* Post-lab question 1: If you transmit 10 bits per pixel, how much data will you transfer per frame?
  + (648 \* 488 pixels/frame) \* 10 bits/pixel = 3,162,240 bits / Frame = 395,280 Bytes / Frame
* Post-lab question 2: If you transmit 10 bits per pixel, what should be the length of the data transfer and block size to transfer one frame?
  + The length of the data transfer would be the 395,280 bytes and the block size to transfer would be an even division of the number for example 640\*648. In the class example they use 8 bits per pixel which leads to a block size of 1024 and a length of 1024 \* 4

# import various libraries necessary to run your Python code

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

import time

import numpy as np

from PIL import Image

import cv2

#%%

# 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

ConfigStatus=dev.ConfigureFPGA("BTPipeExample.bit");

# 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 ();

if ConfigStatus == 0:

print ("Your bit file is successfully loaded in the FPGA.");

else:

print ("Your bit file did not load. The error code number is:" + str(int(ConfigStatus)));

print ("Exiting the progam.");

sys.exit ();

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

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

buf = bytearray(315392)

#%% Write defaults to through SPI

#%%

data = 0x00;

addr = 0x01;

r\_w = 0x02;

start = 0x03;

read\_data = 0x20;

#---------------- Start of sending the default value to the Default Registers ---------------------#

dev.UpdateWireOuts()

Default\_Addr = [1,2,3,4,39,42,43,44,57,58,59,60,68,69,80,83,97,98,100,101,102,103,106,107,108,109,110,117]

Default\_Values = [232,1,0,0,1,100,3,0,3,44,240,10,2,9,2,187,240,10,112,98,34,64,94,110,91,82,80,91]

#----------------- Writes all of the default values -----------------#

for i in range(len(Default\_Addr)):

TEMP\_addr = Default\_Addr[i]

dev.SetWireInValue(addr, TEMP\_addr)

dev.SetWireInValue(r\_w, 1) #------- Sends Address and Value for Address ------#

TEMP\_VALUE = Default\_Values[i]

dev.SetWireInValue(data, TEMP\_VALUE)

dev.UpdateWireIns()

#time.sleep(.01)

dev.SetWireInValue(start, 1)

dev.UpdateWireIns()

dev.SetWireInValue(start, 0) #------ Starts the FSM -----------#

dev.UpdateWireIns()

#time.sleep(.01)

print(str(TEMP\_VALUE) + ' Written To: ' + str(TEMP\_addr))

#time.sleep(.01)

#------------ Reads all of the default values -------------#

print('Write Completed')

for i in range(len(Default\_Addr)):

TEMP\_addr = Default\_Addr[i]

dev.SetWireInValue(addr, TEMP\_addr)

dev.SetWireInValue(r\_w, 0)

dev.UpdateWireIns() #---------- Reads Register Values ------------#

#time.sleep(.01)

dev.SetWireInValue(start, 1)

dev.UpdateWireIns()

dev.SetWireInValue(start, 0)

dev.UpdateWireIns()

#time.sleep(.01)

dev.UpdateWireOuts()

TEMP\_VALUE = dev.GetWireOutValue(read\_data)

print('Register ' + str(TEMP\_addr) + ' value: ' + str(TEMP\_VALUE))

#time.sleep(0.1)

print('Done reading from registers!')

print('Reading all registers')

for i in range(127):

TEMP\_addr = i

dev.SetWireInValue(addr, TEMP\_addr)

dev.SetWireInValue(r\_w, 0)

dev.UpdateWireIns() #---------- Reads Register Values ------------#

#time.sleep(.01)

dev.SetWireInValue(start, 1)

dev.UpdateWireIns()

dev.SetWireInValue(start, 0)

dev.UpdateWireIns()

#time.sleep(.01)

dev.UpdateWireOuts()

TEMP\_VALUE = dev.GetWireOutValue(read\_data)

print('Register ' + str(TEMP\_addr) + ' value: ' + str(TEMP\_VALUE))

#time.sleep(.01)

print('Done reading from registers!')

#%%

dev.SetWireInValue(0x04, 1); #Reset FIFOs and counter

dev.UpdateWireIns(); # Update the WireIns

dev.SetWireInValue(0x04, 0); #Release reset signal

dev.UpdateWireIns(); # Update the WireIns

# dev.ReadFromBlockPipeOut(0xa0, 512 , buf); # Read data from BT PipeOut

#%%

buf = bytearray(315392\*4) #literally the closs common multiple to 486\*648

start = time.time()

dev.ReadFromBlockPipeOut(0xa0, 1024, buf)

stop = time.time()

pixel = np.arange(648\*486).reshape(486,648).astype('uint8')

for i in range (0,486,1):

for j in range (0,648,1):

pixel[i][j] = buf[i\*648\*4+j\*4]

show = Image.fromarray(pixel)

show.show()

print('time for single frame acq'+ str(stop-start) )

`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,

//SPI signaling

input CVM300\_SPI\_OUT,

output CVM300\_SPI\_EN,

output CVM300\_SPI\_IN,

output CVM300\_SPI\_CLK,

//Parallel CMOS out data

output CVM300\_FRAME\_REQ,

output CVM300\_SYS\_RES\_N,

output CVM300\_CLK\_IN,

input CVM300\_CLK\_OUT,

input [9:0] CVM300\_D,

input CVM300\_Line\_valid,

input CVM300\_Data\_valid,

//stuff for clock

input [3:0] button,

output [7:0] led,

input sys\_clkn,

input sys\_clkp

);

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)

);

wire [31:0] data\_in, addr\_in, r\_w, start\_fsm;

wire [7:0] data\_out;

reg sysreq, framereq;

assign CVM300\_SYS\_RES\_N = sysreq;

assign CVM300\_FRAME\_REQ = framereq;

//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 = 3;

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 = 24'd5;

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) );

assign CVM300\_CLK\_IN = FSM\_Clk;

//SPI stuff

SPI\_master SPI\_mod (.FSM\_clk(FSM\_Clk), .ADDR(addr\_in[7:0]),

.D\_IN(data\_in[7:0]), .MISO\_IN(CVM300\_SPI\_OUT), .R\_W(r\_w[0]),

.START\_FSM(start\_fsm[0]), .SPI\_CLK\_OUT(CVM300\_SPI\_CLK),

.MOSI\_OUT(CVM300\_SPI\_IN), .D\_OUT(data\_out),

.SPI\_EN(CVM300\_SPI\_EN), .READ\_DONE(read\_done),

.WRITE\_DONE(write\_done));

localparam STATE\_INIT = 8'd0;

localparam STATE\_RESET = 8'd1;

localparam STATE\_DELAY = 8'd2;

localparam STATE\_RESET\_FINISHED = 8'd3;

localparam FRAME\_BEGIN = 8'd4;

localparam FRAME\_END = 8'd5;

localparam STATE\_FINISH = 8'd6;

reg [31:0] counter = 8'd0;

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;

wire [31:0] Reset\_Counter;

wire [31:0] DATA\_Counter;

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

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

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;

initial begin

write\_reset <= 1'b0;

read\_reset <= 1'b0;

end

always @(negedge 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;

sysreq <= 1'b1;

case (State)

STATE\_INIT: begin

sysreq <= 1'b1;

if (Reset\_Counter[0] == 1'b1)

State <= STATE\_RESET;

end

STATE\_RESET: begin

counter <= 0;

counter\_delay <= 0;

write\_reset <= 1'b1;

read\_reset <= 1'b1;

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

end

STATE\_RESET\_FINISHED: begin

write\_reset <= 1'b0;

read\_reset <= 1'b0;

State <= STATE\_DELAY;

end

STATE\_DELAY: begin

if (counter\_delay == 16'b0000\_0000\_1111\_1111) State <= FRAME\_BEGIN;

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

end

FRAME\_BEGIN: begin

framereq <= 1'b1;

State <= FRAME\_END;

end

FRAME\_END: begin

framereq <= 1'b0;

State <= STATE\_FINISH;

end

STATE\_FINISH: begin

State <= STATE\_INIT;

end

endcase

end

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)

);

//Opal Kelly Stuff

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)

);

okWireIn wire10 ( .okHE(okHE),

.ep\_addr(8'h00),

.ep\_dataout(data\_in));

okWireIn wire11 ( .okHE(okHE),

.ep\_addr(8'h01),

.ep\_dataout(addr\_in));

okWireIn wire12 ( .okHE(okHE),

.ep\_addr(8'h02),

.ep\_dataout(r\_w));

okWireIn wire13 ( .okHE(okHE),

.ep\_addr(8'h03),

.ep\_dataout(start\_fsm));

okWireIn wire14 ( .okHE(okHE),

.ep\_addr(8'h04),

.ep\_dataout(Reset\_Counter));

//wire out stuff

okWireOut wire20 ( .okHE(okHE),

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

.ep\_addr(8'h20),

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

okWireOut wire21 ( .okHE(okHE),

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

.ep\_addr(8'h21),

.ep\_datain({30'd0,read\_done,write\_done}));

endmodule