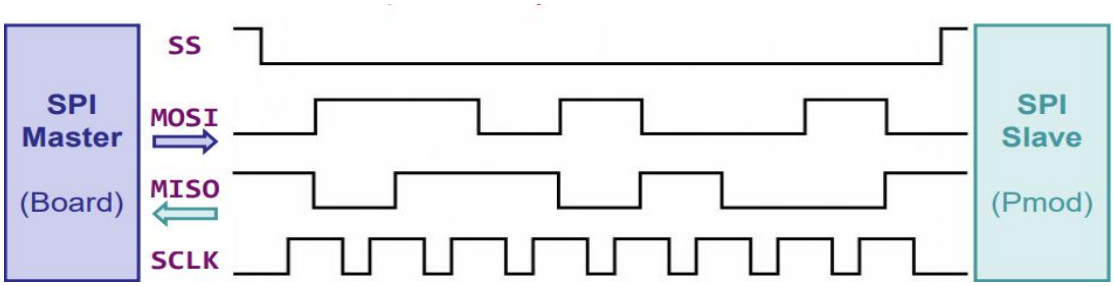
Implement SPI Controller on Xilinx Zedboard :

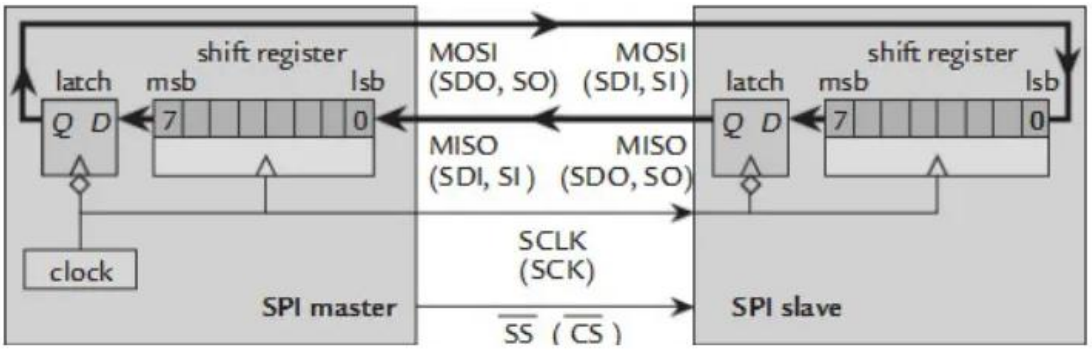
Under this assignment we will start Implementing SPI controller code in Zynq PL part, it will supports SPI write operation only. Which can used for managing the OLED display on the Zed-board. On which we can show the required alphanumeric data.

SPI :

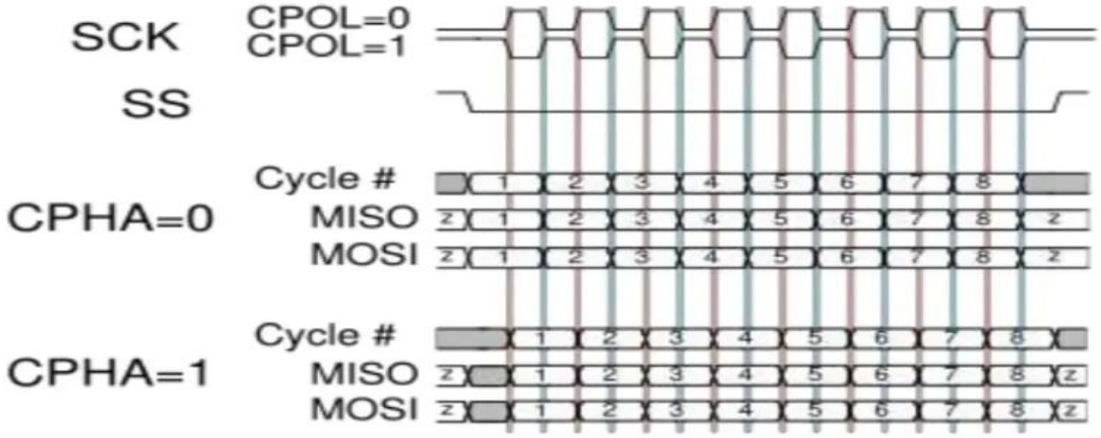
It is serial peripheral communication which uses four wires SCLK, slave select, MOSI, MISO.
It is synchronous serial full duplex communication protocol.
Master Will have control over the clock.
SPI has four modes of operation based on Clock Polarity (CPOL) & Clock Phase(CPHSE).



Below diagram to show in the detail All Pin involved & how the communication will complete bit by bit.
SCLK it provides clock to slave & it is controlled by master which remain in IDLE state when no operation is carried out.
CS this is chip select pin it selects the slave which master wants to transfer the data. I our case we have this pin as ground on zedboard so only one slave can be active.
MOSI is master out slave in pin we will be using this only in our case
MISO is master in slave out as we have configured in in one direction we will be using to write master to slave only which is display. Slave will not have anything to share with master.

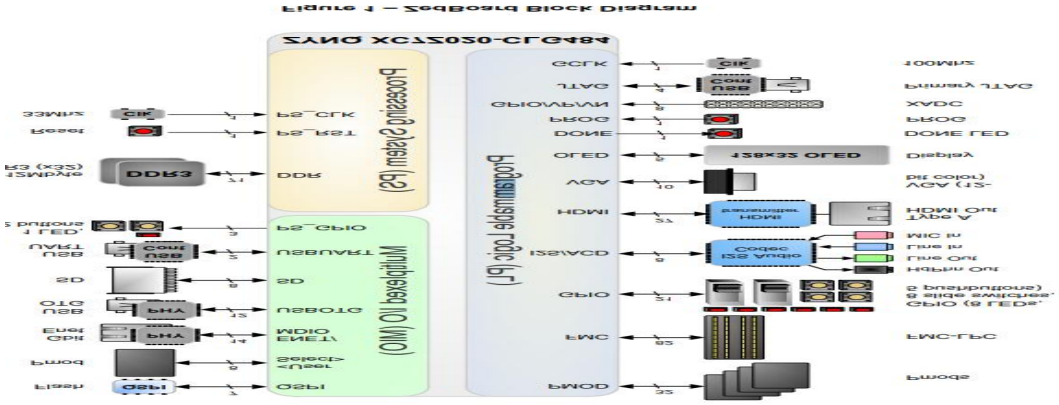


Below diagram is timing diagram for SPI it has different operation modes based on CPOL & CPHSE. Usually these will be given as register bits to configure, available for user to set based on the value of these 2 bits SPI will work in 4 different modes.
basicaly CPOL & CPHSE value sets either data on SPI data line will shift/sample based on positive edge of clock or on Negative edge of clock.



Xilinx Zedboard :

Zedboard is low cost development board for the Xilinx Zynq-7000 all programmable Soc(Ap Soc).it has tightly coupled Arm Processing System(PS) & 7-series programmable logic(PL) to create unique powerful design with Zedboard.
It Provides platform to experiment with zynq Soc which allows users to create embedded systems that leverage processing power of ARM Cores & the flexibility of FPGA for custom hardware acceleration or Logic implementation.
PS part of Zedboard has on chip clock source of 33.33Mhz & PL part has on board 100MHz clock source.
This board has 128*32 pixel,passive matrix monochrome display.
We will be developing the SPI core on PL part of board to communicate with the display so that we can show any kind of data on it like temperature.



Application :

Here we will be implementing the code logic for SPI control instead of using any existing IP core to communicate with OLED display present on Zedboard.
First part is to implement SPI core logic & confirm the working of SPI on simulation then next creating bitstream & confirming same on hardware
Second part is establish communication with slave devices(eg OLED display to show Alphanumeric characters or some BRAM for to perform write/Read)

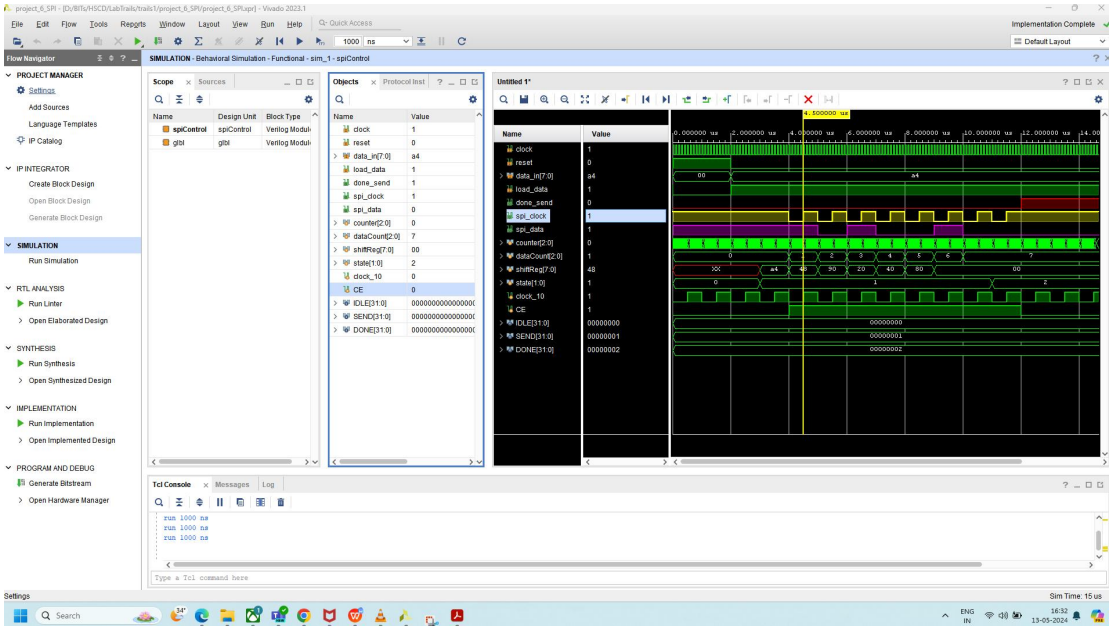
> Code flow:

For SPI application we will divide typically in 3 states of operation
1. IDLE : this is idle state clock line will remain in this state in no operation condition.once we receive some load_data or condition we will set the state as SEND & assign the shift register to data_in & data count to zero.

2. SEND : In this state clock is enabled & SPI_data is updated bit by bit serially till all data from shift register is latched. & then we will be updating the state to DONE.

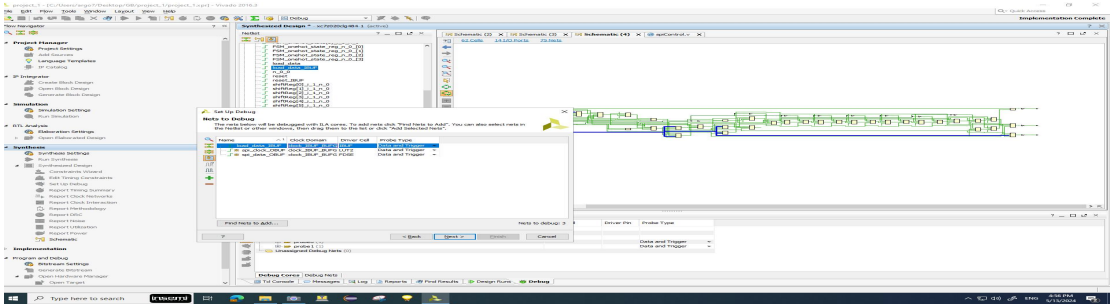
3. DONE : under this state first clock enable is removed. By performing the all necessary updates gain mode is set as IDLE mode.

> Simulation results:
Check the dataIn =0xa4 is written with lien of spi_clock (Yellow)& spi_data(Magenta) in attached snap for simulation result.
See the value of clock enable (CE) value changing which indicates only for the duration of data transfer to slave spi_clock will be active & CE=1 Which will activate based on trigger to start communication.
Load-data pin we can see as trigger in simulation environment.

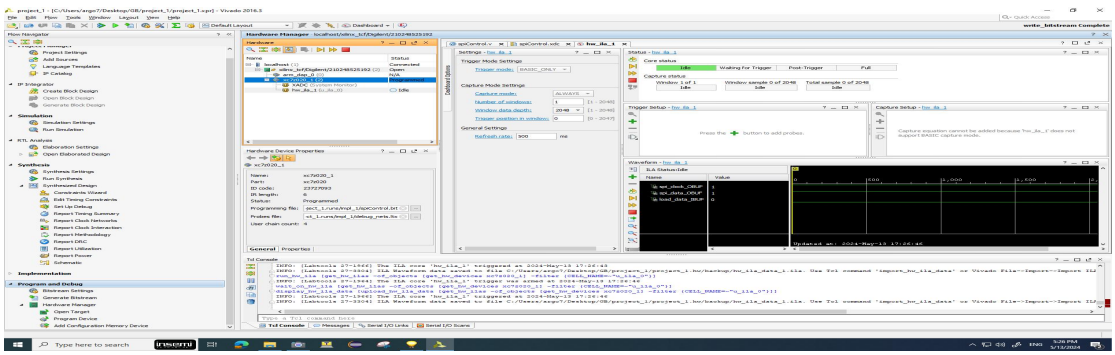


> ILA results :

1. For this we have a after simulation analysis done we don synthesis then we to setup the debugs from the netlist snap for can be seen below



2. All this information will get saved to .xdc file
3. Add I/O planning details of package pins configured for zedboard & can be save d same .xdc file
4. generate bitstream which will create bitsream file (Goes to hardware) & Debug probe files used by ILA for debugging
5. As due to remote hardware I was not able to do any trigger load conditions so we cannot have the data transfer.



Code snippets :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/13/2024 04:43:25 PM
// Design Name:
// Module Name: spiControl
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module spiControl(
input  clock, //On-board Zynq clock (100 MHz)
input  reset,
input [7:0] data_in,
input  load_data, //Signal indicates new data for transmission
output reg done_send,//Signal indicates data has been sent over spi interface
output     spi_clock,//10MHz max
output reg spi_data
);

reg [2:0] counter=0;
reg [2:0] dataCount;
reg [7:0] shiftReg;
reg [1:0] state;
reg clock_10;
reg CE;

assign spi_clock = (CE == 1) ? clock_10 : 1'b1;

always @(posedge clock)
begin
   if(counter != 4)
      counter <= counter + 1;
   else
      counter <= 0;
end

initial
   clock_10 <= 0;

always @(posedge clock)
begin
   if(counter == 4)
      clock_10 <= ~clock_10;
end

localparam IDLE = 'd0,
         SEND = 'd1,
         DONE = 'd2;

always @(negedge clock_10)
begin
   if(reset)
   begin
      state <= IDLE;
      dataCount <= 0;
      done_send <= 1'b0;
      CE <= 0;
      spi_data <= 1'b1;
   end
   else
   begin
      case(state)
         IDLE:begin
            if(load_data)
            begin
               shiftReg <= data_in;
               state <= SEND;
                dataCount <= 0;
            end
         end
         SEND:begin
            spi_data <= shiftReg[7];
            shiftReg <= {shiftReg[6:0],1'b0};
            CE <= 1;
            if(dataCount != 7)
               dataCount <= dataCount + 1;
            else
            begin
               state <= DONE;
            end
         end
         DONE:begin
            CE <= 0;
            done_send <= 1'b1;
            if(!load_data)
            begin
               done_send <= 1'b0;
               state <= IDLE;
            end
         end
      endcase
   end
end


Endmodule
```

Other points :

Block RAM :

It is type of memory specifically designed for FPGA's its also knows as embedded block ram(EBR), It provides high density RAM embedded directly inside the FPGA fabric. This allows efficient storage of data & quick access.

FPGA use BRAM for storing LUTs for logic functions, creating FIFOs buffers for data tarnsferbetween different parts of FPGA ,For implementing dual port memory simultaneous read & write operations. Which provides benefits of speed integration & flexibility.

However it provide speed benefits,but it can be more expensive than external memory. Which make them limited availability for use so careful planning & resource management is requirement.

Attributes :

Keep & DONT TOUCH Attribute : can be used in HDLS during logic synthesis process of FPGAs, synthesis tools performs optimizations to reduce the complexity of logic deisgn. Like merging he logic gates removing unnecessary signals. This attribute tell the tools to keep the specified signal intact even if it is redundant for tool.

1. Preserving for debugging or simulation
2. Floor planning constraints
3. Preserving Hierarchy

I have used "keep" to not modify the name of one the clock signal forILA debugging in the above application.

Integrated Logic Analyser:

ILA core is valuable debugging tool provided with Vivado ip cores, It allows you to monitor & analyse the internal signals of your FPGA design. Also we can add VIO(virtual IO) as trigger to this ILA to see to debug the behaviour like on actual hardware.

Processing system integration :

Zedboard has ARM core processing system & Zynq SOC as PL for tasks the code we have written for SPI will be part of PL logic we will be interacting with PS part for different reasons.