Name: Junzhe Wu Net id: Junzhew3

1. SPI_Transmit.v:

```
`timescale 1ns / 1ps

module SPI_Transmit(
    input   wire    [4:0] okUH,
    output  wire    [2:0] okHU,
    inout   wire    [31:0] okUHU,
    inout   wire    okAA,
    input [3:0] button,
    output [7:0] led,
    input sys_clkn,
    input sys_clkp,
    output CVM300_SPI_CLK,
    output CVM300_SPI_EN,
    output  CVM300_SPI_IN,
    input  CVM300_SPI_OUT,
    output CVM300_CLK_IN
    );

    reg [7:0] LSB;
    reg [7:0] MSB;
    reg [7:0] State;
    reg FSM_Clk_reg;
    reg ILA_Clk_reg;
    reg SPI_CLK;
    reg SPI_EN;
    reg SPI_IN;
    wire SPI_OUT;

    wire okClk;           //These are FrontPanel wires needed to IO
communication
    wire [112:0]    okHE;  //These are FrontPanel wires needed to IO
communication
    wire [64:0]     okEH;  //These are FrontPanel wires needed to IO
communication


    //This is the OK host that allows data to be sent or recived
    okHost hostIF (
        .okUH(okUH),
        .okHU(okHU),
        .okUHU(okUHU),
        .okClk(okClk),
```

```
        .okAA(okAA),
        .okHE(okHE),
        .okEH(okEH)
    );

    localparam  endPt_count = 5;
    wire [endPt_count*65-1:0] okEHx;
    okWireOR # (.N(endPt_count) ) wireOR (okEH, okEHx);

    //Instantiate the ClockGenerator module, where three signals are
generate:
    //High speed CLK signal, Low speed FSM_Clk signal
    wire [23:0] ClkDivThreshold;
    wire FSM_Clk, ILA_Clk;
    ClockGenerator ClockGenerator1 (  .sys_clkn(sys_clkn),
                                      .sys_clkp(sys_clkp),
                                      .ClkDivThreshold(ClkDivThreshold),
                                      .FSM_Clk(FSM_Clk),
                                      .ILA_Clk(ILA_Clk) );

    wire [7:0] A1;
    wire [7:0] A2;

    wire [7:0] D1;
    wire [7:0] D2;

    reg [3:0] button_reg;
    reg error_bit = 1'b1;
    reg  RW = 1'b0;
    reg flag = 1'b0;

    localparam STATE_INIT       = 8'd0;
    assign led[6] = error_bit;
    assign CVM300_SPI_CLK = SPI_CLK;
    assign CVM300_SPI_EN = SPI_EN;
    assign CVM300_SPI_IN = SPI_IN;
    assign SPI_OUT = CVM300_SPI_OUT;
    assign CVM300_CLK_IN = ILA_Clk;

    initial  begin
        SPI_CLK = 1'b0;
        SPI_EN = 1'b0;
        SPI_IN = 1'b0;
        State = 8'd0;
```

```
        MSB= 8'd0;
        LSB = 8'd0;
    end

    always @(*) begin
        button_reg = ~button;
        FSM_Clk_reg = FSM_Clk;
        ILA_Clk_reg = ILA_Clk;
    end

    always @(posedge FSM_Clk) begin
        case (State)
            // Press Button[3] to start the state machine. Otherwise,
stay in the STATE_INIT state
            STATE_INIT : begin
                if (button_reg[3] == 1'b1) begin
                  State <= 8'd1;
                  RW <= 1; //write
                end
                else if (button_reg[2] == 1'b1) begin
                    State <= 8'd1;
                    RW <= 0;  //read
                 end
                else  begin
                      SPI_EN <= 1'b0;
                      SPI_CLK <= 1'b0;
                      State <= 8'd0;
                      flag <= 1'b0;
                end
            end

            // write, This is the Start sequence
            8'd1 : begin
                  SPI_EN <= 1'b1;
                  SPI_CLK <= 1'b0;
                  if (RW == 1) SPI_IN <= 1'b1;
                  else SPI_IN <= 1'b0;
                  State <= State + 1'b1;
            end

            8'd2 : begin
                  SPI_CLK <= 1'b1;
                  State <= State + 1'b1;
            end
```

```
        // transmit bit 6
        8'd3 : begin
              SPI_CLK <= 1'b0;
              if (flag==0)
                SPI_IN <= A1[6];
              else
                SPI_IN <= A2[6];
              State <= State + 1'b1;
        end

        8'd4 : begin
              SPI_CLK <= 1'b1;
              State <= State + 1'b1;
        end

        //5
        8'd5 : begin
              SPI_CLK <= 1'b0;
              if (flag == 0)
                SPI_IN <= A1[5];
              else
                SPI_IN <= A2[5];
              State <= State + 1'b1;
        end

        8'd6 : begin
              SPI_CLK <= 1'b1;
              State <= State + 1'b1;
        end

        // transmit bit 4
        8'd7 : begin
              SPI_CLK <= 1'b0;
              if (flag == 0)
                SPI_IN <= A1[4];
              else
                SPI_IN <= A2[4];
              State <= State + 1'b1;
        end

        8'd8 : begin
              SPI_CLK <= 1'b1;
              State <= State + 1'b1;
```

```
        end


        //3
        8'd9 : begin
              SPI_CLK <= 1'b0;
              if (flag == 0)
                SPI_IN <= A1[3];
              else
                SPI_IN <= A2[3];
              State <= State + 1'b1;
        end

        8'd10 : begin
              SPI_CLK <= 1'b1;
              State <= State + 1'b1;
        end

        // transmit bit 2
        8'd11 : begin
              SPI_CLK <= 1'b0;
              if (flag == 0)
                SPI_IN <= A1[2];
              else
                SPI_IN <= A2[2];
              State <= State + 1'b1;
        end

        8'd12 : begin
              SPI_CLK <= 1'b1;
              State <= State + 1'b1;
        end

        //1
        8'd13 : begin
              SPI_CLK <= 1'b0;
              if (flag == 0)
                SPI_IN <= A1[1];
              else
                SPI_IN <= A2[1];
              State <= State + 1'b1;
        end

        8'd14 : begin
```

```
            SPI_CLK <= 1'b1;
            State <= State + 1'b1;
        end

        // transmit bit 0
        8'd15 : begin
            SPI_CLK <= 1'b0;
            if (flag == 0)
              SPI_IN <= A1[0];
            else
              SPI_IN <= A2[0];
            State <= State + 1'b1;
        end


        //write D
        8'd16 : begin
            SPI_CLK <= 1'b1;
            State <= State + 1'b1;
        end

        //7
        8'd17 : begin
            SPI_CLK <= 1'b0;
            SPI_IN <= 1'b0;
            if (RW == 1 && flag == 0)
              SPI_IN <= D1[7];
            else if (RW == 1 && flag == 1)
              SPI_IN <= D2[7];
            State <= State + 1'b1;
        end

        8'd18 : begin
            SPI_CLK <= 1'b1;
            if (RW == 0 && flag == 0)
              MSB[7] <= SPI_OUT;
            else if (RW == 0 && flag == 1)
              LSB[7] <= SPI_OUT;
            State <= State + 1'b1;
        end

        // transmit bit 6
        8'd19 : begin
            SPI_CLK <= 1'b0;
            if (RW == 1 && flag == 0)
```

```
            SPI_IN <= D1[6];
        else if (RW == 1 && flag == 1)
            SPI_IN <= D2[6];
        State <= State + 1'b1;
    end

    8'd20 : begin
        SPI_CLK <= 1'b1;
        if (RW == 0 && flag == 0)
            MSB[6] <= SPI_OUT;
        else if (RW == 0 && flag == 1)
            LSB[6] <= SPI_OUT;
        State <= State + 1'b1;
    end

    //5
    8'd21 : begin
        SPI_CLK <= 1'b0;
        if (RW == 1 && flag == 0)
            SPI_IN <= D1[5];
        else if (RW == 1 && flag == 1)
            SPI_IN <= D2[5];
        State <= State + 1'b1;
    end

    8'd22 : begin
        SPI_CLK <= 1'b1;
        if (RW == 0 && flag == 0)
            MSB[5] <= SPI_OUT;
        else if (RW == 0 && flag == 1)
            LSB[5] <= SPI_OUT;
        State <= State + 1'b1;
    end

    // transmit bit 4
    8'd23 : begin
        SPI_CLK <= 1'b0;
        if (RW == 1 && flag == 0)
            SPI_IN <= D1[4];
        else if (RW == 1 && flag == 1)
            SPI_IN <= D2[4];
        State <= State + 1'b1;
    end
```

```
            8'd24 : begin
                  SPI_CLK <= 1'b1;
                  if (RW == 0 && flag == 0)
                    MSB[4] <= SPI_OUT;
                  else if (RW == 0 && flag == 1)
                    LSB[4] <= SPI_OUT;
                  State <= State + 1'b1;
            end

            //3
            8'd25 : begin
                  SPI_CLK <= 1'b0;
                  if (RW == 1 && flag == 0)
                    SPI_IN <= D1[3];
                  else if (RW == 1 && flag == 1)
                    SPI_IN <= D2[3];
                  State <= State + 1'b1;
            end

            8'd26 : begin
                  SPI_CLK <= 1'b1;
                  if (RW == 0 && flag == 0)
                    MSB[3] <= SPI_OUT;
                  else if (RW == 0 && flag == 1)
                    LSB[3] <= SPI_OUT;
                  State <= State + 1'b1;
            end

            // transmit bit 2
            8'd27 : begin
                  SPI_CLK <= 1'b0;
                  if (RW == 1 && flag == 0)
                    SPI_IN <= D1[2];
                  else if (RW == 1 && flag == 1)
                    SPI_IN <= D2[2];
                  State <= State + 1'b1;
            end

            8'd28 : begin
                  SPI_CLK <= 1'b1;
                  if (RW == 0 && flag == 0)
                    MSB[2] <= SPI_OUT;
                  else if (RW == 0 && flag == 1)
                    LSB[2] <= SPI_OUT;
```

```
            State <= State + 1'b1;
        end

        //1
        8'd29 : begin
            SPI_CLK <= 1'b0;
            if (RW == 1 && flag == 0)
              SPI_IN <= D1[1];
            else if (RW == 1 && flag == 1)
              SPI_IN <= D2[1];
            State <= State + 1'b1;
        end

        8'd30 : begin
            SPI_CLK <= 1'b1;
            if (RW == 0 && flag == 0)
              MSB[1] <= SPI_OUT;
            else if (RW == 0 && flag == 1)
              LSB[1] <= SPI_OUT;
            State <= State + 1'b1;
        end

        // transmit bit 0
        8'd31 : begin
            SPI_CLK <= 1'b0;
            if (RW == 1 && flag == 0)
              SPI_IN <= D1[0];
            else if (RW == 1 && flag == 1)
              SPI_IN <= D2[0];
            State <= State + 1'b1;
        end

        8'd32 : begin
            SPI_CLK <= 1'b1;
            if (RW == 0 && flag == 0)
              MSB[0] <= SPI_OUT;
            else if (RW == 0 && flag == 1)
              LSB[0] <= SPI_OUT;

            if (flag == 1'b0) begin
              State <= 8'd1;
              flag <= 1'b1;
            end
            else if (flag == 1'b1) begin
```

```
                State <= State + 1'b1;
            end
        end

        8'd33 : begin
            SPI_CLK <= 1'b0;
            State <= State + 1'b1;
        end

        8'd34 : begin
             SPI_EN <= 1'b0;
             SPI_IN <= 1'b0;
             flag <= 1'b0;
             State <= 8'd0;
        end


        //If the FSM ends up in this state, there was an error in teh
FSM code
        //LED[6] will be turned on (signal is active low) in that
case.
        default : begin
            error_bit <= 0;
        end
      endcase
  end

  //  The data is communicated via memeory location 0x00
  okWireIn wire10 (   .okHE(okHE),
                      .ep_addr(8'h00),
                      .ep_dataout(D1));

  //  The data is communicated via memeory location 0x01
  okWireIn wire11 (   .okHE(okHE),
                      .ep_addr(8'h01),
                      .ep_dataout(D2));

  okWireIn wire13 (   .okHE(okHE),
                      .ep_addr(8'h02),
                      .ep_dataout(A1));

   okWireIn wire14 (   .okHE(okHE),
                      .ep_addr(8'h03),
                      .ep_dataout(A2));
```

```
    okWireIn wire15 (   .okHE(okHE),
                        .ep_addr(8'h04),
                        .ep_dataout(ClkDivThreshold));

    // result_wire is transmited to the PC via address 0x20
    okWireOut wire20 (  .okHE(okHE),
                        .okEH(okEHx[ 0*65 +: 65 ]),
                        .ep_addr(8'h20),
                        .ep_datain(MSB));

    okWireOut wire21 (  .okHE(okHE),
                        .okEH(okEHx[ 1*65 +: 65 ]),
                        .ep_addr(8'h21),
                        .ep_datain(LSB));
endmodule
```

2. ClockGenerator.v:

```
`timescale 1ns / 1ps

module ClockGenerator(
    input sys_clkn,
    input sys_clkp,
    input [23:0] ClkDivThreshold,
    output reg FSM_Clk,
    output reg ILA_Clk
    );

    //Generate high speed main clock from two differential clock signals
    wire clk;
    reg [23:0] ClkDiv = 24'd0;
    reg [23:0] ClkDivILA = 24'd0;

    IBUFGDS osc_clk(
        .O(clk),
        .I(sys_clkp),
        .IB(sys_clkn)
    );

    // Initialize the two registers used in this module
    initial begin
        FSM_Clk = 1'b0;
        ILA_Clk = 1'b0;
    end
```

```
    // We derive a clock signal that will be used for sampling signals
for the ILA
    // This clock will be 10 times slower than the system clock.
    always @(posedge clk) begin
        if (ClkDivILA == (ClkDivThreshold-1'b1)) begin
            ILA_Clk <= !ILA_Clk;
            ClkDivILA <= 0;
        end else begin
            ClkDivILA <= ClkDivILA + 1'b1;
        end
    end

    // We will derive a clock signal for the finite state machine from
the ILA clock
    // This clock signal will be used to run the finite state machine
for the I2C protocol
    always @(posedge clk) begin
       if (ClkDiv == 10) begin
         FSM_Clk <= !FSM_Clk;
         ClkDiv <= 0;
       end else begin
         ClkDiv <= ClkDiv + 1'b1;
       end
    end
endmodule
```

3. Python code:

```
# import various libraries necessery to run your Python code
import time   # time related library
import sys    # system related library
ok_loc = 'C:\\Program Files\\Opal
Kelly\\FrontPanelUSB\\API\\Python\\3.6\\x64'
sys.path.append(ok_loc)   # add the path of the OK library
import ok     # OpalKelly library

#%%
# Define FrontPanel device variable, open USB communication and
# load the bit file in the FPGA
dev = ok.okCFrontPanel()  # define a device for FrontPanel communication
SerialStatus=dev.OpenBySerial("")      # open USB communicaiton with the
OK board
ConfigStatus=dev.ConfigureFPGA("U:\ece437\Lab_5_2\Lab_5_2.runs\impl_1\S
PI_Transmit.bit"); # Configure the FPGA with this bit file
```

```
# Check if FrontPanel is initialized correctly and if the bit file is
loaded.
# Otherwise terminate the program

print("----------------------------------------------------")
if SerialStatus == 0:
    print ("FrontPanel host interface was successfully initialized.")
else:
    print ("FrontPanel host interface not detected. The error code
number is:" + str(int(SerialStatus)))
    print("Exiting the program.")
    sys.exit ()

if ConfigStatus == 0:
    print ("Your bit file is successfully loaded in the FPGA.")
else:
    print ("Your bit file did not load. The error code number is:" +
str(int(ConfigStatus)))
    print ("Exiting the progam.")
    sys.exit ()
print("----------------------------------------------------")
print("----------------------------------------------------")


D1 = 1
D2 = 1
A1 = 79
A2 = 78
clk_th = 10
dev.SetWireInValue(0x00, D1)
dev.SetWireInValue(0x01, D2)
dev.SetWireInValue(0x02, A1)
dev.SetWireInValue(0x03, A2)
dev.SetWireInValue(0x04, clk_th)
dev.UpdateWireIns()  # Update the WireIns

print("button 2")
# First recieve data from the FPGA by using UpdateWireOuts
while(1):
    dev.UpdateWireOuts()
    result1 = dev.GetWireOutValue(0x20)
    result2 = dev.GetWireOutValue(0x21)
    value = result2+result1*256
    k = 13*(200/(clk_th*2))/25
```

```
    b = 4900*(200/(clk_th*2))/25
    print((value-b)/k)
    #print("address "+str(A1)+" : " + str(result1))
    #print("address "+str(A2)+" : " + str(result2))
    time.sleep(0.1)

dev.Close
```