**Final project report**

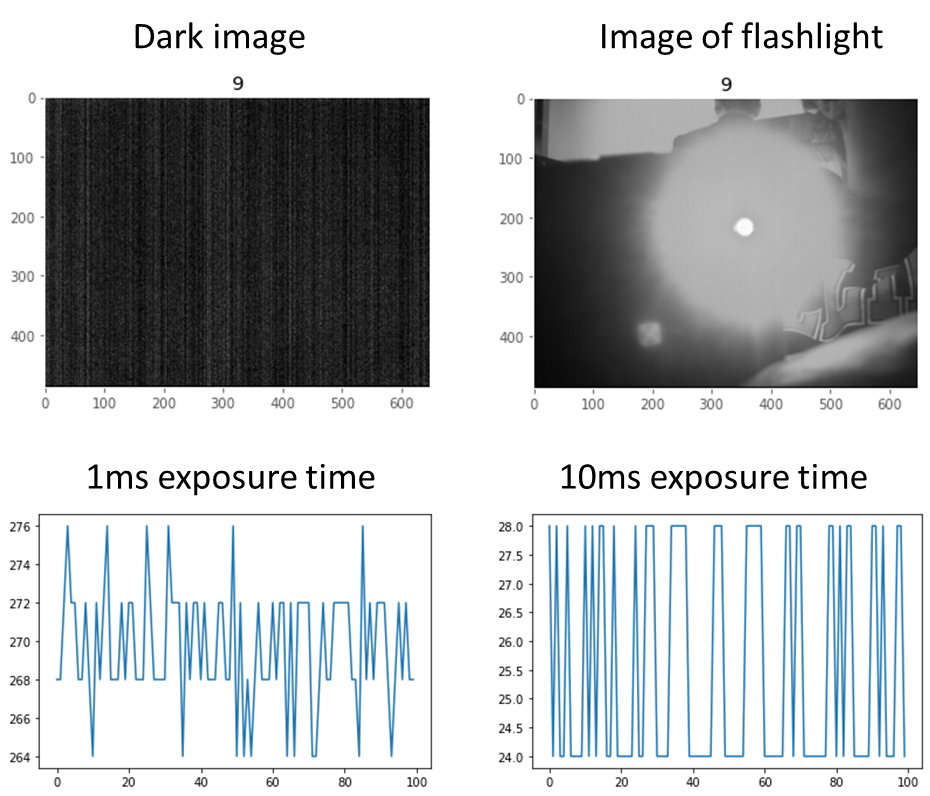
1. **Characterization of the image sensor**

To evaluate the performance of the image sensor, we measured its spatial and temporal noise and signal-to-noise ratio.

**1.1 Spatial and temporal noise of the image sensor**

The spatial noise was measured by capturing a dark frame with the image sensor. We placed a black reference in front of the lens and took a picture. The exposure time was set to 10 msec. The spatial noise was calculated by the standard deviation of the dark frame. The noise image is show in Figure 1 and spatial noise is 0.978. To calculate the temporal noise, we imaged a static object over 100 frames and calculated the standard deviation of the intensity value for pixel 50,50 over these frames as the temporal noise. The temporal fluctuation of the intensity at pixel 50,50 is shown in Figure1 and the temporal noise is 3.0557 when the exposure time is 10 msec. We also calculated the temperature noise when the exposure time is 1 msec and the value is 1,9505.

**1.1 Signal-to-noise ratio of the image sensor**

We collected an image of flashlight, which is our tracking target and calculated the SNR of the image, which can be estimated by comparing the signal level (captured image) with the noise level (dark frame or black reference image). The mean intense of the image is 81.83 and the SNR is 38.448 dB. 

1. **Characterization of acceleration sensor**

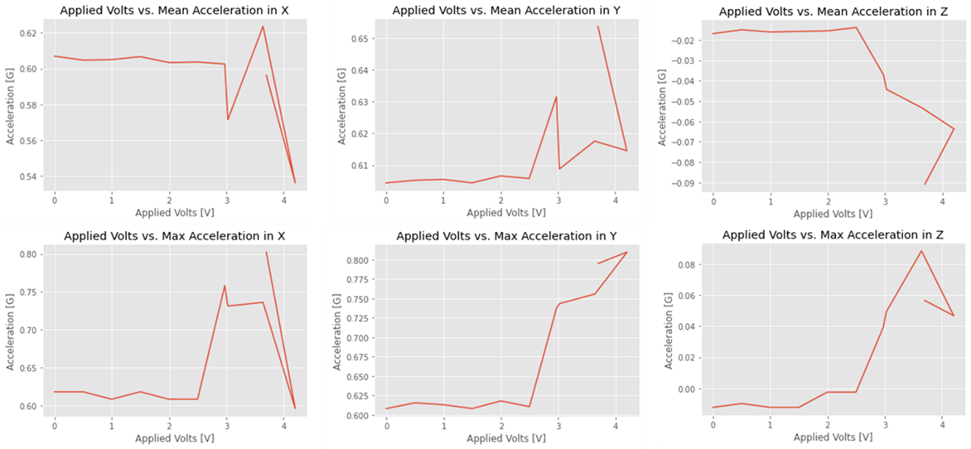
**2.1 Noise in the acceleration sensor**

We measured the acceleration of the motor movement (under the same voltage supply, 3.5 V) and collected 500 acceleration data. During the measurement, the motor did not move to avoid motor noise. The acceleration data in X,Y,Z domain is shown in Figure 2. We calculated the standard deviation of the acceleration signal as the noise of the acceleration sensor. The noise of acceleration measurement in X domain is 0.00453, the noise in Y domain is 0.00384 and the noise in Z domain is 0.00587. We can appreciate the similar noise level in X,Y,Z domain measurement.

**2.2 SNR of the acceleration sensor**

To calculate SNR of the acceleration sensor, we calculated the mean acceleration data in X domain, which is 1.05498 and the SNR is 47.35 dB. 

1. **Characterization of motor**

We applied different voltage on the motor and measured the acceleration under different voltage. The motor was set to rotate colorwisely with 100 cycles. The results in Figure 3 reveal there is no acceleration if the voltage is lower than 3 V. If it is higher than 3V, the acceleration is linear with the power voltage. However, because of the limited space for the movement, the acceleration gets blocked sometimes. 

1. **Tracking algorithm**

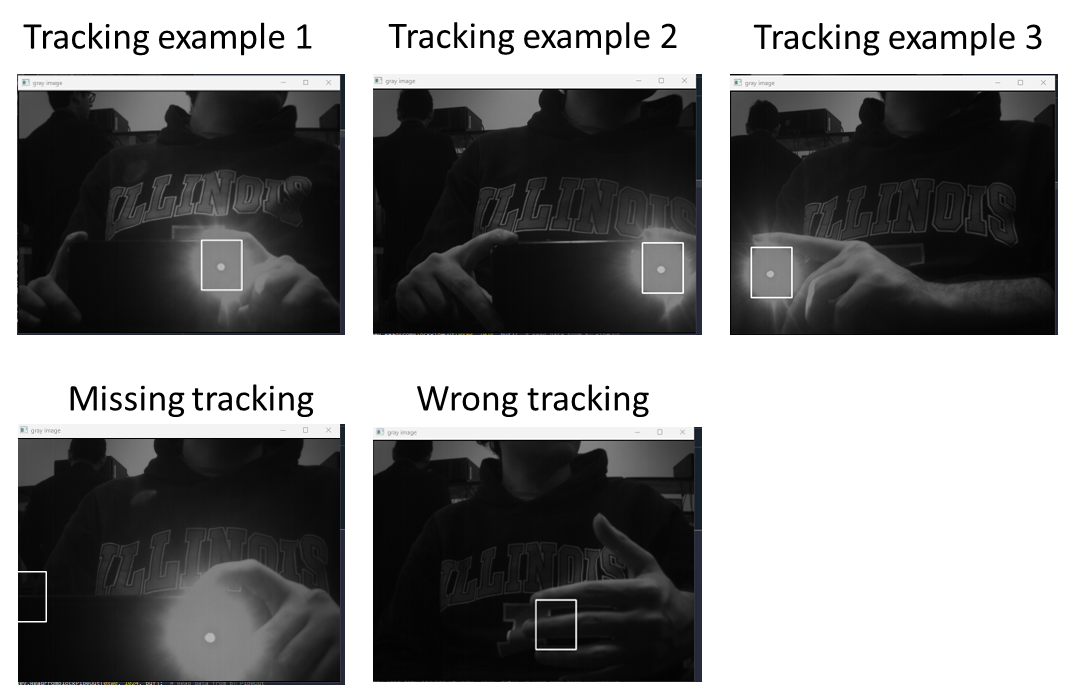
**4.1 Implementation of real-time tracking**

To achieve real-time tracking, we developed a real-time tracking algorithm based on Minimum Output Sum of Square Error tracker (MOSSE, implemented with the function trackerMOSSE in opencv). The MOSSE tracker is known for its efficiency and robustness in real-time object tracking scenarios. The MOSSE tracker is based on a correlation filter designed to learn the appearance of the target object in the frequency domain. The initial frame with the target object is manually selected and the filter is convolved with the FFT-transformed image patches in different time to find the location of the target object, which is the peak of correlation map. After calculating the location of target object each time, we compare the location of target object (in pixels) with the center line of the image (which is 324). If the location is at the left part of the image center, the camera rotates counter-clockwisely using the motor. Other wise, the camera rotates clockwisely. The rotating angle of the camera is proportional to the cycles sending to the motor control. To avoid the camera rotating too much or too little, we set the cycles is proportional to the distance between the location of the target object and the center line in the image, and the ratio is calibrated in the experiment.

In addition, to achieve real-time tracking with real-time image display, we applied multi-thread in python to run image acquisition, tracking, motor control and reading acceleration task and image display task separately. Enabling parallel processing, the final frame rate we achieved is 22 Hz.

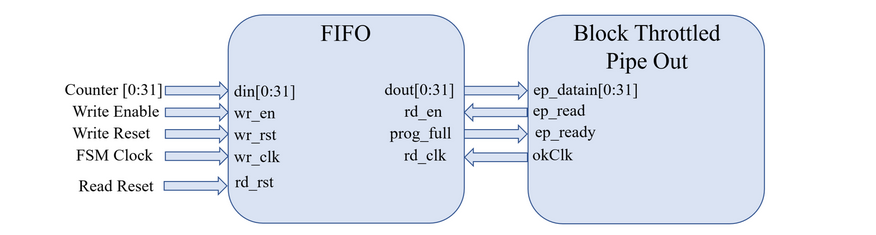
**4.2 Accuracy of the tracking algorithm**

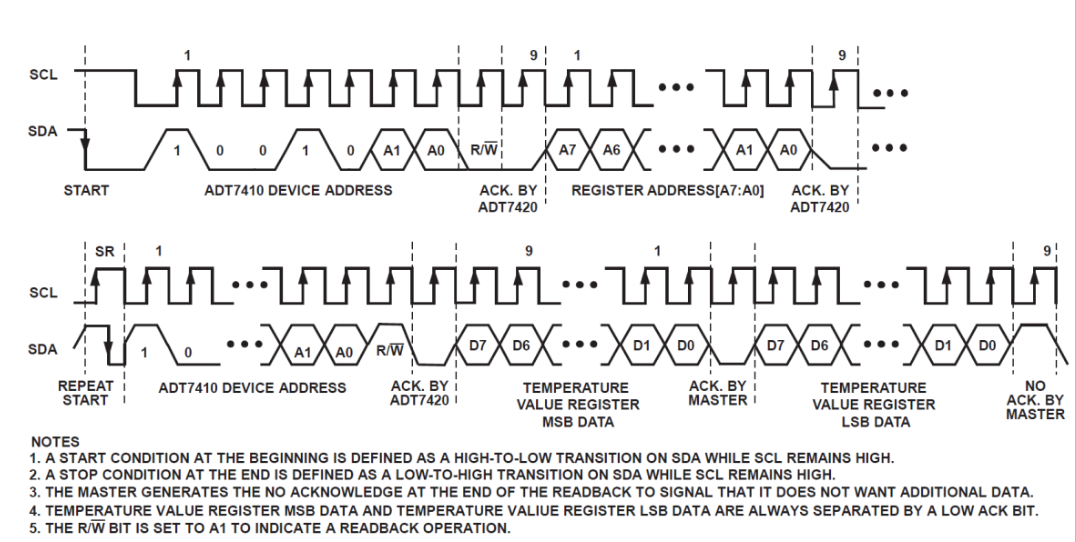
The tracking is started with selected the target object, as show in Figure 4. The tracking algorithm can track the object from left to right (Figure 5, tracking example 2 and 3) and work well for different moving speed. However, the tracking algorithm can miss the object if the object changes its pattern, for example, becoming larger in our case due to closer to the camera (Figure 5, missing tracking). It may also track other objects when the target object is out of imaging view. (Figure 5, wrong tracking). However, the tracking can become normal when the target object goes back to the view. In addition, we also tested the algorithm provided by the instruction. During imaging, we collected two consecutive frames from the image sensor and subtracted the two images to get the difference. Then the user-set threshold was applied to further remove background signal. After that, we calculated the location (x coordinate and y coordinate) of the difference image with the center of mass. The camera rotates counter-clockwisely if the location is at the left part of the image center. Otherwise, the camera rotates clockwisely. However, this algorithm has the worse performance. First, instead of the movement of target object, some objects like people also move, the subtraction can contain another object movement, causing tracking noise. In addition, the algorithm prefers faster movement. For slow movement, the real location of target object is close in two continuous frames and the difference of two continuous frames cannot reflect the real-location of the target object and can cause tracking error.

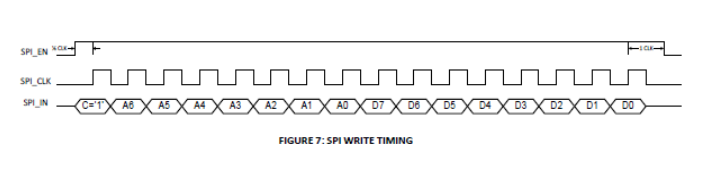


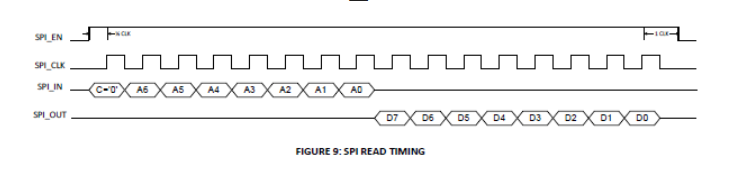
**5.1 Data Transfer**

When transferring the necessary data from the camera to the PC, a major issue that arises is the massive clock differential between the USB data transfer and the camera clock. If we were to simply transfer the camera results to the PC unconditionally, the end result would have lost most of the important data. The solution to this Clock Domain Crossing would be to instantiate a FIFO. Drive the write side of the FIFO with the camera FSM, and the read side of the FIFO with the Block-Throttled pipe. This will safely transfer any necessary data from the low clock domain to the high clock domain without losing any data.

**FIFO implementation in System**

**I2C Protocol Input Sequence** 



**5.2 SPI/I2C implementation and usage**

The protocols that were used to acquire the acceleration data and CMOS image feed were transferred across the I2C and SPI protocols respectively. These protocols were built directly onto the FPGA as a bitfile using Verilog and the Opal kelly python API to transfer the necessary data from the FPGA, to the PC. The toplevel of the program consists of a FSM that correctly resets the FIFO used in acquiring an image from the camera as well as raising the necessary fram\_req signal to acquire the frame. As well as the instantiations of the I2C protocol, and the SPI protocol. The I2C protocol directly handles the data of the accelerator sensor on the FPGA. The SPI protocol handles the necessary SPI registers needed to be programmed for the camera. Both of these protocols acquire their respective input data from the OkWire module, which handles transferring user defined arguments from Python. Some examples of these user arguments are Device Address, Sub-Register Address for I2C, and SPI register, Value to be set for SPI. The main difference between these protocols is that I2C follows a more rigid structure, which involves an ACK bit as well as a Master and Slave device POV for the ACK as well as using two predefined pins, SCL (Hard-Coded Clock), SDA (Data Bus).

**Conclusion**

In this final project, we demonstrated real-time tracking of objects with the ECE 437 sensor board. We showed real-time image display with frame rate of 22 Hz and real-time tracking of flashlight. The camera rotates to track the object based on the location of the object. In addition, the acceleration data can be acquired with a speed of 100 data points per second.

`timescale 1 ps / 1 ps

module BTPipeExample(

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\_CLK\_IN,

input CVM300\_CLK\_OUT,

input[9:0] CVM300\_D,

input CVM300\_Data\_valid,

output CVM300\_FRAME\_REQ,

output I2C\_SCL\_1,

inout I2C\_SDA\_1,

output PMOD\_A1,

output PMOD\_A2

);

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)

);

//Depending on the number of outgoing endpoints, adjust endPt\_count accordingly.

//In this example, we have 1 output endpoints, hence endPt\_count = 1.

localparam endPt\_count = 20;

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 = 5;

wire FSM\_Clk, ILA\_Clk, MotorClk;

reg [20:0] counter = 21'd0;

wire Out;

reg Raise\_REQ\_FLAG;

ClockGenerator ClockGenerator1 ( .sys\_clkn(sys\_clkn),

.sys\_clkp(sys\_clkp),

.ClkDivThreshold(ClkDivThreshold),

.FSM\_Clk(FSM\_Clk),

.ILA\_Clk(ILA\_Clk),

.Motor\_CLK(MotorClk) );

assign CVM300\_CLK\_IN = FSM\_Clk;

assign Out = CVM300\_CLK\_OUT;

localparam STATE\_INIT = 8'd0;

localparam STATE\_RESET = 8'd1;

localparam STATE\_DELAY = 8'd2;

localparam STATE\_RESET\_FINISHED = 8'd3;

localparam STATE\_ENABLE\_WRITING = 8'd4;

localparam STATE\_COUNT = 8'd5;

localparam STATE\_LOWER\_SPI = 8'd8;

localparam STATE\_FRAME = 8'd9;

reg [15:0] counter\_delay = 16'd0;

reg [7:0] State = STATE\_INIT;

reg [7:0] led\_register = 0;

reg [3:0] button\_reg, write\_enable\_counter;

reg write\_reset, read\_reset, write\_enable;

wire [31:0] Reset\_Counter;

wire FIFO\_read\_enable, FIFO\_BT\_BlockSize\_Full, FIFO\_full, FIFO\_empty, BT\_Strobe;

wire [31:0] FIFO\_data\_out;

reg CVM\_FRAM\_REQ, CVM\_SYS\_REG\_N;

assign led[0] = ~FIFO\_empty;

assign led[1] = ~FIFO\_full;

assign led[2] = ~FIFO\_BT\_BlockSize\_Full;

assign led[3] = ~FIFO\_read\_enable;

assign led[7] = ~read\_reset;

assign led[6] = ~write\_reset;

assign CVM300\_FRAME\_REQ = CVM\_FRAM\_REQ;

initial begin

write\_reset <= 1'b0;

read\_reset <= 1'b0;

write\_enable <= 1'b1;

// CVM\_SYS\_REG\_N <= 1'b0;

CVM\_FRAM\_REQ <= 1'b0;

//Start <= 1'b0;

Raise\_REQ\_FLAG = 1'b0;

end

always @(posedge FSM\_Clk) begin

button\_reg <= ~button; // Grab the values from the button, complement and store them in register

if (Reset\_Counter[0] == 1'b1) State <= STATE\_RESET;

case (State)

STATE\_INIT: begin

if(counter < 21'd1) begin //300000

counter <= counter + 1'b1;

State <= STATE\_INIT;

end else

begin

if (Reset\_Counter[0] == 1'b1) // Write a delay before SYS\_RES gets raised

begin State <= STATE\_RESET; counter <= 21'b0; end

else

State <= STATE\_INIT;

end

end

STATE\_RESET: begin

if(counter < 21'd1) begin //300000

counter <= counter + 1'b1;

State <= STATE\_RESET;

end else

begin

//Start <= 1'b1;

//NEED delay before signaling SPI settings, raise SPI Flag

counter\_delay <= 0;

write\_reset <= 1'b1;

read\_reset <= 1'b1;

if (Reset\_Counter[0] == 1'b0) State <= STATE\_LOWER\_SPI;

end

end

STATE\_LOWER\_SPI: begin

// Start <= 1'b0;

State <= STATE\_RESET\_FINISHED;

end

STATE\_RESET\_FINISHED: begin //Go to another state to lower SPI FLAG

write\_reset <= 1'b0;

read\_reset <= 1'b0;

State <= STATE\_DELAY;

counter <= 21'd0;

end

STATE\_DELAY: begin //fifo resets

if (counter\_delay == 16'b0000\_0000\_0000\_1111) State <= STATE\_ENABLE\_WRITING; // if (counter\_delay == 16'b0000\_1111\_1111\_1111)

else counter\_delay <= counter\_delay + 1;

end

STATE\_ENABLE\_WRITING: begin //need delay to set FRAM REQ

if(counter < 21'd1) begin

counter <= counter + 1'b1;

State <= STATE\_ENABLE\_WRITING;

end else begin

Raise\_REQ\_FLAG <= 1'b1;

State <= STATE\_FRAME;

end

end

STATE\_FRAME : begin

Raise\_REQ\_FLAG <= 1'b0;

State <= STATE\_INIT;

end

endcase

end

reg OUT\_STATE = 1'b0;

always @(posedge FSM\_Clk)begin

case (OUT\_STATE)

1'b0: begin if(Raise\_REQ\_FLAG == 1'b1)begin

CVM\_FRAM\_REQ <= 1'b1;

OUT\_STATE <= 1'b1;

end else

OUT\_STATE <= 1'b0;

end

1'b1: begin

CVM\_FRAM\_REQ <= 1'b0;

OUT\_STATE <= 1'b0;

end

endcase

end

wire ACK\_bit, TrigerEvent;

wire SCL, SDA;

wire [7:0] I2C\_State;

wire [31:0] PC\_control;

wire [7:0] DeviceAddress, M;

wire [7:0] SubRegAddress;

wire [7:0] initializeRegisterValues;

wire [7:0] initializeRegisterAddress;

wire [15:0] tmp;

wire EN1;

wire [15:0] pulses;

fifo\_generator\_0 FIFO\_for\_Counter\_BTPipe\_Interface (

.wr\_clk(~CVM300\_CLK\_OUT),

.wr\_rst(write\_reset),

.rd\_clk(okClk),

.rd\_rst(read\_reset),

.din(CVM300\_D[9:2]),

.wr\_en(CVM300\_Data\_valid),

.rd\_en(FIFO\_read\_enable),

.dout(FIFO\_data\_out),

.full(FIFO\_full),

.empty(FIFO\_empty),

.prog\_full(FIFO\_BT\_BlockSize\_Full)

);

okBTPipeOut CounterToPC (

.okHE(okHE),

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

.ep\_addr(8'ha0),

.ep\_datain({FIFO\_data\_out[7:0], FIFO\_data\_out[15:8], FIFO\_data\_out[23:16], FIFO\_data\_out[31:24]}),

.ep\_read(FIFO\_read\_enable),

.ep\_blockstrobe(BT\_Strobe),

.ep\_ready(FIFO\_BT\_BlockSize\_Full)

);

I2C\_Transmit I2C\_Test1 (

//.led(led),

.sys\_clkn(sys\_clkn),

.sys\_clkp(sys\_clkp),

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

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

.PMOD\_A1(PMOD\_A1),

.PMOD\_A2(PMOD\_A2),

.FSM\_Clk(FSM\_Clk),

.ILA\_Clk(ILA\_Clk),

.Motor\_CLK(MotorClk),

.ACK\_bit(ACK\_bit),

.SCL(SCL),

.SDA(SDA),

.State(I2C\_State),

.PC\_control(PC\_control),

.DeviceAddress(DeviceAddress),

.SubRegAddress(SubRegAddress),

.initializeRegisterValues(initializeRegisterValues),

.initializeRegisterAddress(initializeRegisterAddress),

.cycles(pulses),

.value(tmp)

// .Mreturn(M),

//.EN1return(EN1)

);

okWireIn wire10 ( .okHE(okHE),

.ep\_addr(8'h00),

.ep\_dataout(Reset\_Counter));

//This is the OK host that allows data to be sent or recived

okWireIn wire11 ( .okHE(okHE),

.ep\_addr(8'h01),

.ep\_dataout(DeviceAddress));

okWireIn wire12 ( .okHE(okHE),

.ep\_addr(8'h02),

.ep\_dataout(SubRegAddress));

okWireIn wire13 ( .okHE(okHE),

.ep\_addr(8'h03),

.ep\_dataout(initializeRegisterValues));

okWireIn wire14 ( .okHE(okHE),

.ep\_addr(8'h04),

.ep\_dataout(initializeRegisterAddress));

okWireIn wire15 ( .okHE(okHE),

.ep\_addr(8'h05),

.ep\_dataout(pulses));

okWireIn wire16 ( .okHE(okHE),

.ep\_addr(8'h06),

.ep\_dataout(PMOD\_A2));

okWireIn wire17 ( .okHE(okHE),

.ep\_addr(8'h07),

.ep\_dataout(PC\_control[0]));

okWireOut wire20 ( .okHE(okHE),

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

.ep\_addr(8'h20),

.ep\_datain(tmp));

okWireOut wire21 ( .okHE(okHE),

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

.ep\_addr(8'h21),

.ep\_datain(M));

//EN1

//okWireOut wire22 ( .okHE(okHE),

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

// .ep\_addr(8'h22),

// .ep\_datain(EN1));

ila\_0 ila\_sample12 (

.clk(ILA\_Clk),

.probe0({CVM300\_D, CVM300\_Data\_valid, CVM300\_CLK\_IN, Out, State, CVM\_FRAM\_REQ, FSM\_Clk}),

.probe1({FSM\_Clk,CVM300\_Data\_valid })

);

endmodule

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"

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

from scipy import ndimage

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

import cv2

device\_manager = visa.ResourceManager()

devices = device\_manager.list\_resources()

number\_of\_device = len(devices)

from scipy import ndimage

import threading

#%%

# 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("")

ConfigStatus=dev.ConfigureFPGA("U:\\ece437\\test0\\CORRECTLABS\\lab9\\lab9.runs\\impl\_1\\BTPipeExample.bit"); # C:\\Users\\zinovya2\\ece437\\CORRECTLABS\\lab9\\lab9.runs\\impl\_1\\BTPipeExample.bit

# 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 ()

def twos\_comp(val, bits):

"""compute the 2's complement of int value val"""

if (val & (1 << (bits - 1))) != 0: # if sign bit is set e.g., 8bit: 128-255

val = val - (1 << bits) # compute negative value

return val # return positive value as is

#%%

#direction = 0 # toward to rod, clockwise

#direction = 1 # counter-clockwise

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns() # Update the WireIns

time.sleep(1)

bit\_num = 315392;

buf = bytearray(bit\_num); # 648\*488

buf1 = bytearray(bit\_num); # 648\*488

buf2 = bytearray(bit\_num); # 648\*488

img\_final = np.zeros([488,648,100]);

image = np.zeros(488\*648)

pix1 = 648;

pix2 = 488;

img1 = None; # np.zeros([488,648,2]);

im\_array1 = None;

#time.sleep(0.000001)

#ACC XCORD

DeviceAddress = int("00110010",2)

dev.SetWireInValue(0x01, DeviceAddress)

SubRegisterAddress = int("10101000",2) #("10101000",2) int("00100000",2)

dev.SetWireInValue(0x02, SubRegisterAddress)

initializeRegisterValues = int("10010111",2)

dev.SetWireInValue(0x03, initializeRegisterValues)

initializeRegisterAddress = int("00100000",2)

dev.SetWireInValue(0x04, initializeRegisterAddress)

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns()

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns() # Update the WireIns

dev.SetWireInValue(0x00, 1); #Reset FIFOs and counter

dev.UpdateWireIns(); # Update the WireIns

dev.SetWireInValue(0x00, 0); #Release reset signal

dev.UpdateWireIns(); # Update the WireIns

dev.ReadFromBlockPipeOut(0xa0, 1024, buf); # Read data from BT PipeOut

image[0:488\*646] = buf[0:488\*646]

#im\_array = np.array(buf[0:488\*646]).reshape(pix2, pix1)

im\_array = np.array(image).reshape(pix2, pix1) # Reshape array into a 2D array like image #np.max(image)

im\_array1 = im\_array.astype(np.uint8)

#cv2.imshow('gray image',im\_array1) #,origin='lower')

#cv2.waitKey(1)

bbox = cv2.selectROI(im\_array1,False)

tracker = cv2.legacy.TrackerMOSSE\_create()

tracker.init(im\_array1,bbox)

#%%

def acquire\_image():

global img1,im\_array1,tracker,ok,bbox

image = np.zeros(488\*648)

pix1 = 648;

pix2 = 488;

while True:

with threading.Lock(): #while True:

a = time.time()

dev.SetWireInValue(0x00, 1); #Reset FIFOs and counter

dev.UpdateWireIns(); # Update the WireIns

dev.SetWireInValue(0x00, 0); #Release reset signal

dev.UpdateWireIns(); # Update the WireIns

dev.ReadFromBlockPipeOut(0xa0, 1024, buf); # Read data from BT PipeOut

image[0:488\*646] = buf[0:488\*646]

#im\_array = np.array(buf[0:488\*646]).reshape(pix2, pix1)

im\_array = np.array(image).reshape(pix2, pix1) # Reshape array into a 2D array like image

im\_array1 = im\_array.astype(np.uint8)

# update tracker

ok,bbox = tracker.update(im\_array1)

if ok:

## calculate the center of the object

center0 = int(bbox[2]/2+bbox[0])

center1 = int(bbox[3]/2+bbox[1])

print('----------', center0,center1)

# each time, when doing rotation, re-initialize frame0

# print('Max difference value ',np.max(diff))

block\_size = 40

if center0 > 324+block\_size: #648/2 center\_of\_mass[1]

print("----------------------------------------------------")

print("rotate clockwise")

dev.SetWireInValue(0x07, 1)

dev.SetWireInValue(0x05, int(abs(center0/324-1)\*30)) # int(abs(center0/324-1)\*3)+

dev.SetWireInValue(0x06, 0)

dev.UpdateWireIns()

time.sleep(0.001)

dev.SetWireInValue(0x07, 0)

dev.UpdateWireIns()

print('Cycles',int(abs(center0/324-1)\*30))

#img1[:,:,0] = 0;

elif (center0 > 0) and (center0 < 324-block\_size): #center\_of\_mass[1]

print("----------------------------------------------------")

print("rotate counter-clockwise")

dev.SetWireInValue(0x07, 1)

dev.SetWireInValue(0x05, int(abs(center0/324-1)\*30)) # int(abs(center0/324-1)\*3)+

dev.SetWireInValue(0x06, 1)

dev.UpdateWireIns()

time.sleep(0.001)

dev.SetWireInValue(0x07, 0)

dev.UpdateWireIns()

print('Cycles ',int(abs(center0/324-1)\*30))

#img1[:,:,0] = 0;

##########################################################

for jj in range(5):

#a = time.time()

#time.sleep(0.000001)

#ACC XCORD

dev.SetWireInValue(0x01, int("00110010",2))

dev.SetWireInValue(0x02, int("10101000",2))

dev.SetWireInValue(0x07, 1)

dev.UpdateWireIns()

dev.SetWireInValue(0x07, 0)

dev.UpdateWireIns() # Update the WireIns

# 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(0.000001)

dev.UpdateWireOuts()

X\_ACC = dev.GetWireOutValue(0x20)

bin\_ = '{0:016b}'.format(X\_ACC)

X\_Decimal\_ACL = twos\_comp(int(float(((int(bin\_[-8:],2)<<8) + int(bin\_[:8],2)))), 16)/26000\*1.8;

SubRegisterAddress = int("10101010",2)

dev.SetWireInValue(0x02, SubRegisterAddress)

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns()

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns() # Update the WireIns

dev.UpdateWireOuts()

Y\_ACC = dev.GetWireOutValue(0x20)

bin\_ = '{0:016b}'.format(Y\_ACC)

Y\_Decimal\_ACL = twos\_comp(int(float(((int(bin\_[-8:],2)<<8) + int(bin\_[:8],2)))), 16)/26000\*1.8; #= twos\_comp(int(float(((MSB<<8) + LSB))), 16)/26000;

#time.sleep(0.000001)

#Z Cord ACCL

SubRegisterAddress = int("10101100",2)

dev.SetWireInValue(0x02, SubRegisterAddress)

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns()

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

dev.SetWireInValue(0x07, PC\_Control)

dev.UpdateWireIns() # Update the WireIns

dev.UpdateWireOuts()

Z\_ACC = dev.GetWireOutValue(0x20)

bin\_ = '{0:016b}'.format(Z\_ACC)

Z\_Decimal\_ACL = twos\_comp(int(float(((int(bin\_[-8:],2)<<8) + int(bin\_[:8],2)))), 16)/26000\*1.8; #twos\_comp(int(float(((MSB<<8) + LSB))), 16)/26000;

print("XYZ axis of Acclerator (g): %s %s %s" % (X\_Decimal\_ACL,Y\_Decimal\_ACL,Z\_Decimal\_ACL))

b = time.time()

print('Image acquisition Frame rate',1/(b-a))

###########################################################

time.sleep(0.0001)

def display\_image():

global im\_array1,ok,bbox

image = np.zeros(488\*648)

pix1 = 648;

pix2 = 488;

while True:

with threading.Lock():

a = time.time()

if im\_array1 is not None:

if ok:

p1 = (int(bbox[0]),int(bbox[1]))

p2 = (int(bbox[0]+bbox[2]),int(bbox[1]+bbox[3]))

cv2.rectangle(im\_array1,p1,p2,(255,255,0),2)

cv2.imshow('gray image',im\_array1) #,origin='lower')

cv2.waitKey(1)

time.sleep(0.035)

b = time.time()

print('Time Frame rate',1/(b-a))

acq\_thred = threading.Thread(target=acquire\_image)

display\_thred = threading.Thread(target=display\_image)

acq\_thred.start()

display\_thred.start()

`