Postlab question:

The relationship between the acceleration and applied power voltage is overall linear, with higher acceleration when higher voltage is applied.

Verilog code:

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\_1,

inout I2C\_SDA\_1,

input [4:0] okUH,

output [2:0] okHU,

inout [31:0] okUHU,

inout okAA,

output PMOD\_A1,

output PMOD\_A2,

input PMOD\_A3,

input PMOD\_A4,

output PMOD\_A7,

output PMOD\_A8,

input PMOD\_A9,

input PMOD\_A10

);

// 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 [31:0] PC\_slave\_addr;

wire [31:0] PC\_addr;

wire [31:0] PC\_val;

wire [31:0] PMOD\_UTIL;

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));

okWireIn wire11 ( .okHE(okHE),

.ep\_addr(8'h01),

.ep\_dataout(PC\_slave\_addr));

okWireIn wire12 ( .okHE(okHE),

.ep\_addr(8'h02),

.ep\_dataout(PC\_addr));

okWireIn wire13 ( .okHE(okHE),

.ep\_addr(8'h03),

.ep\_dataout(PC\_val));

okWireIn wire14 ( .okHE(okHE),

.ep\_addr(8'h04),

.ep\_dataout(PMOD\_UTIL));

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;

wire busy;

I2C\_driver I2C\_SERDES (

.busy(busy),

.led(led),

.clk(clk),

.ADT7420\_A0(ADT7420\_A0),

.ADT7420\_A1(ADT7420\_A1),

.I2C\_SCL\_0(I2C\_SCL\_1),

.I2C\_SDA\_0(I2C\_SDA\_1),

.ACK(ACK),

.SCL(SCL),

.SDA(SDA),

.State(State),

.tx\_byte(tx\_byte),

.rx\_byte(rx\_byte),

.next\_step(next\_step),

.ready(ready)

);

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

wire [9:0] cur\_state;

wire [31:0] PC\_rx\_reg1;

wire [31:0] PC\_rx\_reg2;

wire [3:0] motor\_fb;

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

TS\_controller TS\_controller(

.clk(clk),

.PC\_rx(PC\_rx),

.PC\_tx(PC\_tx),

.PC\_slave\_addr(PC\_slave\_addr),

.PC\_addr(PC\_addr),

.PC\_val(PC\_val),

.next\_step(next\_step),

.tx\_byte(tx\_byte),

.rx\_byte(rx\_byte),

.cur\_state(cur\_state),

.PC\_rx\_reg1(PC\_rx\_reg1),

.PC\_rx\_reg2(PC\_rx\_reg2),

.ready(ready)

);

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

//PMOD Interface////////////////////////////////////////////////////////////////////////

PMOD\_driver PMOD\_driver(

.clk(clk),

.PMOD\_1(PMOD\_A1),

.PMOD\_2(PMOD\_A2),

.PMOD\_3(PMOD\_A3),

.PMOD\_4(PMOD\_A4),

.PMOD\_7(PMOD\_A7),

.PMOD\_8(PMOD\_A8),

.PMOD\_9(PMOD\_A9),

.PMOD\_10(PMOD\_A10),

.motor\_fb(motor\_fb),

.PMOD\_UTIL(PMOD\_UTIL)

);

//Instantiate the ILA module

ila\_0 ila\_sample12 (

.clk(clk),

.probe0({PMOD\_A1,PMOD\_A2,PMOD\_A3,PMOD\_A4,PMOD\_A7,PMOD\_A8,PMOD\_A9,PMOD\_A10}),

.probe1(PMOD\_UTIL),

.probe2(motor\_fb),

.probe3(cur\_state));

endmodule

PMOD.v:

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 2024/10/21 10:45:31

// Design Name:

// Module Name: PMOD

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module PMOD\_driver(

input wire clk,

output reg[3:0] motor\_fb,

output wire PMOD\_1,

output wire PMOD\_2,

input wire PMOD\_3,

input wire PMOD\_4,

output wire PMOD\_7,

output wire PMOD\_8,

input wire PMOD\_9,

input wire PMOD\_10,

input wire[31:0] PMOD\_UTIL

);

// deconcanate input signal

wire[1:0] motor\_sel;

wire[1:0] dir\_sel;

wire[27:0] cycle\_set;

// internal counter

reg[28:0] cycle\_counter;

reg[18:0] clock\_counter;

//CDC utilities

reg[31:0] CDC\_REG1;

reg[31:0] CDC\_REG2;

//Latches

reg[1:0] motor\_sel\_reg;

reg[27:0] cycle\_set\_reg;

reg[1:0] dir\_sel\_reg;

reg PMOD\_CLK;

reg CLK\_EN;

initial begin

cycle\_counter <= 29'd0;

clock\_counter <= 19'd0;

motor\_sel\_reg <= 2'd0;

cycle\_set\_reg <= 28'd0;

PMOD\_CLK <= 1'b0;

CLK\_EN <= 1'b0;

end

always @(posedge clk)begin

CDC\_REG1 <= PMOD\_UTIL;

CDC\_REG2 <= CDC\_REG1;

motor\_fb <= {PMOD\_3,PMOD\_4,PMOD\_9,PMOD\_10};

end

//deconcatenate input signal vector

assign motor\_sel = CDC\_REG2[31:30];

assign dir\_sel = CDC\_REG2[29:28];

assign cycle\_set = CDC\_REG2[27:0];

//settting output signal

assign PMOD\_1 = PMOD\_CLK & motor\_sel\_reg[0];

assign PMOD\_2 = dir\_sel\_reg[0];

assign PMOD\_7 = PMOD\_CLK & motor\_sel\_reg[1];

assign PMOD\_8 = dir\_sel\_reg[1];

always @(posedge clk) begin

case (CLK\_EN)

1'd0 : begin

if (motor\_sel) begin

cycle\_counter <= 30'd0;

clock\_counter <= 19'd0;

motor\_sel\_reg <= motor\_sel;

cycle\_set\_reg <= cycle\_set;

dir\_sel\_reg <= dir\_sel;

CLK\_EN <= 1'd1;

end

end

1'd1 : begin

case (clock\_counter)

19'd0 : begin

clock\_counter <= clock\_counter + 1;

cycle\_counter <= cycle\_counter + 1;

PMOD\_CLK <= ~PMOD\_CLK;

end

19'd499999 : begin

clock\_counter <= 19'd0;

if(cycle\_counter[28:1] == cycle\_set\_reg)begin

cycle\_counter <= 30'd0;

motor\_sel\_reg <= 2'd0;

cycle\_set\_reg <= 28'd0;

dir\_sel\_reg <= 2'd0;

CLK\_EN <= 1'd0;

PMOD\_CLK <= 1'd0;

end

end

default : clock\_counter <= clock\_counter + 1;

endcase

end

endcase

end

endmodule

Python code:

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

#%%

# import various libraries necessary to run your Python code

import pyvisa as visa # You should pip install pyvisa and restart the kernel.

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

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"

mpl.style.use('ggplot')

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

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

import ok # OpalKelly library

#%%

def write\_to\_device(slave\_addr, reg\_addr, value):

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns()

dev.SetWireInValue(0x01, slave\_addr)

dev.SetWireInValue(0x02, reg\_addr)

dev.SetWireInValue(0x03, value)

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.5)

dev.SetWireInValue(0x00, 1) # Write trigger

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.5)

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns() # Update the WireIns

#%%

def read\_from\_device(slave\_addr, reg\_addr):

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.05)

dev.SetWireInValue(0x01, slave\_addr)

dev.SetWireInValue(0x02, reg\_addr)

dev.SetWireInValue(0x00, 2) # Read trigger

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.05)

dev.UpdateWireOuts()

read = dev.GetWireOutValue(0x20)

if slave\_addr == 0x3C:

m\_L = read // 2\*\*8

m\_H = read - (m\_L \* 2\*\*8)

read = m\_H \* 2\*\*8 + m\_L

if read >= 2\*\*15:

read = read - 2\*\*16 # deal with 2's complement

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns()

return read

#%%

# dir = 0 => forward, dir = 1 => backward

def run\_motor(direction, duration):

pmod\_util = duration + 3 \* 2 \*\* 30

pmod\_util = pmod\_util + (3 \* direction) \* 2 \*\* 28

dev.SetWireInValue(0x04, pmod\_util)

dev.UpdateWireIns()

time.sleep(0.2)

dev.SetWireInValue(0x04, 0)

dev.UpdateWireIns()

#%%

# 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 ()

#%%

# This section of the code cycles through all USB connected devices to the computer.

# The code figures out the USB port number for each instrument.

# The port number for each instrument is stored in a variable named “instrument\_id”

# If the instrument is turned off or if you are trying to connect to the

# keyboard or mouse, you will get a message that you cannot connect on that port.

device\_manager = visa.ResourceManager()

devices = device\_manager.list\_resources()

number\_of\_device = len(devices)

power\_supply\_id = -1

waveform\_generator\_id = -1

digital\_multimeter\_id = -1

oscilloscope\_id = -1

# assumes only the DC power supply is connected

for i in range (0, number\_of\_device):

# check that it is actually the power supply

try:

device\_temp = device\_manager.open\_resource(devices[i])

print("Instrument connect on USB port number [" + str(i) + "] is " + device\_temp.query("\*IDN?"))

if (device\_temp.query("\*IDN?") == 'HEWLETT-PACKARD,E3631A,0,3.2-6.0-2.0HEWLETT-PACKARD,E3631A,0,3.2-6.0-2.0\r\n'):

power\_supply\_id = i

if (device\_temp.query("\*IDN?") == 'HEWLETT-PACKARD,E3631A,0,3.0-6.0-2.0\r\n'):

power\_supply\_id = i

if (device\_temp.query("\*IDN?") == 'Agilent Technologies,33511B,MY52301259,3.03-1.19-2.00-52-00\n'):

waveform\_generator\_id = i

if (device\_temp.query("\*IDN?") == 'Agilent Technologies,34461A,MY53208026,A.01.10-02.25-01.10-00.35-01-01\n'):

digital\_multimeter\_id = i

if (device\_temp.query("\*IDN?") == 'Keysight Technologies,34461A,MY53212931,A.02.08-02.37-02.08-00.49-01-01\n'):

digital\_multimeter\_id = i

if (device\_temp.query("\*IDN?") == 'KEYSIGHT TECHNOLOGIES,MSO-X 3024T,MY54440318,07.50.2021102830\n'):

oscilloscope\_id = i

device\_temp.close()

except:

print("Instrument on USB port number [" + str(i) + "] cannot be connected. The instrument might be powered of or you are trying to connect to a mouse or keyboard.\n")

#%%

# Open the USB communication port with the power supply.

# The power supply is connected on USB port number power\_supply\_id.

# If the power supply ss not connected or turned off, the program will exit.

# Otherwise, the power\_supply variable is the handler to the power supply

if (power\_supply\_id == -1):

print("Power supply instrument is not powered on or connected to the PC.")

else:

print("Power supply is connected to the PC.")

power\_supply = device\_manager.open\_resource(devices[power\_supply\_id])

#%%

# Open the USB communication port with the power supply.

# The power supply is connected on USB port number power\_supply\_id.

# If the power supply ss not connected or turned off, the program will exit.

# Otherwise, the power\_supply variable is the handler to the power supply

if (digital\_multimeter\_id == -1):

print("Digital multimeter instrument is not powered on or connected to the PC.")

else:

print("Digital multimeter is connected to the PC.")

digital\_multimeter = device\_manager.open\_resource(devices[digital\_multimeter\_id])

#%%

# Open the USB communication port with the power supply.

# The power supply is connected on USB port number power\_supply\_id.

# If the power supply ss not connected or turned off, the program will exit.

# Otherwise, the power\_supply variable is the handler to the power supply

if (oscilloscope\_id == -1):

print("Oscilloscope instrument is not powered on or connected to the PC.")

else:

print("Oscilloscope is connected to the PC.")

oscilloscope = device\_manager.open\_resource(devices[oscilloscope\_id])

#%% Reg and value constants

ctrl\_reg\_1\_addr = 0x20

ctrl\_reg\_1\_value = 0x37

mr\_reg\_m\_addr = 0x02

mr\_reg\_m\_value = 0x00

accel\_slave\_addr = 0x32

magnet\_slave\_addr = 0x3C

x\_a\_reg\_addr = 0xA8

y\_a\_reg\_addr = 0xAA

z\_a\_reg\_addr = 0xAC

x\_m\_reg\_addr = 0x03

y\_m\_reg\_addr = 0x07

z\_m\_reg\_addr = 0x05

#%%

print(power\_supply.write("OUTPUT ON"))

write\_to\_device(accel\_slave\_addr, ctrl\_reg\_1\_addr, ctrl\_reg\_1\_value) # Enable output

write\_to\_device(magnet\_slave\_addr, mr\_reg\_m\_addr, mr\_reg\_m\_value) # Continuous-conversion mode

output\_voltage = np.arange(3, 5.5, 0.5)

measured\_accel = np.array([]) # create an empty list to hold our values

timer = np.arange(0.1, 1.1, 0.1)

try:

for v in output\_voltage:

accels = np.array([])

power\_supply.write("APPLy P25V, %0.2f, 0.1" % v)

run\_motor(0, 400)

time.sleep(2)

run\_motor(1, 200)

for i in range(10):

accels = np.append(accels, abs(read\_from\_device(accel\_slave\_addr, z\_a\_reg\_addr)))

measured\_accel = np.append(measured\_accel, np.max(accels))

time.sleep(1)

except KeyboardInterrupt:

pass

print(power\_supply.write("OUTPUT OFF"))

plt.figure()

plt.plot(output\_voltage, measured\_accel)

plt.title("Applied Volts vs. Measured Acceleration")

plt.xlabel("Applied Volts [V]")

plt.ylabel("Measured Acceleration [g]")

plt.draw()