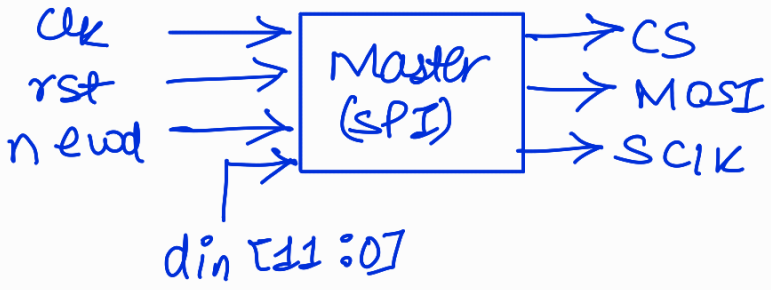
NAME :- RITUSREE MAHAPATRA

→ SPI Design specifications :-



- Here input ports clk, rst, newd, din [11:0] & output ports CS, MOSI, sclk
- Global signals: clk, rst.
- "newd" referred as new data signal wiz used to indicate when user have new data which it need to transmit to the device through an SPI transaction.

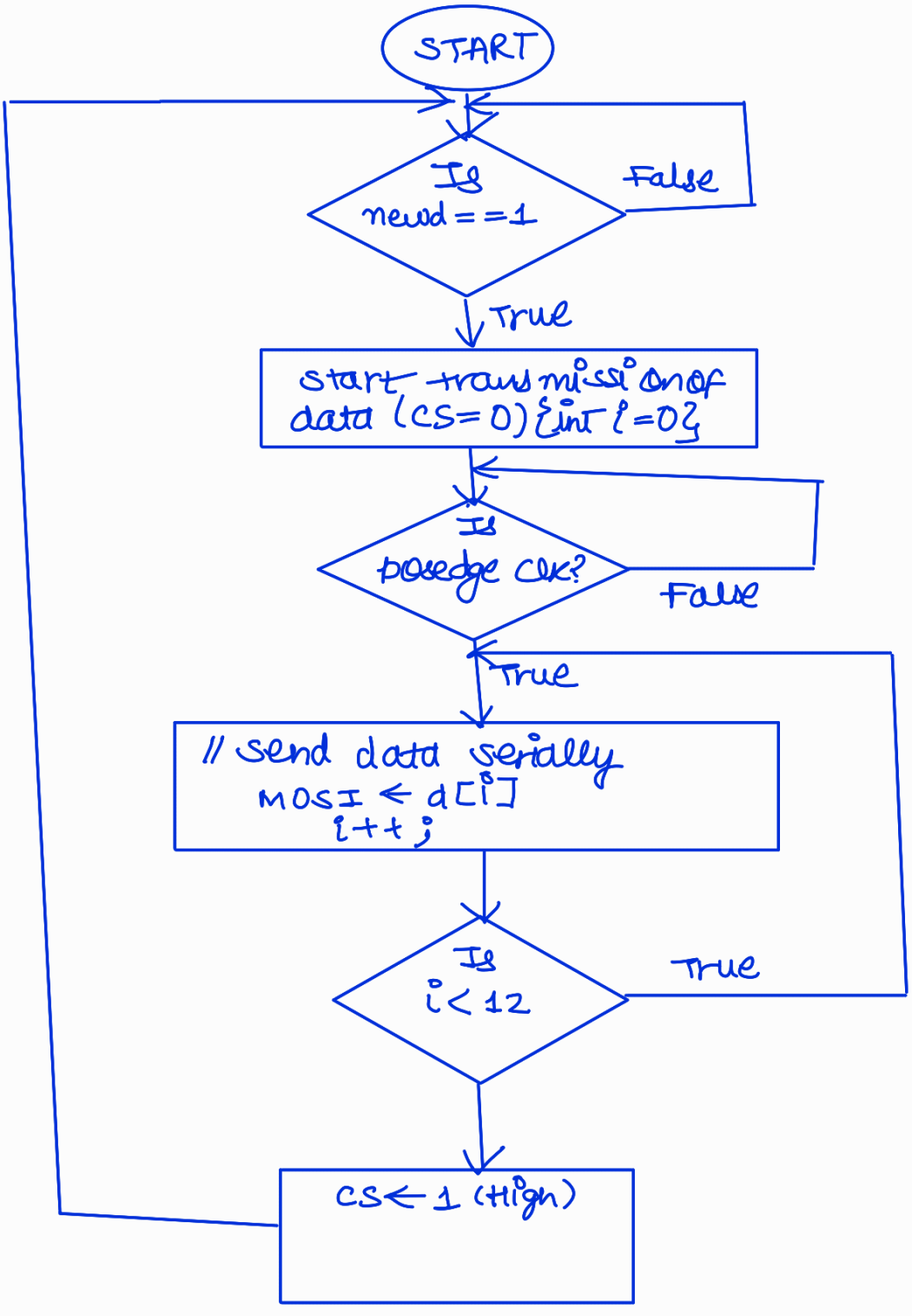
So, as soon we have new data our device will take the data from din [11:0] bus & generate the respective transaction on an output bus.

- "CS" is used to Enable our slave device. an active low on this pin will start a transaction.
- "MOSI" is the pin used to transmit the data serially from master to slave & "sclk" is the serial clock which will be going to slave for synchronization.

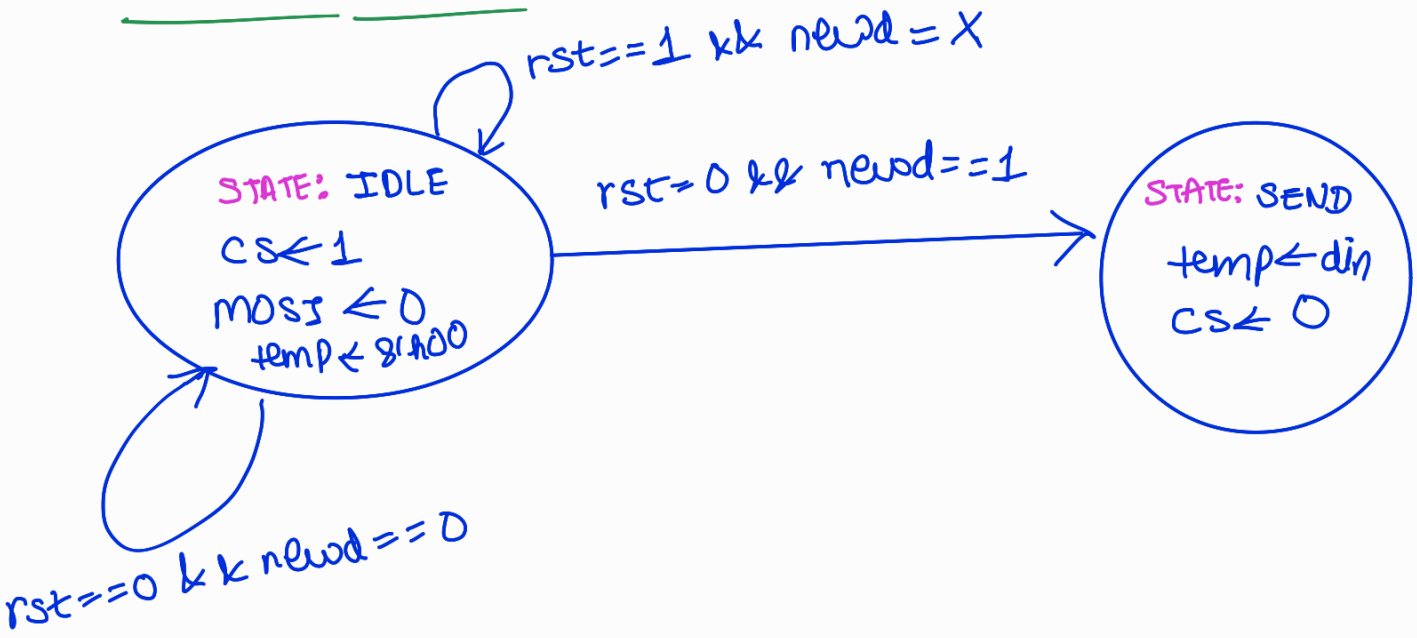
→ OPERATION OF THIS DEVICE :

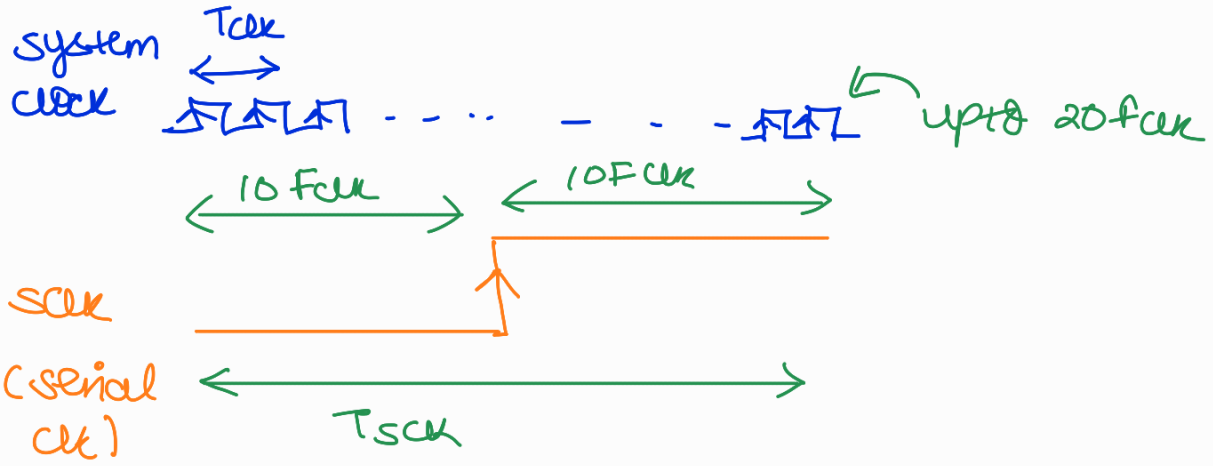
- we will wait till user have a new data AS soon as user make "newd" signal high we will be sampling the data that we have on an din bus & start transmitting it to the slave device. IDLE value for cs is 1 (high). So when user wish to start a transaction, it conveys to the slave by making cs 0 (low). This is what is mean by starting a transaction. & from next clock tick onwards we start sending data serially one after another we wait till all the bits are sent serially as soon as we complete sending all the bits we will be ending our transaction by making cs high

FLOWCHART:—


START
Is
newd == 1
False
True
start transmission of
data (CS=0) {int i=0;
Is
posedge clk?
False
True
// send data serially
MOSI ← d[i]
i++;
Is
i < 12
True
CS ← 1 (High)

STATE DIAGRAM:





$T_{SCLK} = 20\ T_{CLK}$

$\frac{1}{F_{SCLK}} = \frac{20}{F_{CLK}}$

$\Rightarrow \boxed{F_{SCLK} = \frac{f_{CLK}}{20}}$

T_{SCLK} = Time period of serial clock

T_{CLK} = Time period of global clock signal

F_{CLK} = frequency of global clock

F_{SCLK} = frequency of serial clock

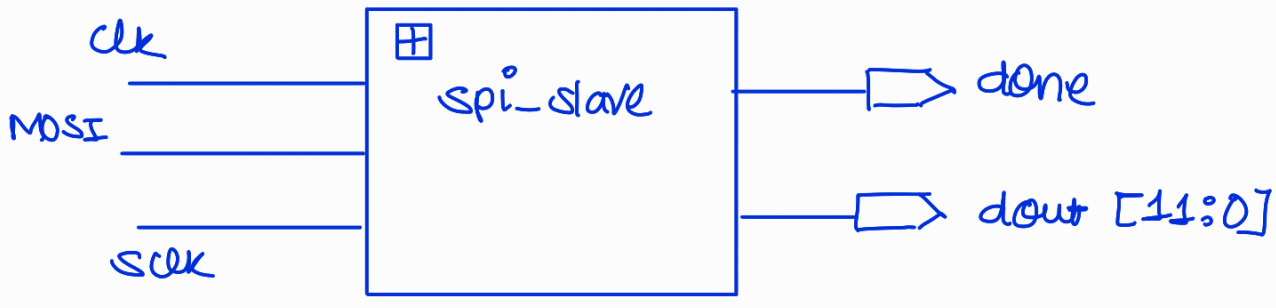
→ SPI Master Design code :

```
module spi (
input clk, newd, rst,
input [11:0] din,
output reg sclk, cs, mosi
);

typedef enum bit [1:0] {idle = 2'b00, enable = 2'b01,
send = 2'b10, comp = 2'b11} state_type;

state_type state = idle;
int countc = 0;
int count = 0;

////// generation of sclk
always@ (posedge clk)
begin
  if (rst == 1'b1) begin
    countc <= 0;
    sclk <= 1'b0;
  end
  else begin
    if (countc < 10)      /// fclk/20
      countc <= countc + 1;
    else
    begin
      countc <= 0;
      sclk <= ~sclk;
```

```
    end
  end
end

//////// state machine
reg [11:0] temp;
always @(posedge sclk)
begin
if (rst == 1'b1) begin
cs <= 1'b1;
mosi <= 1'b0;
end
else begin
case(state)
idle:
  begin
    if (newd == 1'b1) begin
    state <= send;
    temp <= din;
     cs <= 1'b0;
   end
  else begin
  state <= idle;
   temp <= 8'h00;
  end
 end

send: begin
if(count <= 11) begin
mosi <= temp [count]; // sending lsb first
count <= count + 1;
end
else
begin
```

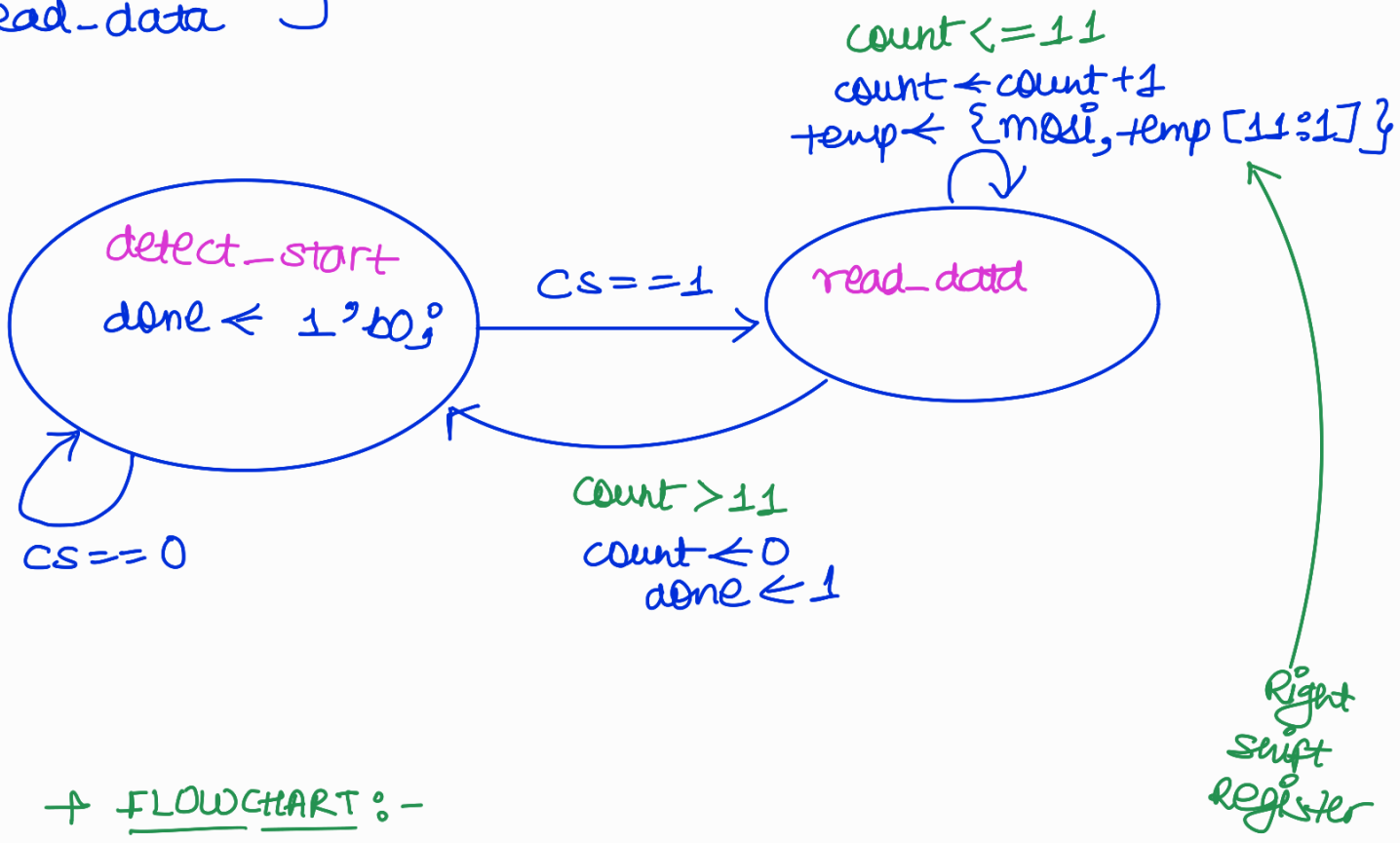
```
      count <= 0;
      state <= idle;
       cs <= 1'b1;
       mosi <= 1'b0;

    end
  end

  default : state <= idle;

  endcase
  end
  end
endmodule
////////////////
interface spi_if;
  logic clk;
  logic newd;
  logic rst;
  logic [11:0] din;
  logic sclk;
  logic cs;
  logic mosi;
endinterface
```

→ SPI slave :-

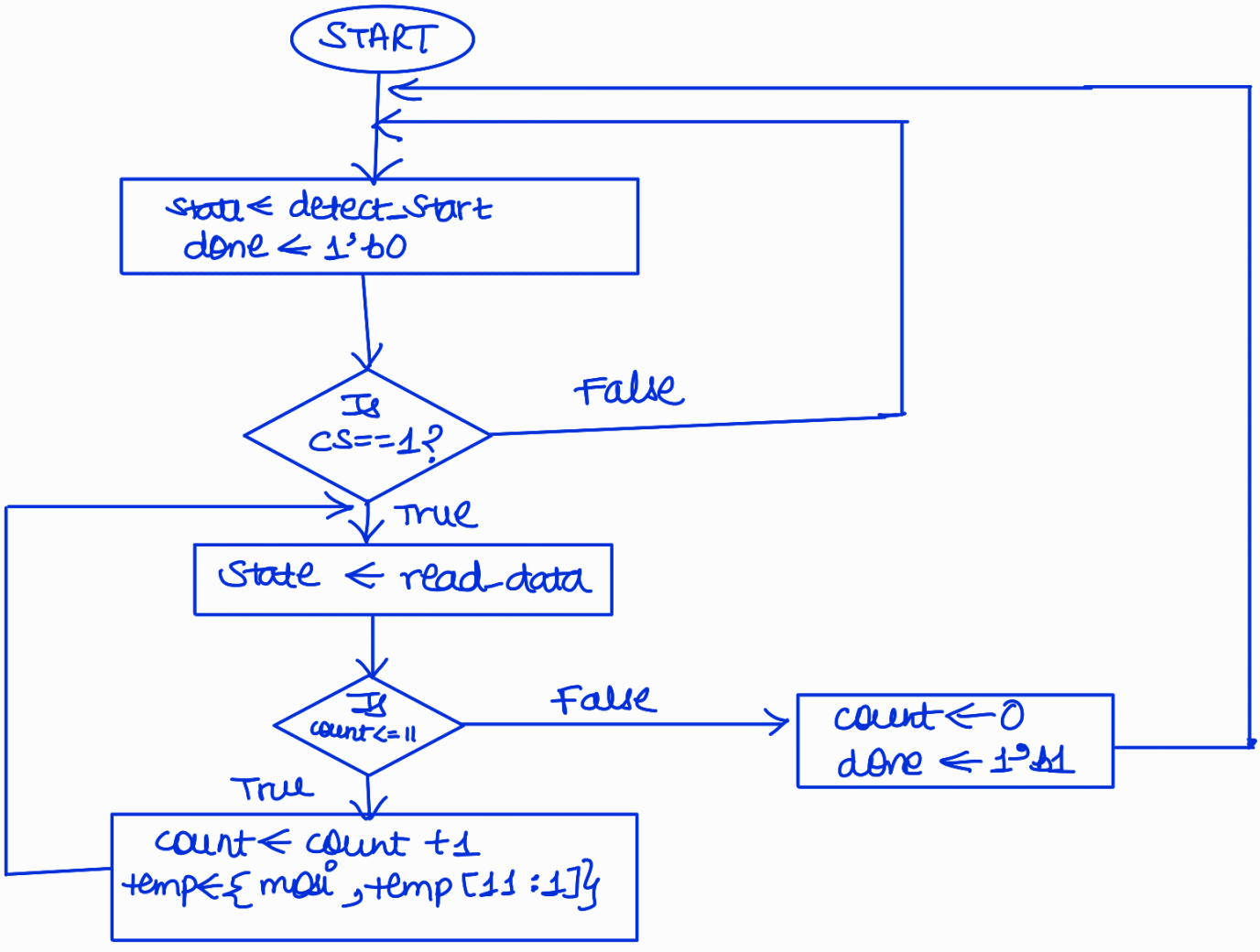


STATE DIAGRAM :

detect_start, read_data } Two states



FLOWCHART :-



→ DESIGN CODE for complete SPI module (Master + slave):

```
module spi_master (
 input clk , newd , rst,
 input [11:0] din,
 output reg sclk, cs, mosi
   );
 typedef enum bit [1:0] { idle = 2'b00, enable = 2'b01,
          send = 2'b10 , comp = 2'b11 } state_type;
  state_type state = idle;
  int countc = 0;
  int count = 0;
 //////// generation of sclk.
  always @(posedge clk)
   begin
   if (rst == 1'b1) begin
     countc <= 0;
     sclk <= 1'b0;
   end
   else begin
    if ( countc < 10)
      countc <= countc +1;
   else
    begin
    countc <= 0;
     sclk <= ~sclk;
    end
  end
  end
  ////////// state machine
  reg [11:0] temp;
```

```
always@(posedge clk)
begin
if(rst==1'b1) begin
 cs<=1'b1;
 mosi<=1'b0;
 end
 else begin
 case(state)
  idle:
  begin
   if(newd==1'b1) begin
    state<=send;
    temp<=din;
     cs<=  1'b0;
   end
   else begin
   state<=idle;
   temp<=8'h00;
   end
   end
   send: begin
   if(count<=11) begin
    mosi<=temp[count]; ////sending LSB first
    count<=count+1;
    end
    else
        begin
      count<=0;
      state<=idle;
       cs<=1'b1;
      mosi<=1'b0;
```

```
end
end
default : state <= idle ;
endcase
end
end
endmodule.
////////////
module spi_slave (
input sclk, cs, mosi,
output [11:0] dout,
output reg done
);
typedef enum bit {detect_start = 1'b0,
read_data = 1'b1} state_type;
state_type state = detect_start;
reg [11:0] temp = 12'h00;
int count=0;
always @ (posedge sclk)
begin
case (state)
detect_start:
begin
done <= 1'b0;
if (cs == 1'b0)
state <= read_data;
else
state <= detect_start;
```

```
end
read_data : begin
if (count<=11)
begin
count<= count+1;
temp <= {mosi, temp[11:1]}
end
else
begin
count <=0;
done <= 1'b1;
state <= detect_start;
end
end
endcase
end
assign dout = temp;
endmodule.
//////////////////////////////////////

module topc
 input clk, rst, newd,
 input [11:0] din,
 output [11:0] dout,
 output done
 );
 wire sclk, cs, mosi;
 spi_master m1( clk, newd, rst, din,
                sclk, cs, mosi);
```

```
spi_slave s1 ( sclk , cs, mosi, dout,
done);

endmodule.
```