1. The maximum speed for running SCL signal is 400 kHz.
2. To obtain multiple temperature values from the sensor, I will create a while loop in my python code such that it will send a go signal every second by using time.sleep().
3. The minimum value it can record is -256 Celsius and the maximum value it can record is 255 Celsius. (However, it will not be accurate when the temperature is above 150 Celsius or below -40 Celsius.
4. The resolution of the sensor is 0.0625 Celsius when operating in 13-bits and the resolution is 0.0078 Celsius when operating in 16-bits.

Milestone1:

JTEG\_Test\_File.v:

`timescale 1ns / 1ps

module JTEG\_Test\_File(

output [7:0] led,

input sys\_clkn,

input sys\_clkp,

output ADT7420\_A0,

output ADT7420\_A1,

output I2C\_SCL\_0,

inout I2C\_SDA\_0,

input [4:0] okUH,

output [2:0] okHU,

inout [31:0] okUHU,

inout okAA

);

wire ILA\_Clk, ACK\_bit, FSM\_Clk, TrigerEvent;

wire [23:0] ClkDivThreshold = 1\_000;

wire SCL, SDA;

wire [7:0] State;

wire [31:0] PC\_control;

assign TrigerEvent = PC\_control[0];

//Instantiate the module that we like to test

I2C\_Transmit I2C\_Test1 (

.led(led),

.sys\_clkn(sys\_clkn),

.sys\_clkp(sys\_clkp),

.ADT7420\_A0(ADT7420\_A0),

.ADT7420\_A1(ADT7420\_A1),

.I2C\_SCL\_0(I2C\_SCL\_0),

.I2C\_SDA\_0(I2C\_SDA\_0),

.FSM\_Clk\_reg(FSM\_Clk),

.ILA\_Clk\_reg(ILA\_Clk),

.ACK\_bit(ACK\_bit),

.SCL(SCL),

.SDA(SDA),

.State(State),

.PC\_control(PC\_control),

.okUH(okUH),

.okHU(okHU),

.okUHU(okUHU),

.okAA(okAA)

);

//Instantiate the ILA module

ila\_0 ila\_sample12 (

.clk(ILA\_Clk),

.probe0({State, SDA, SCL, ACK\_bit}),

.probe1({FSM\_Clk, TrigerEvent})

);

endmodule

I2C\_Transmit.v:

`timescale 1ns / 1ps

module I2C\_Transmit(

output [7:0] led,

input sys\_clkn,

input sys\_clkp,

output ADT7420\_A0,

output ADT7420\_A1,

output I2C\_SCL\_0,

inout I2C\_SDA\_0,

output reg FSM\_Clk\_reg,

output reg ILA\_Clk\_reg,

output reg ACK\_bit,

output reg SCL,

output reg SDA,

output reg [7:0] State,

output wire [31:0] PC\_control,

input wire [4:0] okUH,

output wire [2:0] okHU,

inout wire [31:0] okUHU,

inout wire okAA

);

//Instantiate the ClockGenerator module, where three signals are generate:

//High speed CLK signal, Low speed FSM\_Clk signal

wire [23:0] ClkDivThreshold = 100;

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) );

reg [7:0] SingleByteData = 8'b1001\_0001;

reg error\_bit = 1'b1;

localparam STATE\_INIT = 8'd0;

assign led[7] = ACK\_bit;

assign led[6] = error\_bit;

assign ADT7420\_A0 = 1'b0;

assign ADT7420\_A1 = 1'b0;

assign I2C\_SCL\_0 = SCL;

assign I2C\_SDA\_0 = SDA;

initial begin

SCL = 1'b1;

SDA = 1'b1;

ACK\_bit = 1'b1;

State = 8'd0;

end

always @(\*) begin

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 (PC\_control[0] == 1'b1) State <= 8'd1;

else begin

SCL <= 1'b1;

SDA <= 1'b1;

State <= 8'd0;

end

end

// This is the Start sequence

8'd1 : begin

SCL <= 1'b1;

SDA <= 1'b0;

State <= State + 1'b1;

end

8'd2 : begin

SCL <= 1'b0;

SDA <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 7

8'd3 : begin

SCL <= 1'b0;

SDA <= SingleByteData[7];

State <= State + 1'b1;

end

8'd4 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd5 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd6 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 6

8'd7 : begin

SCL <= 1'b0;

SDA <= SingleByteData[6];

State <= State + 1'b1;

end

8'd8 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd9 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd10 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 5

8'd11 : begin

SCL <= 1'b0;

SDA <= SingleByteData[5];

State <= State + 1'b1;

end

8'd12 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd13 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd14 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 4

8'd15 : begin

SCL <= 1'b0;

SDA <= SingleByteData[4];

State <= State + 1'b1;

end

8'd16 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd17 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd18 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 3

8'd19 : begin

SCL <= 1'b0;

SDA <= SingleByteData[3];

State <= State + 1'b1;

end

8'd20 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd21 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd22 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 2

8'd23 : begin

SCL <= 1'b0;

SDA <= SingleByteData[2];

State <= State + 1'b1;

end

8'd24 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd25 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd26 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 1

8'd27 : begin

SCL <= 1'b0;

SDA <= SingleByteData[1];

State <= State + 1'b1;

end

8'd28 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd29 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd30 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// transmit bit 0

8'd31 : begin

SCL <= 1'b0;

SDA <= SingleByteData[0];

State <= State + 1'b1;

end

8'd32 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd33 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd34 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

// read the ACK bit from the sensor and display it on LED[7]

8'd35 : begin

SCL <= 1'b0;

SDA <= 1'bz;

State <= State + 1'b1;

end

8'd36 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd37 : begin

SCL <= 1'b1;

ACK\_bit <= SDA;

State <= State + 1'b1;

end

8'd38 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

//stop bit sequence and go back to STATE\_INIT

8'd39 : begin

SCL <= 1'b0;

SDA <= 1'b0;

State <= State + 1'b1;

end

8'd40 : begin

SCL <= 1'b1;

SDA <= 1'b0;

State <= State + 1'b1;

end

8'd41 : begin

SCL <= 1'b1;

SDA <= 1'b1;

// State <= STATE\_INIT;

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

// OK Interface

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)

);

// PC\_controll is a wire that contains data sent from the PC to FPGA.

// The data is communicated via memeory location 0x00

okWireIn wire10 ( .okHE(okHE),

.ep\_addr(8'h00),

.ep\_dataout(PC\_control));

endmodule

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 == 10) 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 ILA\_Clk) begin

if (ClkDiv == ClkDivThreshold) begin

FSM\_Clk <= !FSM\_Clk;

ClkDiv <= 0;

end else begin

ClkDiv <= ClkDiv + 1'b1;

end

end

endmodule

Python code:

# -\*- coding: utf-8 -\*-

#%%

# import various libraries necessary to run your Python code

import time # time related library

import sys,os # system related library

ok\_sdk\_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\Python\\x64"

ok\_dll\_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\lib\\x64"

sys.path.append(ok\_sdk\_loc) # add the path of the OK library

os.add\_dll\_directory(ok\_dll\_loc)

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 communication with the OK board

# We will NOT load the bit file because it will be loaded using JTAG interface from Vivado

# 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 ()

#%%

# Define the two variables that will send data to the FPGA

# We will use WireIn instructions to send data to the FPGA

PC\_Control = 1; # send a "go" signal to the FSM

dev.SetWireInValue(0x00, PC\_Control)

dev.UpdateWireIns() # Update the WireIns

print("Send GO signal to the FSM")

#%%

# Since we are using a slow clock on the FPGA to compute the results

# we need to wait for the result to be computed

time.sleep(0.5)

PC\_Control = 0; # send a "stop" signal to the FSM

dev.SetWireInValue(0x00, PC\_Control)

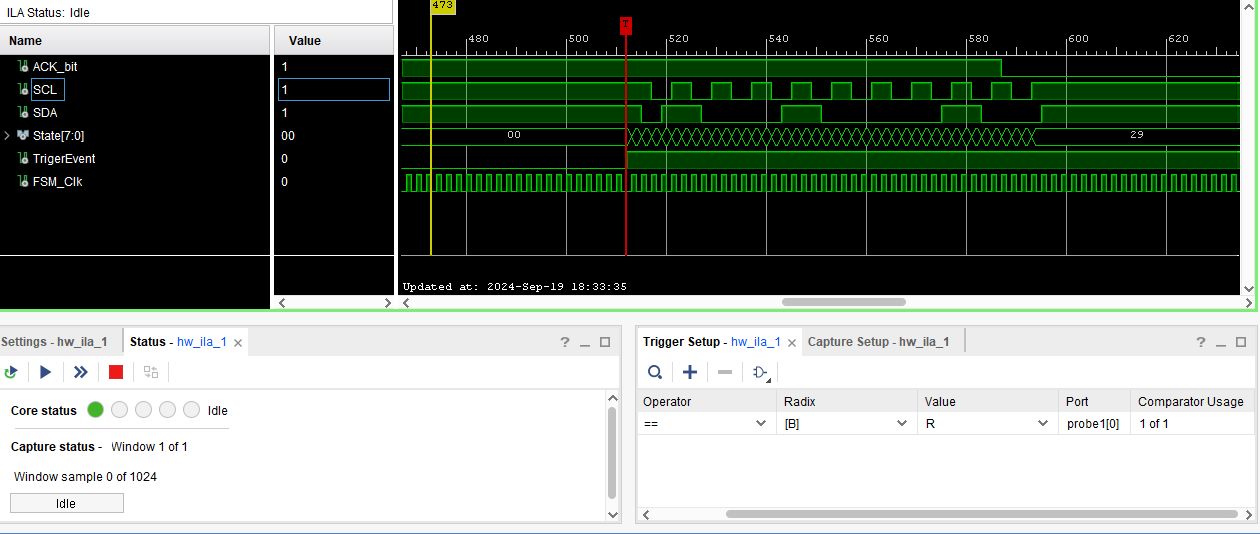
dev.UpdateWireIns() # Update the WireIns

print("Send STOP signal to the FSM")

dev.Close

#%%

Milestone1 waveforms:

A screenshot of a computer

Description automatically generated

Milestone2:

I2C.v:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/09/22 02:36:21

// Design Name:

// Module Name: I2C

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module I2C\_driver(

output [7:0] led,

input clk,

output ADT7420\_A0,

output ADT7420\_A1,

output I2C\_SCL\_0,

inout I2C\_SDA\_0,

output reg ACK,

output reg SCL,

output reg SDA,

output reg [5:0] State,

input wire [7:0] tx\_byte,

output reg [7:0] rx\_byte,

input wire [1:0] next\_step,

output reg ready

);

localparam idle\_ = 6'b000000;

localparam start\_ = 6'b000001;

localparam tx = 6'b000010;

localparam tx\_ack = 6'b000100;

localparam rx = 6'b001000;

localparam rx\_ack = 6'b010000;

localparam end\_ = 6'b100000;

localparam error\_ = 6'b111111;

reg [2:0] bit\_counter;

reg [9:0] clk\_counter;

reg [7:0] rx\_byte\_reg;

reg [7:0] tx\_byte\_reg;

reg error;

assign led[7] = ACK;

assign led[6] = SCL;

assign led[5] = SDA;

assign led[4:0] = {5{error}};

assign I2C\_SCL\_0 = SCL;

assign I2C\_SDA\_0 = SDA;

assign ADT7420\_A0 = 1'b0;

assign ADT7420\_A1 = 1'b0;

initial begin

SCL = 1'b1;

SDA = 1'b1;

ACK = 1'b1;

error = 1'b0;

ready = 1'b1;

State = idle\_;

rx\_byte = 8'b00000000;

rx\_byte\_reg = 0;

tx\_byte\_reg = 0;

end

always @(posedge clk) begin

case (State)

idle\_ : begin

if (next\_step == 2'b01)begin

State <= start\_;

clk\_counter <= 10'd400;

bit\_counter <= 0;

end

end

start\_: begin

case (clk\_counter)

10'd0 : begin

SCL <= 1'b0;

SDA <= 1'bz;

clk\_counter <= clk\_counter + 1;

end

10'd400 : begin

SCL <= 1'b1;

clk\_counter <= clk\_counter + 1;

end

10'd600 : begin

SCL <= 1'b1;

SDA <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

10'd799 : begin

State <= tx;

tx\_byte\_reg <= tx\_byte;

clk\_counter <= 10'd0;

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

tx: begin

case (clk\_counter)

10'd0 : begin

SCL <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

10'd200 : begin

SDA <= tx\_byte\_reg[bit\_counter];

SCL <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

10'd400 : begin

SCL <= 1'b1;

clk\_counter <= clk\_counter + 1;

end

10'd799 : begin

if (bit\_counter == 3'd7) begin

rx\_byte <= rx\_byte\_reg;

State <= tx\_ack;

bit\_counter <= 3'd0;

end

else begin

bit\_counter <= bit\_counter + 1;

end

clk\_counter <= 10'd0;

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

tx\_ack : begin

case (clk\_counter)

10'd0 : begin

SCL <= 1'b0;

SDA <= 1'bz;

clk\_counter <= clk\_counter + 1;

end

10'd400 : begin

SCL <= 1'b1;

ACK <= SDA;

clk\_counter <= clk\_counter + 1;

end

10'd799 : begin

tx\_byte\_reg <= tx\_byte;

clk\_counter <= 10'd0;

case (next\_step)

2'b00: begin

State <= end\_;

end

2'b01: begin

State <= start\_;

end

2'b10: begin

State <= tx;

end

2'b11: begin

State <= rx;

end

endcase

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

rx: begin

case (clk\_counter)

10'd0 : begin

SCL <= 1'b0;

SDA <= 1'bz;

clk\_counter <= clk\_counter + 1;

end

10'd400 : begin

SCL <= 1'b1;

clk\_counter <= clk\_counter + 1;

end

10'd500 : begin

rx\_byte\_reg[bit\_counter] <= SDA;

clk\_counter <= clk\_counter + 1;

end

10'd799 : begin

if (bit\_counter == 3'd7) begin

rx\_byte <= rx\_byte\_reg;

State <= rx\_ack;

bit\_counter <= 3'd0;

end else begin

bit\_counter <= bit\_counter + 1;

end

clk\_counter <= 10'd0;

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

rx\_ack : begin

case (clk\_counter)

10'd0 : begin

SCL <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

10'd200: begin

SDA <= tx\_byte\_reg[0];

clk\_counter <= clk\_counter + 1;

end

10'd400 : begin

SCL <= 1'b1;

clk\_counter <= clk\_counter + 1;

end

10'd799 : begin

clk\_counter <= 10'd0;

tx\_byte\_reg <= tx\_byte;

case (next\_step)

2'b00: begin

State <= end\_;

end

2'b01: begin

State <= start\_;

end

2'b10: begin

State <= tx;

end

2'b11: begin

State <= rx;

end

endcase

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

end\_ : begin

case (clk\_counter)

10'd0: begin

SCL <= 1'b0;

SDA <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

10'd400: begin

SCL <= 1'b1;

SDA <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

10'd600 : begin

SCL <= 1'b1;

SDA <= 1'b1;

clk\_counter <= 10'd0;

State <= idle\_;

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

default : begin

error <= 1'b1;

end

endcase

end

always @(posedge clk) begin

case (State)

tx : ready <= 1'b0;

rx : ready <= 1'b0;

default : ready <=1'b1;

endcase

end

endmodule

Main.v:

`timescale 1ns / 1ps

module Main(

output [7:0] led,

input sys\_clkn,

input sys\_clkp,

output ADT7420\_A0,

output ADT7420\_A1,

output I2C\_SCL\_0,

inout I2C\_SDA\_0,

input [4:0] okUH,

output [2:0] okHU,

inout [31:0] okUHU,

inout okAA

);

// Clock generation//////////////////////////////////////////////////////////////////

reg ILA\_Clk;

wire clk;

reg [23:0] ClkDivILA = 24'd0;

IBUFGDS osc\_clk(

.O(clk),

.I(sys\_clkp),

.IB(sys\_clkn)

);

always @(posedge clk) begin

if (ClkDivILA == 10) begin

ILA\_Clk <= !ILA\_Clk;

ClkDivILA <= 0;

end else begin

ClkDivILA <= ClkDivILA + 1'b1;

end

end

// Clock generation; ///////////////////////////////////////////////////////////////

//PC communication/////////////////////////////////////////////////////////////////

// TODO verify OK communication function

wire [31:0] PC\_rx;

wire [31:0] PC\_tx;

wire [112:0] okHE;

wire [64:0] okEH;

localparam endPt\_count = 2;

wire [endPt\_count\*65-1:0] okEHx;

okWireOR # (.N(endPt\_count)) wireOR (okEH, okEHx);

okHost hostIF (

.okUH(okUH),

.okHU(okHU),

.okUHU(okUHU),

.okClk(okClk),

.okAA(okAA),

.okHE(okHE),

.okEH(okEH)

);

okWireIn wire10 ( .okHE(okHE),

.ep\_addr(8'h00),

.ep\_dataout(PC\_rx));

okWireOut wire20 ( .okHE(okHE),

.okEH(okEHx[ 0\*65 +: 65 ]),

.ep\_addr(8'h20),

.ep\_datain(PC\_tx));

// PC communication////////////////////////////////////////////////////////////////

//I2C SERDES///////////////////////////////////////////////////////////////////////

wire SCL, SDA,ACK;

wire [5:0] State;

wire [7:0] tx\_byte,rx\_byte;

wire [1:0] next\_step;

wire ready;

I2C\_driver I2C\_SERDES (

.led(led),

.clk(clk),

.ADT7420\_A0(ADT7420\_A0),

.ADT7420\_A1(ADT7420\_A1),

.I2C\_SCL\_0(I2C\_SCL\_0),

.I2C\_SDA\_0(I2C\_SDA\_0),

.ACK(ACK),

.SCL(SCL),

.SDA(SDA),

.State(State),

.tx\_byte(tx\_byte),

.rx\_byte(rx\_byte),

.next\_step(next\_step),

.ready(ready)

);

// I2C SERDES ////////////////////////////////////////////////////////////////////////

//Sensor Controller///////////////////////////////////////////////////////////////////

TS\_controller TS\_controller(

.clk(clk),

.PC\_rx(PC\_rx),

.PC\_tx(PC\_tx),

.next\_step(next\_step),

.tx\_byte(tx\_byte),

.rx\_byte(rx\_byte),

.ready(ready)

);

//Sensor Controller/////////////////////////////////////////////////////////////////////

//Instantiate the ILA module

ila\_0 ila\_sample12 (

.clk(clk),

.probe0({State, SDA, SCL, ACK}),

.probe1(next\_step));

endmodule

main\_TB.v:

`timescale 1ns / 1ps

module Main\_TB();

//I2C SERDES///////////////////////////////////////////////////////////////////////

wire SCL, SDA, ACK;

wire [5:0] State;

wire [7:0] tx\_byte,rx\_byte;

wire [1:0] next\_step;

wire ready;

wire ADT7420\_A0;

wire ADT7420\_A1;

wire I2C\_SCL\_0;

wire I2C\_SDA\_0;

reg [31:0] PC\_rx;

wire [31:0] PC\_tx;

reg clk = 1;

I2C\_driver I2C\_SERDES (

.led(led),

.clk(clk),

.ADT7420\_A0(ADT7420\_A0),

.ADT7420\_A1(ADT7420\_A1),

.I2C\_SCL\_0(I2C\_SCL\_0),

.I2C\_SDA\_0(I2C\_SDA\_0),

.ACK(ACK),

.SCL(SCL),

.SDA(SDA),

.State(State),

.tx\_byte(tx\_byte),

.rx\_byte(rx\_byte),

.next\_step(next\_step),

.ready(ready)

);

// I2C SERDES ////////////////////////////////////////////////////////////////////////

//Sensor Controller///////////////////////////////////////////////////////////////////

TS\_controller TS\_controller(

.clk(clk),

.PC\_rx(PC\_rx),

.PC\_tx(PC\_tx),

.next\_step(next\_step),

.tx\_byte(tx\_byte),

.rx\_byte(rx\_byte),

.ready(ready)

);

//Sensor Controller/////////////////////////////////////////////////////////////////////

always begin

#5 clk = ~clk;

end

initial begin

#0 PC\_rx <= 0;

#400 PC\_rx <= 1;

end

endmodule

TS\_controller.v:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/09/22 14:34:42

// Design Name:

// Module Name: TS\_controller

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module TS\_controller(

input clk,

input wire [31:0] PC\_rx,

output reg [31:0] PC\_tx,

output reg [1:0] next\_step,

output reg [7:0] tx\_byte,

input wire [7:0] rx\_byte,

input wire ready

);

reg ready\_reg;

reg [7:0] tx\_byte\_reg;

reg [7:0] rx\_byte\_reg;

reg [5:0] cur\_state;

reg [31:0] PC\_rx\_reg;

reg [31:0] PC\_tx\_reg;

reg byte2\_flag;

localparam idle\_ = 6'b000000;

localparam start\_rt = 6'b000001;

localparam tx\_rt = 6'b000010;

localparam rstart\_rt = 6'b000100;

localparam rx\_rt = 6'b001000;

localparam end\_rt = 6'b010000;

localparam ns\_start = 2'b01;

localparam ns\_tx = 2'b10;

localparam ns\_rx = 2'b11;

localparam ns\_end = 2'b00;

localparam device\_addr\_wr = 8'b10010000;

localparam device\_addr\_rd = 8'b10010001;

localparam temp\_reg\_addr = 8'b00000000;

initial begin

cur\_state <= idle\_;

next\_step <= ns\_end;

PC\_rx\_reg <= 0;

PC\_tx\_reg <= 0;

tx\_byte\_reg <= 0;

rx\_byte\_reg <= 0;

byte2\_flag <= 1'b0;

ready\_reg <= 1'b1;

end

integer i;

always @(posedge clk) begin

for (i=0; i<8; i=i+1) begin

tx\_byte[i] <= tx\_byte\_reg[7-i];

rx\_byte\_reg[i] <= rx\_byte[7-i];

end

end

always @(posedge clk) begin

case (cur\_state)

idle\_ : begin

PC\_rx\_reg <= PC\_rx;

if (PC\_rx\_reg != PC\_rx) begin

cur\_state <= start\_rt;

end

end

start\_rt: begin

ready\_reg <= ready;

tx\_byte\_reg <= device\_addr\_wr;

next\_step <= ns\_start;

if (ready\_reg == 1'b0 && ready == 1'b1) begin

cur\_state <= tx\_rt;

end

end

tx\_rt: begin

ready\_reg <= ready;

tx\_byte\_reg <= temp\_reg\_addr;

next\_step <= ns\_tx;

if(ready\_reg == 1'b0 && ready == 1'b1) begin

cur\_state <= rstart\_rt;

end

end

rstart\_rt : begin

ready\_reg <= ready;

tx\_byte\_reg <= device\_addr\_rd;

next\_step <= ns\_start;

if (ready\_reg == 1'b0 && ready == 1'b1) begin

cur\_state <= rx\_rt;

end

end

rx\_rt : begin

ready\_reg <= ready;

tx\_byte\_reg <= {8{byte2\_flag}};

next\_step <= ns\_rx;

if (ready\_reg == 1'b0 && ready == 1'b1) begin

case(byte2\_flag)

1'b0: begin

byte2\_flag <= 1'b1;

for (i=0; i<8; i =i+1) begin

PC\_tx\_reg[12-i] <= rx\_byte\_reg[7-i];

end

end

1'b1: begin

byte2\_flag <= 1'b0;

for (i=0; i<5; i = i+1) begin

PC\_tx\_reg[4-i] <= rx\_byte\_reg[7-i];

end

cur\_state <= end\_rt;

end

endcase

end

end

end\_rt : begin

tx\_byte\_reg <= {8{1'b0}};

next\_step <= ns\_end;

cur\_state <= idle\_;

PC\_tx <= PC\_tx\_reg;

end

default : begin

tx\_byte\_reg <= {8{1'b0}};

next\_step <= ns\_end;

cur\_state <= idle\_;

end

endcase

end

endmodule

Python code:

# -\*- coding: utf-8 -\*-

#%%

# import various libraries necessary to run your Python code

import time # time related library

import sys,os # system related library

ok\_sdk\_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\Python\\x64"

ok\_dll\_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\lib\\x64"

sys.path.append(ok\_sdk\_loc) # add the path of the OK library

os.add\_dll\_directory(ok\_dll\_loc)

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 communication with the OK board

# We will NOT load the bit file because it will be loaded using JTAG interface from Vivado

# 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 ()

#%%

# Define the two variables that will send data to the FPGA

# We will use WireIn instructions to send data to the FPGA

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns() # Update the WireIns

time.sleep(1)

dev.SetWireInValue(0x00, 1)

dev.UpdateWireIns() # Update the WireIns

print("Send GO signal to the FSM")

#%%

# Since we are using a slow clock on the FPGA to compute the results

# we need to wait for the result to be computed

time.sleep(1)

dev.UpdateWireOuts()

temp\_read = dev.GetWireOutValue(0x20)

print("temp read is " + str(temp\_read/16))

#PC\_Control = 0; # send a "stop" signal to the FSM

#dev.SetWireInValue(0x00, PC\_Control)

#dev.UpdateWireIns() # Update the WireIns

#print("Send STOP signal to the FSM")

dev.Close

#%%

Milestone2 waveforms:

