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()
```