we will consider CPOL=0 and CPHA=1. For CPOL=0, the base value of the clock is zero. For CPHA=1, data are captured on the clock's falling edge and data are propagated on the rising edge.



SPI Master:

The SPI master upon getting the start signal, will select the slave to communicate with by lowering the corresponding chip_select (SCSQ) line. Then the master will generate a spi clock signal with number of clock ticks equal to the number of bits to be transferred or read. Since the SPI mode chosen is CPOL=0 and CPHA=1, the master transmits the data bit by bit for every rising edge of the spi clock (sclk) on the MOSI line with MSB first. On every falling edge of sclk, the master reads the data sent by the slave from its MISO line. Once the data is sent, the sclk line becomes 0 and the chip select is made to 1.

Master Algorithm Explanation:



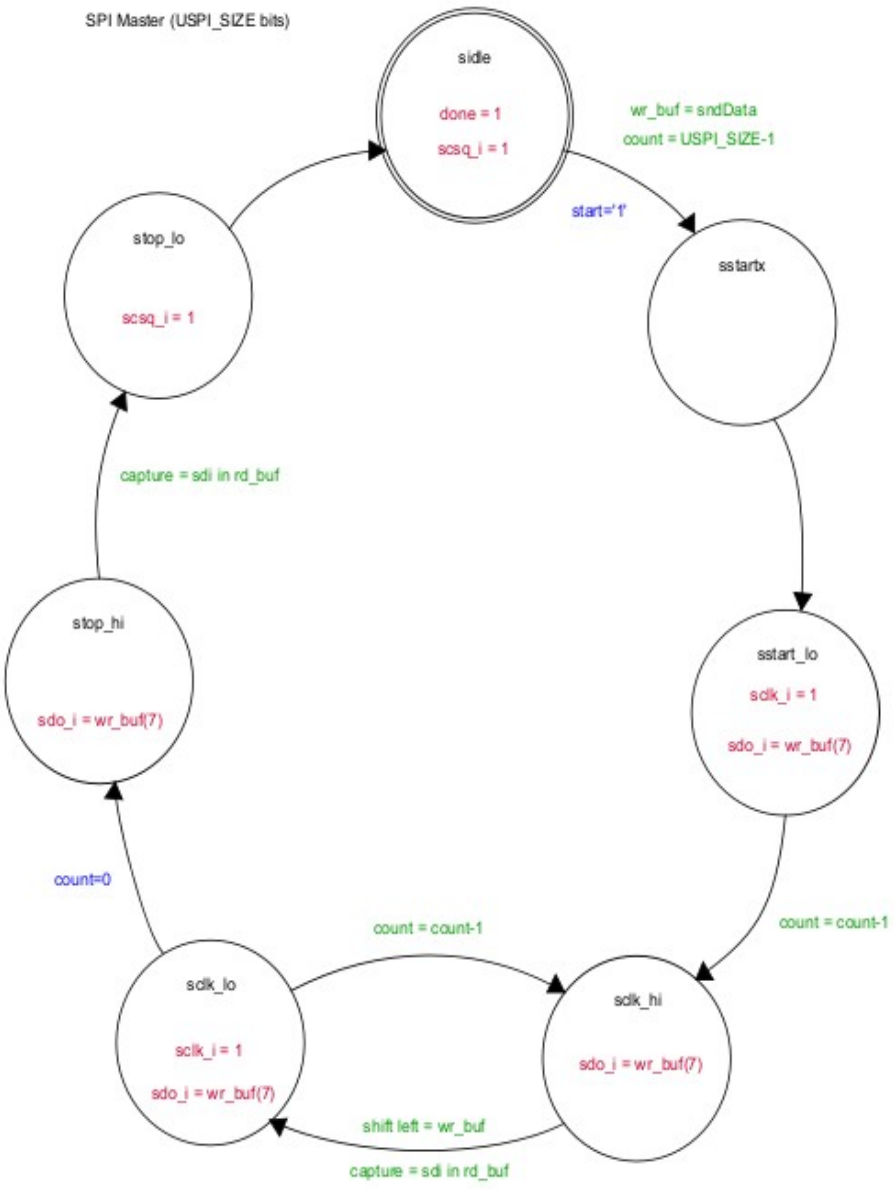
- The intended behavior of our spi logic are well defined as states. The spi master will remain in ***sidle*** state unless the start is triggered (not mentioned in the above figure)
- Then the chip select is brought down by the master in the ***sstartx*** state. Then in the ***sstart_lo*** sclk is made low and the master copies the 1st bit (MSB) into it's write buffer ready to be transmitted in the next state
- In the transition to the next state ***sclk_hi***, the spi transmits its 1st bit (MSB) in the MOSI line and holds the MOSI line with that bit
- In the next state ***sclk_lo***, the master makes the sclk low and during the transition to the state, it reads its MISO line to copy the data sent from slave on to its read buffer. During this time the MOSI line is not updated. The next bit is transferred only on the next rising edge of the sclk.
- On the successive states, the spi toggles between the ***sclk_hi*** and ***sclk_lo*** states unless last but one bit of the data is sent

- The last bit (LSB) is sent during the transition to next ***stop_hi*** state

- In the transition to the ***stop_lo*** state *, the* MISO line is read onto the read buffer. Then the chipselect, SCSQ is made high again and the master goes to ***sidle*** state
- The master makes the done signal to 1 indicating completion of the communication

State Diagram (Master):

STATE MACHINE		
name	spims	
clock	bclk	posedge
implied_loopback	1	
reset_signal	resetN	negedge
reset_state	sidle	
INPUTS		
resetn	1	
bclk		
sndData		[7:0]
OUTPUTS		
done	0	comb
scsq_i	0	comb
sclk_i	0	comb
sdo_i	0	comb
STATES		
done	0	output
scsq_i	0	output
sclk_i	0	output
sdo_i	0	output
TRANSITIONS		
equation	1	def_type
wr_buf		def_type
capture		def_type
count		
shift left		



In the state diagram, the blue colored are conditions which has to be satisfied to make the transition and the green colored statements are actions to be performed during the transition. The statements inside the states are combinational whereas the ones in green are sequential. Since the values of the buffers and count are declared as sequential so that they get created as flip-flops making their signals stable between the rising edges of clock.

The transition of states does not happen in the bus clock frequency (bclk) as it is too high for the slave devices to interact. The spi clock is generated (sclk) which is slower than the bus clock. In our case, we have divided the spi clock for every 3rd rising edge of bclk and hence 1 period of sclk = 6 period of bclk.

VHDL Code Explanation (Master)

Generic and Port Declaration:

The USPI_SIZE and max_clk_div are declared as a generic parameter. USPI_SIZE dentoes the number of bits to be transmitted (in our code it is taken as 8) and the max_clk_div denotes the spi clock frequency in terms of factor of the bus clock (in our code it is taken as 3). The states will transfer only for every 3rd bclk pulse making the spi clock frequency will be 1/6 th of the bus clock which will be the data transmission speed.
The input ports and the outputs are declared in the entity section. The sndData and rcvData are the lines where the data to be sent by the master and data received by the master will be stored.

Combinational Process (evaluate next state):

The combinational process is declared to evaluate the next state logic. The sdo_i (internal signal) becomes the MSB of the wr_buf vector whenever the state has low sclk so that, on the next transition the sdo_i signal will be mapped to the output port sdo and and it is ready in the mosi line for the sclk high state. When the count has not reached to 0 (meaning not all bits are transferred) the nx_state loops back to sclk_hi. When all bits are transferred the next states will stop_hi and stop_lo leading to the idle state.

Sequential Process (update the state):

The sequential process is bound to or evaluated only for the rising edge of the bclk signal. The clock_div process generates pulse to divide the bclk. When this pulse becomes 1 and during the rising edge of bclk, the process updates the next state as the present state of the system. At the same time, if the nx_state is where sclk is low (sclk_lo and stop_lo) then the rd_buf will be shifted left and appended with the bit in the miso line. Simultaneously, the wr_buf will be shifted left to make the MSB as the bit to be transferred next.
If the nx_state is where sclk is high (sclk_hi), the count is decremented by 1 to keep the count of number of digits transferred. The initial value of count is USPI_SIZE -1 (in our case 7).
The parallel data to be sent is converted to serial using the buffers and data is transmitted bit by bit in the sdo (***parallel to serial***). The same happens while reading the serial data from sdi pin into the serial data in buffer (***serial to parallel***).

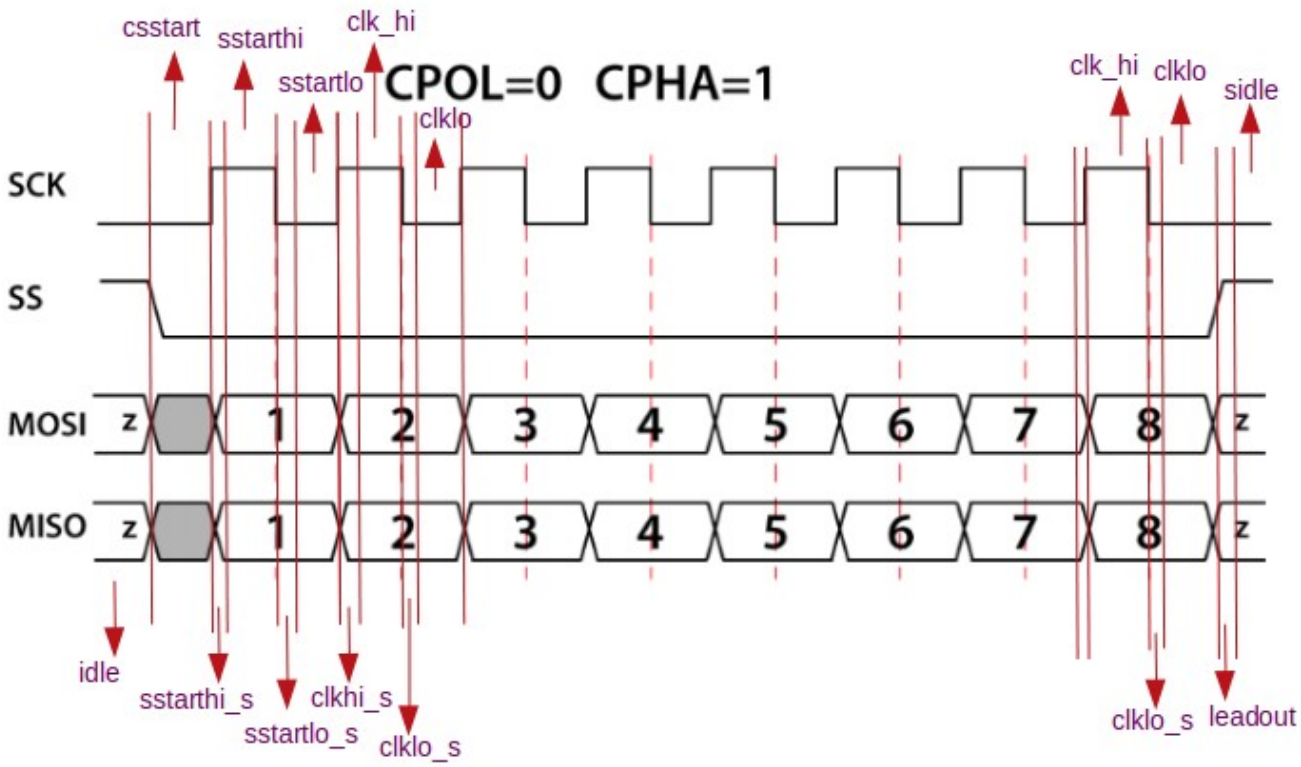
Sequential Process (bus clock divider):

The clock divider process divides the base clock and generates the sclk_p signal. Based on this pulse signal, the sclk is generated and states are updated.

SPI Slave:

The slave devices listens to its chip select (slcsq) line. When the slcsq line becomes low, then the slave understands that it is being selected by the master. The slave starts transmitting data on its serial data out port (miso) on the rising edge of the sclk simultaneously, reading data from its serial data in port (mosi) on every falling edge of sclk. The sclk is generated and transmitted by the master to the slave. The SPI mode chosen is CPOL=0 and CPHA=1

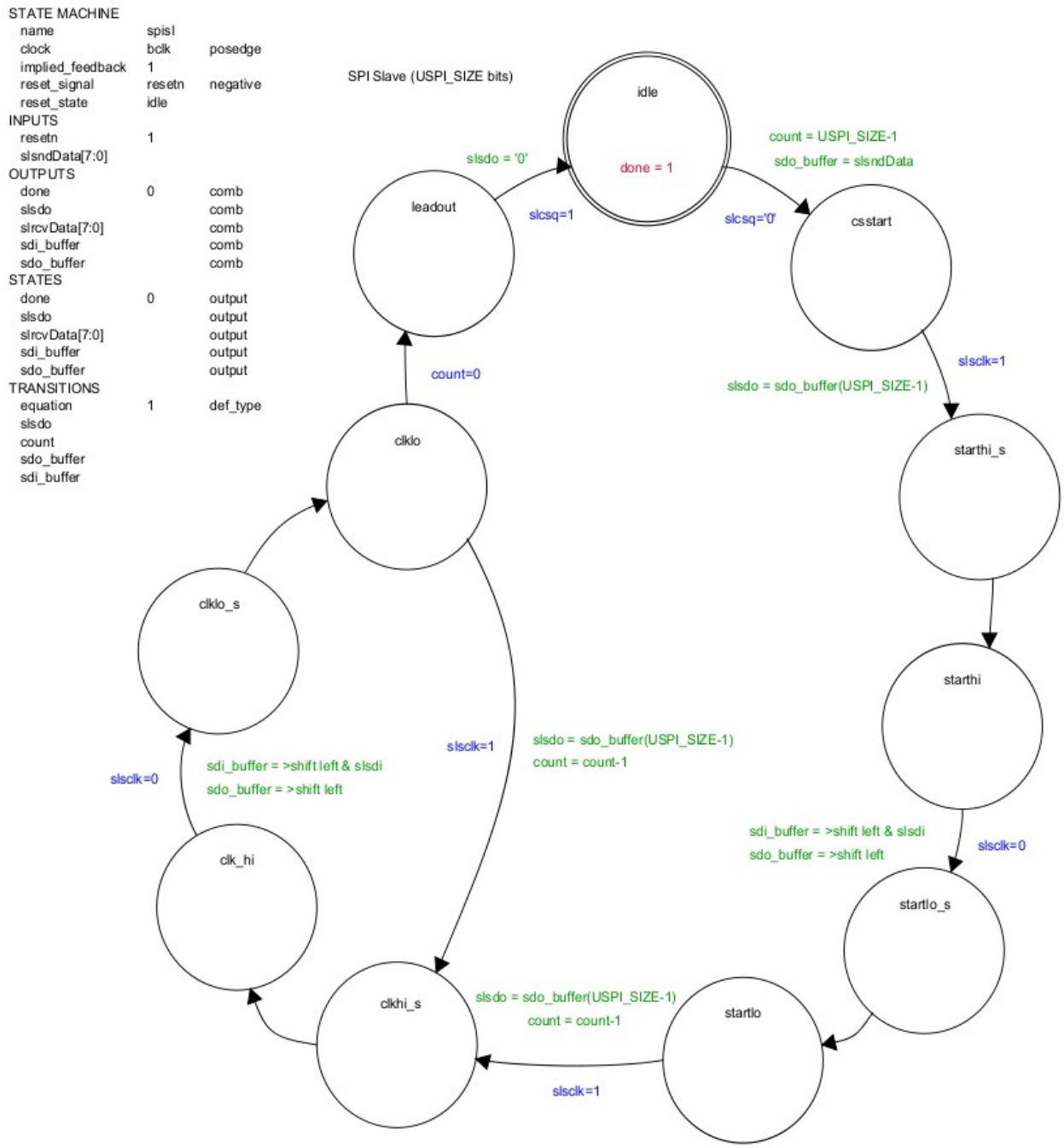
Slave Algorithm Explanation:



- The intended behavior of the spi logic are well defined as the above states. The spi slave will remain in ***idle*** state unless the chip select is made low by the master
- Once the chip select is brought down by the master the slave goes to ***csstart*** state. The slave initiates its sdo buffer with the data to be sent
- When the slclk from the master becomes high, the slave goes to ***starthi_s*** state and during the transition writes its MSB from the sdo buffer to the output port slsdo (miso). The count variable is decremented from the bit size of the data to be transmitted for every rising edge of slsclk
- The next state will be ***starthi*** where the data out line is maintained with the transmitted bit
- When the sclk from the master becomes low, the slave goes to ***startlo_s*** state and during the transition, the read buffer is bit shifted to left and the serial data input bit (mosi) is appended as the LSB. Simultaneously, the sdo_buffer is bit shifted left so that the next bit to be transmitted is made the MSB of the buffer
- In The next state ***startlo*** the serial output lines are held as same
- The states loop henceforth till all the bits are transmitted between ***clk_hi_s , clk_hi, clklo_s*** and ***clklo.*** The sdo buffer and sdi buffer is bit shifted and sdi buffer is appended with mosi bit for every falling edge of sclk. The slave data out (miso) line is transmitted with data for every rising edge of sclk.

- When all the bits are transmitted the count becomes 0 and the loop breaks. The next state becomes ***leadout***. Once the chip select by the master is made to high again, the slave goes back to ***idle*** state.

State Diagram (Slave):



VHDL Code Explanation (Slave):

Generic and Port Declaration:

The USPI_SIZE is declared as a generic parameter. USPI_SIZE dentoes the number of bits to be transmitted (in our code it is taken as 8). The input ports and the outputs are declared in the entity section. The slsndData and slrcvData are the lines where the data to be sent by the master and data received by the master will be stored.

Combinational Process (evaluate next state):

The combinational process is declared to evaluate the next state logic. The MSB of sdo_buf (internal signal) is transmitted to the output port (slsdo) whenever the state makes a transition to a state where slsclk is high. When the count has not reached to 0 (meaning not all bits are transferred) the nx_state loops back to clkhi_s. When all bits are transferred the next states will be leadout and then to the idle state.

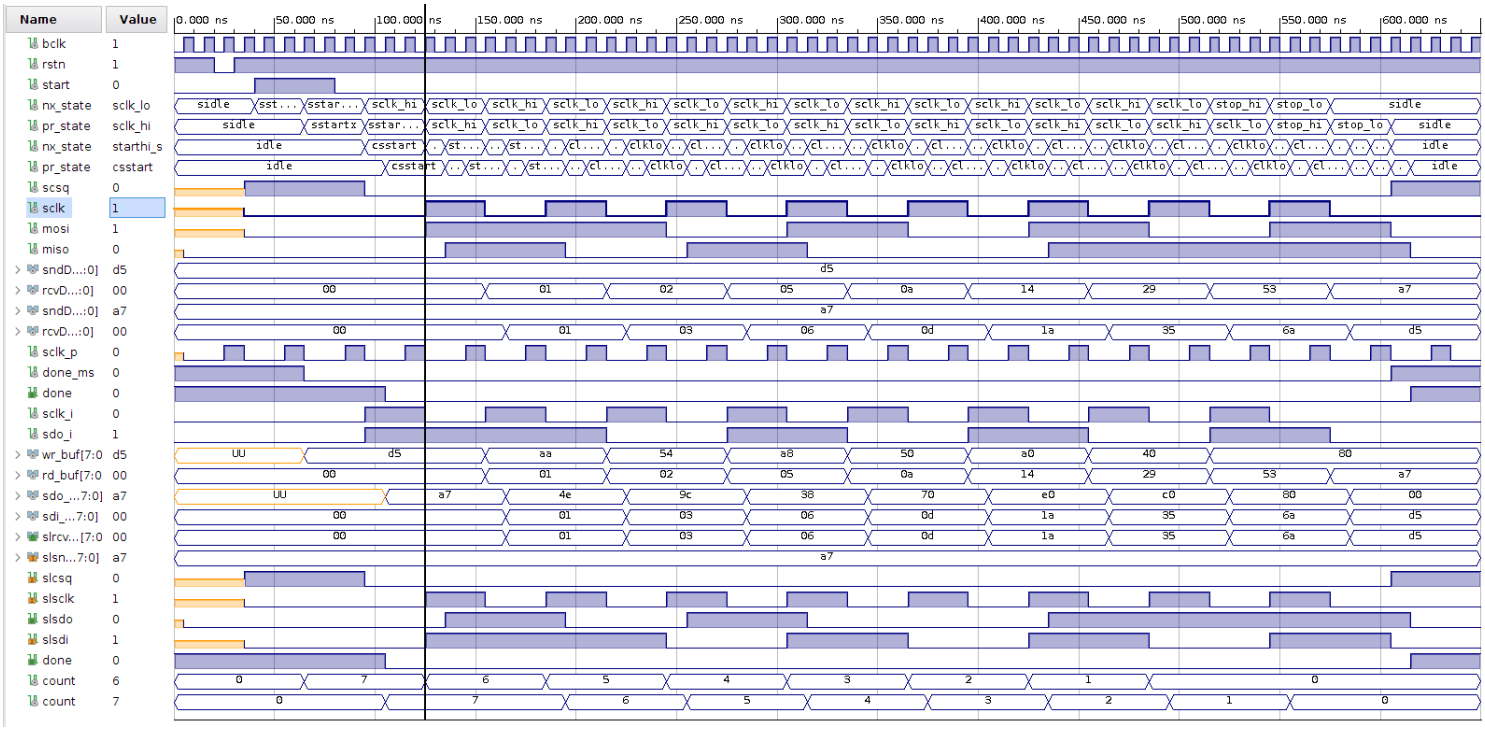
Sequential Process (update the state):

The sequential process is bound to or evaluated only for the rising edge of the bclk signal. The process updates the next state as the present state of the system based on the spi clock from the master.
When the nx_state is where sclk is low (startlo_s and clklo_s) then the sdi_buf will be shifted left and appended with the bit in the mosi line. Simultaneously, the sdo_buf will be shifted left to make the next bit to be transferred as the MSB.
If the nx_state is where sclk is high (clkhi_s), the count is decremented by 1 to keep the count of number of digits transferred. The initial value of count is USPI_SIZE -1 (in our case 7).

VHDL Code Explanation (Testbench):

Two constants are declared in the test bench file ***spi_bits*** and ***spi_clk_div***. The spi_bits represent the number of bits to be transferred and the ***spi_clk_div*** denotes the factor by which the bus clock is to be slowed down.The time period of the spi clock generated will be ***spi_clk_div*** times the bus clock time period.
The master and slave are instantiated in the testbench file as components. The ports of the slave and master are mapped with the signals generated by the testbench. Similarly the generic parameters are mapped as well with the two constants discussed above.
The miso and mosi lines are mapped with the master-slave with their respective data in and data out ports.
The bus clock, resetn are common signals to both the master and slave components.The bus clock is generated with the clock period of 10ns in the clk process.The start signal required to initiate the spi communication is generated by the start_p process.

Simulation Waveform:



- As mentioned in the testbench file, the sndData of master is set with “11010101” (x’d5) and the slsndData is initialized with “10100111” (x’a7)
- Initially the reset signal is made low and both master and slave are in idle states. Then the start signal generated in test bench file becomes high at 40 ns and triggers the master
- The master makes the chip select (scsq) low and generates the sclk signal to transmit the data to the slave device bit by bit for every rising edge of sclk
- The master left bit shifts its internal wr_buf every falling edge of sclk to make the next bit to be transmitted as its MSB
- On every falling edge of the sclk, the master reads its miso line and appends the miso bit as the lsb of its rd_buf after left bit shifting the rd_buf vector signal
- The count decreases for every rising sclk edge. Once all bits are transferred the master makes the scsq high again and stops the sclk, going back to idle state
- The slave checks its ports for every rising edge of the bclk. The slave gets activated once its chip select (slscsq) signal is lowered by the master
- When the sclk from the master is high, the slave recognizes it has to send data on slave data out line (miso) and listen on the slsdi line (mosi). It can be noted that the slave data out line (miso) and slave read buffer (sldi_buffer) updates 1 bclk times slower than the master

- This happens since the slave listens to the sclk line and after the master has made it 1, the slave combinational process recognizes and updates the next state but the state is updated only on the rising edge of bclk. Hence on the next rising edge of the bclk, the slave updates its state and starts transmitting the MSB in the MISO line
- Similarly, the slave has to read on the falling edge of sclk. This is registered as the next state in the combinational process immediately after sclk becoming 0 but the present state is updated in the sequential process only for the rising edge of bclk. Hence, on the next rising edge of bclk, the slave left bit shifts the slsdi buffer and appends the MOSI bit as the LSB
- The sclk_p is generated to slow down the sclk. It generates a pulse for very 3^{rd} bclk rising edge. The states of the master changes only when the sclk_p is high during a rising edge of bclk