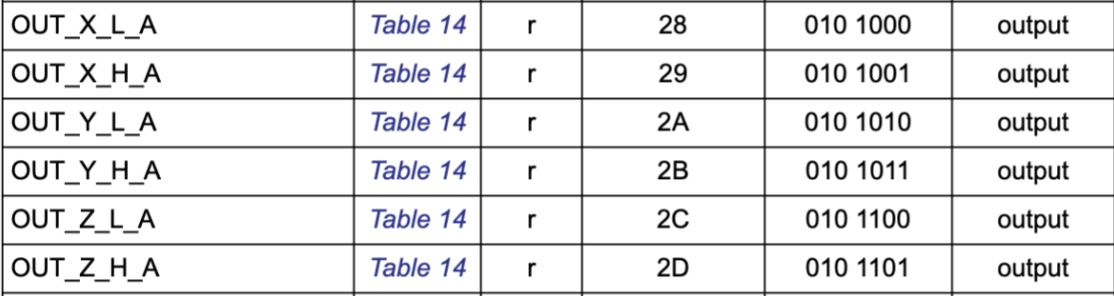
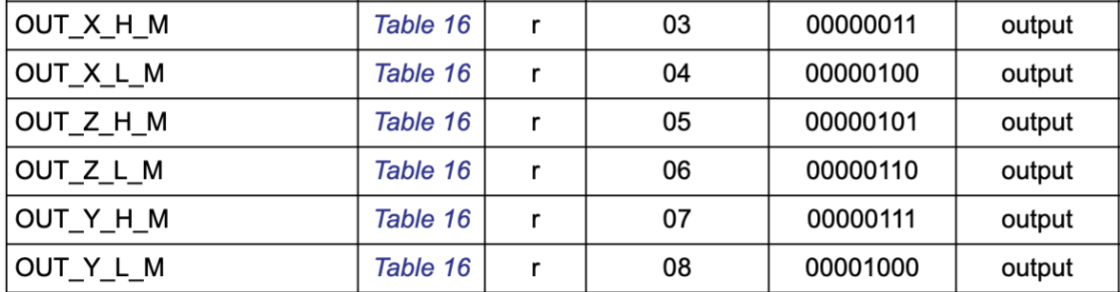
1. Look over the online documentation for the sensor. How many registers are there in the sensor? (5 pts)
   1. There are 45 registers to use and the rest are reserved but they range from 00-3A in hex.
2. What is the address for the registers that contain the acceleration and magnetic data? (5 pts)
   1. 
   2. 
3. How many bits is the output data per channel (meaning X, Y and Z direction for either magnetic or acceleration data)? (5 pts)
   1. 16 bits -> 8+8
4. Figure 1 of the sensor’s datasheet presents a block diagram of the entire system. How do you think capacitors can be used to sense acceleration information? (5 pts)
   1. The top plate of the capacitor is allowed to spring in both directions. When we do this, the capacitance of the capacitor changes in proportion to movement. Then it is up to the mechanical design to properly transduce the movements into a capacitance change.

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

#%%

# import various libraries necessary to run your Python code

import time # time related library

import sys,os # system related library

import numpy as np

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 ()

#%%

#%%

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

# we need to wait for the result to be computed

#0 is a write, 1 is a read

#address 6 was made to be PC\_control because we were trying some things.

#this is where we begin writing to the registers to enable both the Gyroscope and the Accelerometer

def Use\_Sensor(R\_or\_W, dev\_addr, sub\_addr, read\_count, data, dev):

########### Writes the Contrl Reg ############################

dev.SetWireInValue(0x01, read\_count) #how many time to iterate

dev.SetWireInValue(0x02, R\_or\_W) #Decides read or write

dev.SetWireInValue(0x03, data) # Data to write

dev.SetWireInValue(0x04, sub\_addr) #Sets the Sub Adress

dev.SetWireInValue(0x05, dev\_addr) #Sets the Dev Adress

dev.UpdateWireIns()

dev.SetWireInValue(0x00,1) #Starts FSM

dev.UpdateWireIns()

dev.SetWireInValue(0x00,0) #Turns FSM Off

dev.UpdateWireIns()

dev.UpdateWireOuts()

if(read\_count == 1):

addr = 0x21

elif(read\_count == 2):

addr = 0x22

elif(read\_count == 3):

addr = 0x23

elif(read\_count == 4):

addr = 0x24

elif(read\_count == 5):

addr = 0x25

elif(read\_count == 6):

addr = 0x26

else:

return

if (R\_or\_W == 1):

x = dev.GetWireOutValue(addr)

return x

else:

return

def twobit(m1, m2):

val\_x = (bin(m1 + m2 << 8))

if((int(val\_x, 2) >> 15)):

val\_x = int(val\_x, 2) - (1<<16)

else:

val\_x = int(val\_x, 2)

return val\_x

############### Getting Linear Accelleration ###############

'''

Use\_Sensor(0, 0x32, 0x20, 1, 0x97, dev) #write to Lin Accell CTRL\_REG\_A 8'b01010111

time.sleep(0.5)

while (1):

X\_L\_A = Use\_Sensor(1, 0x33, 0x28, 1, 0, dev) #Reads Lin Al from X Low

time.sleep(.02)

X\_H\_A = Use\_Sensor(1, 0x33, 0x29, 2, 0, dev)

time.sleep(.02)

Y\_L\_A = Use\_Sensor(1, 0x33, 0x2A, 3, 0, dev)

time.sleep(.02)

Y\_H\_A = Use\_Sensor(1, 0x33, 0x2B, 4, 0, dev)

time.sleep(.02)

Z\_L\_A = Use\_Sensor(1, 0x33, 0x2C, 5, 0, dev)

time.sleep(.02)

Z\_H\_A = Use\_Sensor(1, 0x33, 0x2D, 6, 0, dev)

time.sleep(.02)

X\_A = twobit(X\_L\_A,X\_H\_A)/16000

Y\_A = twobit(Y\_L\_A,Y\_H\_A)/16000

Z\_A = twobit(Z\_L\_A,Z\_H\_A)/16000

print(X\_A,Y\_A,Z\_A)

time.sleep(.5)

'''

############### Getting Magnetic Field ###################

while(1):

Use\_Sensor(0, 0x3C, 0x02, 1, 0x00, dev)

time.sleep(0.5)

X\_H\_M = Use\_Sensor(1, 0x3C, 0x03, 1, 0, dev) #Reads Mag Field X-Y-Z

time.sleep(.02)

X\_L\_M = Use\_Sensor(1, 0x3C, 0x04, 2, 0, dev)

time.sleep(.02)

Z\_H\_M = Use\_Sensor(1, 0x3C, 0x05, 3, 0, dev)

time.sleep(.02)

Z\_L\_M = Use\_Sensor(1, 0x3C, 0x06, 4, 0, dev)

time.sleep(.02)

Y\_H\_M = Use\_Sensor(1, 0x3C, 0x07, 5, 0, dev)

time.sleep(.02)

Y\_L\_M = Use\_Sensor(1, 0x3C, 0x08, 6, 0, dev)

X\_A = twobit(X\_L\_M,X\_H\_M)/1000

Y\_A = twobit(Y\_L\_M,Y\_H\_M)/1000

Z\_A = twobit(Z\_L\_M,Z\_H\_M)/1000

print(X\_A, Y\_A, Z\_A)

dev

dev

dev.Close

`timescale 1ns / 1ps

module I2C\_Transmit(

output [7:0] led,

input sys\_clkn,

input sys\_clkp,

output I2C\_SCL\_1,

inout I2C\_SDA\_1,

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,

output wire [16:0] out\_data,

input wire [4:0] okUH,

output wire [2:0] okHU,

inout wire [31:0] okUHU,

inout wire okAA

);

wire [31:0] sub\_addr, dev\_addr;

wire [31:0] data\_in;

reg [7:0] data\_out\_temp, data\_out\_1,data\_out\_2,data\_out\_3,data\_out\_4,data\_out\_5,data\_out\_6;

wire [31:0] stop\_read;

reg [31:0] read\_counter;

wire stop\_flag\_wire;

wire [31:0] r\_w;

reg stop\_flag;

assign stop\_flag\_wire = stop\_flag;

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

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

wire [23:0] ClkDivThreshold = 400;

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 error\_bit = 1'b1;

localparam STATE\_INIT = 8'd0;

assign led[7] = ACK\_bit;

assign led[6] = error\_bit;

assign I2C\_SCL\_1 = SCL;

assign I2C\_SDA\_1 = SDA;

initial begin

SCL = 1'b1;

SDA = 1'b1;

ACK\_bit = 1'b1;

State = 8'd0;

data\_out\_temp = 8'd0;

stop\_flag = 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 == 32'b1) begin

State <= 8'd1;

stop\_flag <= 1'b0;

end

else if (PC\_control == 32'b0) 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;

read\_counter <= 32'd0;

State <= State + 1'b1;

end

8'd2 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

//tranny address

// transmit bit 7

8'd3 : begin

SCL <= 1'b0;

SDA <= dev\_addr[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 <= dev\_addr[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 <= dev\_addr[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 <= dev\_addr[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 <= dev\_addr[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 <= dev\_addr[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 <= dev\_addr[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 <= 1'b0;

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

8'd39 : begin

SCL <= 1'b0;

SDA <= 0;

State <= State + 1'b1;

end // MSB Address bit 7

8'd40 : begin

SCL <= 1'b1;

SDA <= 0;

State <= State + 1'b1;

end

8'd41 : begin

SCL <= 1'b1;

SDA <= 0;

State <= State + 1'b1;

end

8'd42 : begin

SCL <= 1'b0;

SDA <= 0;

State <= State + 1'b1;

end

8'd43 : begin

SCL <= 1'b0;

SDA <= sub\_addr[6];

State <= State + 1'b1;

end // MSB Address bit 6

8'd44 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd45 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd46 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd47 : begin

SCL <= 1'b0;

SDA <= sub\_addr[5];

State <= State + 1'b1;

end // MSB Address bit 5

8'd48 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd49 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd50 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd51 : begin

SCL <= 1'b0;

SDA <= sub\_addr[4];

State <= State + 1'b1;

end // MSB Address bit 4

8'd52 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd53 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd54 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd55 : begin

SCL <= 1'b0;

SDA <= sub\_addr[3];

State <= State + 1'b1;

end // MSB Address bit 3

8'd56 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd57 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd58 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd59 : begin

SCL <= 1'b0;

SDA <= sub\_addr[2];

State <= State + 1'b1;

end // MSB Address bit 2

8'd60 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd61 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd62 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd63 : begin

SCL <= 1'b0;

SDA <= sub\_addr[1];

State <= State + 1'b1;

end // MSB Address bit 1

8'd64 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd65 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd66 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd67 : begin

SCL <= 1'b0;

SDA <= sub\_addr[0];

State <= State + 1'b1;

end // MSB Address bit 0

8'd68 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd69 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd70 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

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

8'd71 : begin

SCL <= 1'b0;

SDA <= 1'bz;

State <= State + 1'b1;

end

8'd72 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd73 : begin

SCL <= 1'b1;

ACK\_bit <= SDA;

State <= State + 1'b1;

end

8'd74 : begin

SCL <= 1'b0;

if (r\_w[0] == 1'b0)begin //goes to write

State <= 8'd192;

SDA <= 1'b0;

end

else if (r\_w[0] == 1'b1) begin //goes to rea

State <= State + 1'b1;

end

end

/////////// Repeat Start to read values /////////////

8'd75 : begin

SCL <= 1'b0;

SDA <= 1'b1;

State <= State + 1'b1;

end

8'd76 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd77 : begin

SCL <= 1'b1;

SDA <= 1'b0;

State <= State + 1;

end

8'd78 : begin

SCL <= 1'b0;

SDA <= 1'b0;

State <= State + 1;

end

//this is when we resend the address with read

8'd79 : begin

SCL <= 1'b0;

SDA <= dev\_addr[7];

State <= State +1'b1;

end

8'd80 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd81 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd82 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd83 : begin

SCL <= 1'b0;

SDA <= dev\_addr[6];

State <= State +1'b1;

end

8'd84 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd85 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd86 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd87 : begin

SCL <= 1'b0;

SDA <= dev\_addr[5];

State <= State +1'b1;

end

8'd88 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd89 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd90 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd91 : begin

SCL <= 1'b0;

SDA <= dev\_addr[4];

State <= State +1'b1;

end

8'd92 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd93 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd94 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd95 : begin

SCL <= 1'b0;

SDA <= dev\_addr[3];

State <= State +1'b1;

end

8'd96 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd97: begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd98: begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd99 : begin

SCL <= 1'b0;

SDA <= dev\_addr[2];

State <= State +1'b1;

end

8'd100 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd101 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd102 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd103 : begin

SCL <= 1'b0;

SDA <= dev\_addr[1];

State <= State +1'b1;

end

8'd104 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd105 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd106 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd107 : begin

SCL <= 1'b0;

SDA <= 1'b1;

State <= State +1'b1;

end

8'd108 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd109 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd110 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

////// recive Ack knowledge bit ////////////

8'd111 : begin

SCL <= 1'b0;

SDA <= 1'bz;

State <= State + 1'b1;

end

8'd112 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd113 : begin

SCL <= 1'b1;

ACK\_bit <= SDA;

State <= State + 1'b1;

end

8'd114 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

///////////////// Read the byte

8'd115 : begin

SCL <= 1'b0;

data\_out\_temp[7] <= SDA;

State <= State +1'b1;

end

8'd116 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd117 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd118 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd119 : begin

SCL <= 1'b0;

data\_out\_temp[6] <= SDA;

State <= State +1'b1;

end

8'd120 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd121 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd122 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd123 : begin

SCL <= 1'b0;

data\_out\_temp[5] <= SDA;

State <= State +1'b1;

end

8'd124 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd125 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd126 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd127 : begin

SCL <= 1'b0;

data\_out\_temp[4] <= SDA;

State <= State +1'b1;

end

8'd128 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd129 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd130 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd131 : begin

SCL <= 1'b0;

data\_out\_temp[3] <= SDA;

State <= State +1'b1;

end

8'd132 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd133 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd134 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd135 : begin

SCL <= 1'b0;

data\_out\_temp[2] <= SDA;

State <= State +1'b1;

end

8'd136 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd137 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd138 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd139 : begin

SCL <= 1'b0;

data\_out\_temp[1] <= SDA;

State <= State +1'b1;

end

8'd140 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd141 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd142 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd143 : begin

SCL <= 1'b0;

data\_out\_temp[0] <= SDA;

State <= State +1'b1;

end

8'd144 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd145 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd146 : begin

SCL <= 1'b0;

State <= 8'd183; //we are reading one bit at a time

end

////// Send Ack knowledge bit ////////////

8'd147 : begin

SCL <= 1'b0;

SDA <= 1'bz;

read\_counter <= read\_counter + 1;

State <= State + 1'b1;

end

8'd148 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd149 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd150 : begin

SCL <= 1'b0;

SDA <= 1'b0;

if (read\_counter == stop\_read[3:0]) begin

State <= 8'd183;

end

else begin

State <= 8'd115;

end

end

//////// SEND NACK ///////////

8'd183 : begin

SCL <= 1'b0;

SDA <= 1'b1;

case (stop\_read)

32'd1 : data\_out\_1 <= data\_out\_temp;

32'd2 : data\_out\_2 <= data\_out\_temp;

32'd3 : data\_out\_3 <= data\_out\_temp;

32'd4 : data\_out\_4 <= data\_out\_temp;

32'd5 : data\_out\_5 <= data\_out\_temp;

32'd6 : data\_out\_6 <= data\_out\_temp;

endcase

State <= State +1'b1;

stop\_flag <= 1'b1;

end

8'd184 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd185 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd186 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

///////// Stop sequenc and back to state 1 to reset //////////

8'd187 : begin

SCL <= 1'b0;

SDA <= 1'b0;

State <= State + 1'b1;

end

8'd188 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd189 : begin

SCL <= 1'b1;

SDA <= 1'b1;

State <= State + 1'b1;

end

8'd190 : begin

SCL <= 1'b0;

State <= State + 1'b1;

end

8'd191 : begin

SCL <= 1'b0;

State <= 8'd0;

end

//transmit data for register

8'd192 : begin

SCL <= 1'b0;

SDA <= data\_in[7];

State <= State + 1;

end

8'd193 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd194 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd195 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd196 : begin

SCL <= 1'b0;

SDA <= data\_in[6];

State <= State + 1;

end

8'd197 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd198 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd199 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd200 : begin

SCL <= 1'b0;

SDA <= data\_in[5];

State <= State + 1;

end

8'd201 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd202 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd203 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd204 : begin

SCL <= 1'b0;

SDA <= data\_in[4];

State <= State + 1;

end

8'd205 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd206 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd207 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd208 : begin

SCL <= 1'b0;

SDA <= data\_in[3];

State <= State + 1;

end

8'd209 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd210 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd211 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd212 : begin

SCL <= 1'b0;

SDA <= data\_in[2];

State <= State + 1;

end

8'd213 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd214 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd215 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd216 : begin

SCL <= 1'b0;

SDA <= data\_in[1];

State <= State + 1;

end

8'd217 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd218 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd219 : begin

SCL <= 1'b0;

State <= State + 1;

end

8'd220 : begin

SCL <= 1'b0;

SDA <= data\_in[0];

State <= State + 1;

end

8'd221 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd222 : begin

SCL <= 1'b1;

State <= State + 1;

end

8'd223 : begin

SCL <= 1'b0;

State <= State + 1;

end

//read the ack bit

8'd224 : begin

SCL <= 1'b0;

SDA <= 1'bz;

State <= State + 1'b1;

end

8'd225 : begin

SCL <= 1'b1;

State <= State + 1'b1;

end

8'd226 : begin

SCL <= 1'b1;

ACK\_bit <= SDA;

State <= State + 1'b1;

end

8'd227 : begin

SCL <= 1'b0;

State <= 8'd187;

end

default : begin

error\_bit <= 0;

end

endcase

end

// OK Interface

wire okClk;

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

// The data is communicated via memeory location 0x00

localparam endPt\_count = 6;

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

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

okWireIn wire10 (.okHE(okHE),

.ep\_addr(8'h00),

.ep\_dataout(PC\_control));

okWireIn wire11 (.okHE(okHE),

.ep\_addr(8'h01),

.ep\_dataout(stop\_read));

okWireIn wire12 (.okHE(okHE),

.ep\_addr(8'h02),

.ep\_dataout(r\_w));

okWireIn wire13 (.okHE(okHE),

.ep\_addr(8'h03),

.ep\_dataout(data\_in));

okWireIn wire14 (.okHE(okHE),

.ep\_addr(8'h04),

.ep\_dataout({sub\_addr}));

okWireIn wire15 (.okHE(okHE),

.ep\_addr(8'h05),

.ep\_dataout({dev\_addr}));

okWireOut wire21 (.okHE(okHE),

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

.ep\_addr(8'h21),

.ep\_datain({24'd0,data\_out\_1}));

okWireOut wire22 (.okHE(okHE),

.okEH(okEHx[1\*65+:65]),

.ep\_addr(8'h22),

.ep\_datain({24'd0,data\_out\_2}));

okWireOut wire23 (.okHE(okHE),

.okEH(okEHx[2\*65+:65]),

.ep\_addr(8'h23),

.ep\_datain({24'd0,data\_out\_3}));

okWireOut wire24 (.okHE(okHE),

.okEH(okEHx[3\*65+:65]),

.ep\_addr(8'h24),

.ep\_datain({24'd0,data\_out\_4}));

okWireOut wire25 (.okHE(okHE),

.okEH(okEHx[4\*65+:65]),

.ep\_addr(8'h25),

.ep\_datain({24'd0,data\_out\_5}));

okWireOut wire26 (.okHE(okHE),

.okEH(okEHx[5\*65+:65]),

.ep\_addr(8'h26),

.ep\_datain({24'd0,data\_out\_6}));

endmodule