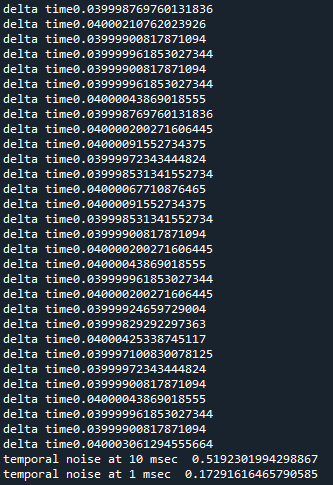
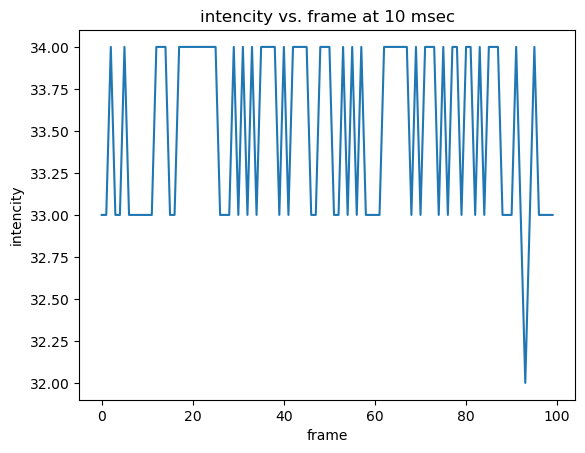
Postlab:

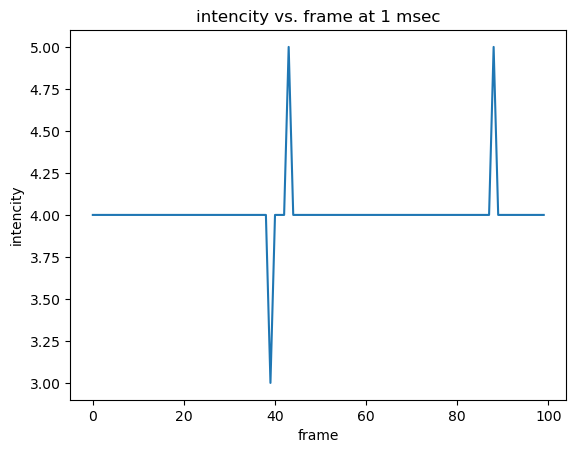


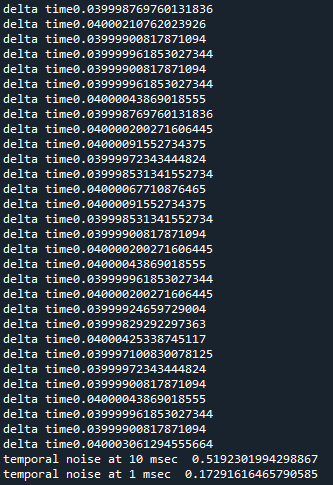
As we can see from the delta time for each frame reading, it takes less than 0.05 seconds for acquiring each frame, showing that we are able to acquire 20 frames per second from the image sensor.

Plot for pixel value at 50, 50 with 10 ms integration:



Plot for pixel value at 50, 50 with 1 ms integration:





The temporal noise is calculated by taking the standard deviation of the pixel value at 50,50 with 10 msec and 1 msec. As we can see, while the value for 10ms is far higher than 1ms, the standard deviation of 10ms is also higher than the 1ms since it integrate for a longer amount of time, introducing more noise and fluctuation in the reading.

Python:

lab10.py:

#%%

# import various libraries necessary to run your Python code

import time   # time related library

import sys,os    # system related library

import numpy as np

import matplotlib.pyplot as plt

import cv2

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

#%%

def reset\_image\_sensor():

    dev.SetWireInValue(0x01, 1)

    dev.UpdateWireIns()

    time.sleep(0.5)

    dev.SetWireInValue(0x01, 0)

    dev.UpdateWireIns()

    time.sleep(0.1)

#%%

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

    dev.SetWireInValue(0x00, 0)

    dev.UpdateWireIns()

    dev.SetWireInValue(0x02, reg\_addr)

    dev.SetWireInValue(0x03, value)

    dev.UpdateWireIns()  # Update the WireIns

    time.sleep(0.1)

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

    dev.UpdateWireIns()  # Update the WireIns

    time.sleep(0.1)

    dev.SetWireInValue(0x00, 0)

    dev.UpdateWireIns()  # Update the WireIns

#%%

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

    dev.SetWireInValue(0x00, 0)

    dev.UpdateWireIns()  # Update the WireIns

    time.sleep(0.1)

    dev.SetWireInValue(0x02, reg\_addr)

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

    dev.UpdateWireIns()  # Update the WireIns

    time.sleep(0.1)

    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

#%%

def setup\_image\_sensor():

    print("setting up...")

    reset\_image\_sensor()

    write\_to\_device(3, 8)

    write\_to\_device(4, 160)

    write\_to\_device(57, 3)

    write\_to\_device(58, 44)

    write\_to\_device(59, 240)

    write\_to\_device(60, 10)

    write\_to\_device(69, 9)

    write\_to\_device(80, 2)

    write\_to\_device(83, 187)

    write\_to\_device(97, 240)

    write\_to\_device(98, 10)

    write\_to\_device(100, 112)

    write\_to\_device(101, 98)

    write\_to\_device(102, 34)

    write\_to\_device(103, 64)

    write\_to\_device(106, 94)

    write\_to\_device(107, 110)

    write\_to\_device(108, 91)

    write\_to\_device(109, 82)

    write\_to\_device(110, 80)

    write\_to\_device(117, 91)

    print("setting up done")

#%%

def read\_a\_frame(HS\_counter):

    buf = bytearray(315392)

    dev.SetWireInValue(0x01, HS\_counter)

    dev.UpdateWireIns()

    dev.ReadFromBlockPipeOut(0xa0, 1024, buf)

    return buf

#%%

# 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 ()

#%% Reg and value constants

HS\_counter = 0

#%%

# Define the two variables that will send data to the FPGA

# We will use WireIn instructions to send data to the FPGA

time.sleep(1)

setup\_image\_sensor()

last\_time = 0

counter = 0

intencities1 = np.array([])

intencities2 = np.array([])

write\_to\_device(42,98)

write\_to\_device(43,2)

for counter in range(200):

    current\_time = time.time()

    print("delta time" + str(current\_time - last\_time))

    last\_time = current\_time

    HS\_counter = HS\_counter + 2

    buf = read\_a\_frame(HS\_counter)

    arr = np.frombuffer(buf, dtype=np.uint8, count=314928)

    arr = arr.reshape((486,648))

    if counter < 100:

        intencities1 = np.append(intencities1, arr[50][50])

    if counter == 99:

        write\_to\_device(42, 61)

        write\_to\_device(43, 0)

    if counter >= 100 and counter < 200:

        intencities2 = np.append(intencities2, arr[50][50])

    cv2.imshow("image",arr)

    cv2.waitKey(1)

plt.figure()

plt.plot(range(100), intencities1)

plt.title("intencity1 vs. frame")

plt.xlabel("frame")

plt.ylabel("intencity1")

plt.draw()

print("noise ", np.std(intencities1))

plt.figure()

plt.plot(range(100), intencities2)

plt.title("intencity2 vs. frame")

plt.xlabel("frame")

plt.ylabel("intencity2")

plt.draw()

plt.show()

print("noise ", np.std(intencities2))

#while (True):

#    current\_time = time.time()

#    print("delta time" + str(current\_time - last\_time))

#    last\_time = current\_time

#    HS\_counter = HS\_counter + 2

#    buf = read\_a\_frame(HS\_counter)

#    arr = np.frombuffer(buf, dtype=np.uint8, count=314928)

#    arr = arr.reshape((486,648))

#    cv2.imshow("image",arr)

#    cv2.waitKey(1)

#    plt.imshow(arr, cmap = 'gray')

#    plt.show()

#  # Read data from BT PipeOut

#for i in range (0, 1024, 1):

#   result = buf[i];

#   print (result)

dev.Close

Verilog: (same as lab 9)

Main.v:

`timescale 1ns / 1ps

module Main(

    output [7:0] led,

    input sys\_clkn,

    input sys\_clkp,

    input  CVM300\_CLK\_OUT,

    output CVM300\_CLK\_IN,

    output CVM300\_SYS\_RES\_N,

    output CVM300\_FRAME\_REQ,

    output CVM300\_SPI\_EN,

    output CVM300\_SPI\_CLK,

    input  CVM300\_SPI\_OUT,

    output CVM300\_SPI\_IN,

    input CVM300\_Line\_valid,

    input CVM300\_Data\_valid,

    input [9:0] CVM300\_D,

    input  [4:0] okUH,

    output [2:0] okHU,

    inout  [31:0] okUHU,

    inout  okAA

);

    wire clk;

    IBUFGDS osc\_clk(

        .O(clk),

        .I(sys\_clkp),

        .IB(sys\_clkn)

    );

    wire [31:0]PC\_rx;

    wire [31:0]PC\_tx;

    wire [31:0]PC\_command;

    wire [31:0]PC\_addr;

    wire [31:0]PC\_val;

    wire FIFO\_wr\_clk;

    wire FIFO\_wr\_enable;

    wire [31:0]FIFO\_data\_in;

    wire FIFO\_full;

    wire FIFO\_BT;

    wire FIFO\_read\_enable;

    wire FIFO\_read\_reset;

    wire FIFO\_write\_reset;

    wire USB\_ready;

    //PC communication/////////////////////////////////////////////////////////////////

    USB\_Driver USB\_Driver(

        .clk(clk),

        .okUH(okUH),

        .okHU(okHU),

        .okUHU(okUHU),

        .okAA(okAA),

        .PC\_rx(PC\_rx),

        .PC\_tx(PC\_tx),

        .PC\_command(PC\_command),

        .PC\_addr(PC\_addr),

        .PC\_val(PC\_val),

        .FIFO\_wr\_clk(FIFO\_wr\_clk),

        .FIFO\_read\_reset(FIFO\_read\_reset),

        .FIFO\_write\_reset(FIFO\_write\_reset),

        .FIFO\_wr\_enable(FIFO\_wr\_enable),

        .FIFO\_data\_in(FIFO\_data\_in),

        .FIFO\_full(FIFO\_full),

        .FIFO\_BT(FIFO\_BT),

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

        .USB\_ready(USB\_ready));

    // PC communication////////////////////////////////////////////////////////////////

    CVM300\_driver CVM300\_driver (

        .clk(clk),

        .CVM300\_CLK\_OUT(CVM300\_CLK\_OUT),

        .CVM300\_CLK\_IN(CVM300\_CLK\_IN),

        .CVM300\_SYS\_RES\_N(CVM300\_SYS\_RES\_N),

        .CVM300\_FRAME\_REQ(CVM300\_FRAME\_REQ),

        .CVM300\_SPI\_EN(CVM300\_SPI\_EN),

        .CVM300\_SPI\_CLK(CVM300\_SPI\_CLK),

        .CVM300\_SPI\_OUT(CVM300\_SPI\_OUT),

        .CVM300\_SPI\_IN(CVM300\_SPI\_IN),

        .CVM300\_LVAL(CVM300\_Line\_valid),

        .CVM300\_DVAL(CVM300\_Data\_valid),

        .CVM300\_D(CVM300\_D),

        .PC\_rx(PC\_rx),

        .PC\_tx(PC\_tx),

        .PC\_command(PC\_command),

        .PC\_addr(PC\_addr),

        .PC\_val(PC\_val),

        .FIFO\_wr\_clk(FIFO\_wr\_clk),

        .FIFO\_read\_reset(FIFO\_read\_reset),

        .FIFO\_write\_reset(FIFO\_write\_reset),

        .FIFO\_wr\_enable(FIFO\_wr\_enable),

        .FIFO\_data\_in(FIFO\_data\_in),

        .FIFO\_full(FIFO\_full),

        .FIFO\_BT(FIFO\_BT),

        .USB\_ready(USB\_ready));

    //Instantiate the ILA module

    ila\_0 ila\_sample12 (

        .clk(clk),

        .probe0({CVM300\_D,CVM300\_Line\_valid,CVM300\_Data\_valid,CVM300\_CLK\_OUT}),

        .probe1(CVM300\_FRAME\_REQ),

        .probe2(FIFO\_BT),

        .probe3(FIFO\_read\_enable),

        .probe4(FIFO\_wr\_enable),

        .probe5(USB\_ready));

endmodule

SPI.v:

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 2024/10/19 12:59:21

// Design Name:

// Module Name: SPI

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module SPI\_driver(

    input  wire clk,

    output reg [2:0] cur\_state,

    input wire  SPI\_MISO,

    output reg  SPI\_MOSI,

    output reg  SPI\_CLK,

    output reg  SPI\_EN,

    output reg busy,

    input wire command\_read,

    input wire rx\_read,

    input wire tx\_read,

    input wire [1:0] Spi\_rw,

    output reg [7:0] Spi\_rx\_reg,

    input wire [7:0] Spi\_tx\_reg

    );

localparam IDLE  = 3'b000;

localparam SPITX = 3'b001;

localparam SPIRX = 3'b010;

localparam SPIED = 3'b100;

reg [1:0] command\_FIFO[15:0];

reg [7:0] tx\_FIFO[15:0];

reg [7:0] rx\_FIFO[15:0];

reg [3:0] command\_addrw;

reg [3:0] command\_addrr;

reg [3:0] tx\_addrw;

reg [3:0] tx\_addrr;

reg [3:0] rx\_addrw;

reg [3:0] rx\_addrr;

wire command\_empty;

wire tx\_empty;

wire rx\_empty;

initial begin

    cur\_state = 7'd0;

    SPI\_MOSI = 1'b0;

    SPI\_CLK = 1'b0;

    SPI\_EN = 1'b0;

    busy = 1'b0;

    command\_addrw = 4'b0;

    command\_addrr = 4'd0;

    tx\_addrw = 4'd0;

    tx\_addrr = 4'd0;

    rx\_addrw = 4'd0;

    rx\_addrr = 4'd0;

end

assign tx\_empty = &(~(tx\_addrr^tx\_addrw));

assign rx\_empty = &(~(rx\_addrr^rx\_addrw));

assign command\_empty = &(~(command\_addrr^command\_addrw));

always @(posedge clk) begin

    if (command\_read == 1'b1) begin

        command\_FIFO[command\_addrw] <= Spi\_rw;

        command\_addrw <= command\_addrw + 1;

    end

    if (tx\_read == 1'b1) begin

        tx\_FIFO[tx\_addrw] <= Spi\_tx\_reg;

        tx\_addrw <= tx\_addrw + 1;

    end

    if (rx\_read == 1'b1) begin

        if(rx\_empty != 1'b1) begin

            Spi\_rx\_reg <= rx\_FIFO[rx\_addrr];

            rx\_addrr <= rx\_addrr + 1;

        end else begin

            Spi\_rx\_reg <= 8'b11111111;

        end

    end

end

reg[2:0] bit\_counter;

reg[2:0] clk\_counter;

reg[7:0] rx\_temp\_reg;

always @(posedge clk) begin

        case(cur\_state)

            IDLE : begin

                busy <= 1'b0;

                if(command\_empty == 1'b0)begin

                    if(command\_FIFO[command\_addrr] == 2'b01)begin

                        command\_addrr  <= command\_addrr + 1;

                        cur\_state      <= SPITX;

                        clk\_counter    <= 3'b000;

                        bit\_counter    <= 3'b111;

                        busy <= 1'b1;

                    end

                    //add error detection if the first command out of IDLE is rx, this is incorrectly set by the controller

                    // also add error detection if when entering TX, check for TX\_FIFO empty, if not, controller is incorrectly set

                end

            end

            SPITX: begin

                case(clk\_counter)

                    3'b000 : begin

                        SPI\_EN   <= 1'b1;

                        SPI\_CLK  <= 1'b0;

                        SPI\_MOSI <= tx\_FIFO[tx\_addrr][bit\_counter];

                        busy <= 1'b1;

                        clk\_counter <= clk\_counter + 1;

                    end

                    3'b100 : begin

                        SPI\_CLK <= 1'b1;

                        clk\_counter <= clk\_counter + 1;

                    end

                    3'b111 : begin

                        if(bit\_counter != 3'b000) begin

                            bit\_counter <= bit\_counter -1;

                            clk\_counter <= 3'b000;

                        end else begin

                            bit\_counter <= 3'b111;

                            clk\_counter <= 3'b000;

                            tx\_addrr <= tx\_addrr + 1;

                            if(command\_empty == 1'b1)cur\_state   <= SPIED;

                            else begin

                                if (command\_FIFO[command\_addrr] == 2'b01) cur\_state <= SPITX;

                                else cur\_state <= SPIRX;

                                command\_addrr <= command\_addrr + 1;

                                //add error detection if 2'b10 or 2'b01 is not the data read from the FIFO

                            end

                        end

                    end

                    default : begin

                        clk\_counter <= clk\_counter + 1;

                    end

                endcase

            end

            SPIRX: begin

                case(clk\_counter)

                    3'b000 : begin

                        SPI\_EN  <= 1'b1;

                        SPI\_CLK <= 1'b0;

                        clk\_counter <= clk\_counter + 1;

                    end

                    3'b100 : begin

                        SPI\_CLK  <= 1'b1;

                        rx\_temp\_reg[bit\_counter] <= SPI\_MISO;

                        clk\_counter <= clk\_counter +1;

                    end

                    3'b111 : begin

                        if(bit\_counter != 3'b000) begin

                            bit\_counter <= bit\_counter - 1;

                            clk\_counter <= 3'b000;

                        end else begin

                            bit\_counter <= 3'b111;

                            clk\_counter <= 3'b000;

                            rx\_FIFO[rx\_addrw] <= rx\_temp\_reg;

                            rx\_addrw <= rx\_addrw + 1;

                            if(command\_empty == 1'b1) cur\_state <= SPIED;

                            else begin

                                if(command\_FIFO[command\_addrr] == 2'b01) cur\_state <= SPITX;

                                else cur\_state <= SPIRX;

                                command\_addrr <= command\_addrr + 1;

                            end

                        end

                    end

                    default : begin

                        clk\_counter <= clk\_counter + 1;

                    end

                endcase

            end

            SPIED : begin

                case(clk\_counter)

                    3'b000 : begin

                        SPI\_EN  <=1'b1;

                        SPI\_CLK <=1'b0;

                        clk\_counter <= clk\_counter + 1;

                    end

                    3'b100 : begin

                        SPI\_EN   <= 1'b0;

                        SPI\_CLK  <= 1'b0;

                        SPI\_MOSI <= 1'b0;

                        clk\_counter <= 3'b000;

                        cur\_state <= IDLE;

                    end

                    default : begin

                        clk\_counter <= clk\_counter + 1;

                    end

                endcase

            end

        endcase

end

endmodule

SPI\_controller.v:

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 2024/09/22 14:34:42

// Design Name:

// Module Name: TS\_controller

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module SPI\_controller(

    input clk,

    input wire [31:0] PC\_rx,

    input wire [31:0] PC\_addr,

    input wire [31:0] PC\_val,

    output reg [31:0] PC\_tx,

    output reg command\_read,

    output reg rx\_read,

    output reg tx\_read,

    output reg [7:0] tx\_byte,

    output reg [1:0] rw,

    output reg [9:0] cur\_state,

    input wire [7:0] rx\_byte,

    input wire busy

    );

    reg busy\_reg = 0;

    reg [7:0] tx\_byte\_reg;

    reg [7:0] rx\_byte\_reg;

    reg [2:0] read\_counter;

    reg [31:0] PC\_rx\_reg1, PC\_rx\_reg2;

    localparam idle\_     = 10'b0000000001;

    localparam start\_wr  = 10'b0000000010;

    localparam tx\_wr1    = 10'b0000000100;

    localparam tx\_wr2    = 10'b0000001000;

    localparam end\_wr    = 10'b0000010000;

    localparam start\_rt  = 10'b0000100000;

    localparam tx\_rt     = 10'b0001000000;

    localparam rx\_rt     = 10'b0010000000;

    localparam wait\_rt   = 10'b0100000000;

    localparam end\_rt    = 10'b1000000000;

    initial begin

        cur\_state <= idle\_;

        PC\_rx\_reg1 <= 0;

        PC\_rx\_reg2 <= 0;

        tx\_byte\_reg <= 0;

        rx\_byte\_reg <= 0;

        busy\_reg <= 1'b0;

        read\_counter <= 0;

    end

    integer i;

    always @(posedge clk) begin

        for (i=0; i<8; i=i+1) begin

            tx\_byte[i] <= tx\_byte\_reg[i];

            rx\_byte\_reg[i] <= rx\_byte[i];

        end

    end

    always @(posedge clk) begin

        case (cur\_state)

            idle\_ : begin

                command\_read <= 1'b0;

                tx\_read <= 1'b0;

                rx\_read <= 1'b0;

                PC\_rx\_reg1 <= PC\_rx;

                PC\_rx\_reg2 <= PC\_rx\_reg1;

                if (PC\_rx\_reg2[0] == 1'b0 && PC\_rx\_reg1[0] == 1'b1) begin

                    cur\_state <= start\_wr;

                end

                if (PC\_rx\_reg2[1] == 1'b0 && PC\_rx\_reg1[1] == 1'b1) begin

                    cur\_state <= start\_rt;

                end

            end

            //Write single byte

            start\_wr: begin

                tx\_byte\_reg <= {1'b1, PC\_addr[6:0]};

                cur\_state <= tx\_wr1;

            end

            tx\_wr1: begin

                tx\_byte\_reg <= PC\_val[7:0];

                command\_read <= 1'b1;

                rw <= 2'b01;

                tx\_read <= 1'b1;

                cur\_state <= tx\_wr2;

            end

            tx\_wr2: begin

                command\_read <= 1'b1;

                rw <= 2'b01;

                tx\_read <= 1'b1;

                cur\_state <= end\_wr;

            end

            end\_wr : begin

                tx\_byte\_reg <= {8{1'b0}};

                command\_read <= 1'b0;

                tx\_read <= 1'b0;

                cur\_state <= idle\_;

            end

            //Read two byte

            start\_rt: begin

                tx\_byte\_reg <= {1'b0, PC\_addr[6:0]};

                cur\_state <= tx\_rt;

            end

            tx\_rt: begin

                command\_read <= 1'b1;

                rw <= 2'b01;

                tx\_read <= 1'b1;

                cur\_state <= rx\_rt;

            end

            rx\_rt : begin

                command\_read <= 1'b1;

                tx\_read <= 1'b0;

                rw <= 2'b10;

                cur\_state <= wait\_rt;

            end

            wait\_rt : begin

                tx\_byte\_reg <= {8{1'b0}};

                command\_read <= 1'b0;

                rw <= 2'b00;

                busy\_reg <= busy;

                if (busy\_reg == 1'b1 && busy == 1'b0) begin

                    rx\_read <= 1'b1;

                    cur\_state <= end\_rt;

                end

            end

            end\_rt: begin

                rx\_read <= 1'b0;

                read\_counter <= read\_counter + 1;

                if (read\_counter == 2) begin

                    PC\_tx[7:0] <= rx\_byte\_reg;

                    read\_counter <= 0;

                    cur\_state <= idle\_;

                end

            end

            default : begin

                tx\_byte\_reg <= {8{1'b0}};

                cur\_state <= idle\_;

            end

         endcase

     end

endmodule

USB\_Driver.v:

`timescale 1 ps / 1 ps

module USB\_Driver(

    input clk,

    input   wire    [4:0] okUH,

    output  wire    [2:0] okHU,

    inout   wire    [31:0] okUHU,

    inout   wire    okAA,

    output [31:0]PC\_rx,

    input  [31:0]PC\_tx,

    output [31:0]PC\_command,

    output [31:0]PC\_addr,

    output [31:0]PC\_val,

    input   FIFO\_wr\_clk,

    input   FIFO\_read\_reset,

    input   FIFO\_write\_reset,

    input   FIFO\_wr\_enable,

    input   [31:0]FIFO\_data\_in,

    output  FIFO\_full, // currently not used

    output  FIFO\_BT,

    output FIFO\_read\_enable,

    input USB\_ready

    );

    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 = 2;

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

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

    //Wire In/////////////////////////////

    okWireIn wire10 (   .okHE(okHE),

                        .ep\_addr(8'h00),

                        .ep\_dataout(PC\_rx));

    okWireIn wire11 (   .okHE(okHE),

                        .ep\_addr(8'h01),

                        .ep\_dataout(PC\_command));

    okWireIn wire12 (   .okHE(okHE),

                        .ep\_addr(8'h02),

                        .ep\_dataout(PC\_addr));

    okWireIn wire13 (   .okHE(okHE),

                        .ep\_addr(8'h03),

                        .ep\_dataout(PC\_val));

    //Wire Out//////////////////////////////////

    okWireOut wire20 (  .okHE(okHE),

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

                        .ep\_addr(8'h20),

                        .ep\_datain(PC\_tx));

    wire [31:0] FIFO\_data\_out;

    wire prog\_full;

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

        .wr\_clk(FIFO\_wr\_clk),

        .wr\_rst(FIFO\_write\_reset),

        .rd\_clk(okClk),

        .rd\_rst(FIFO\_read\_reset),

        .din(FIFO\_data\_in[9:2]),

        .wr\_en(FIFO\_wr\_enable),

        .rd\_en(FIFO\_read\_enable),

        .dout(FIFO\_data\_out),

        .full(FIFO\_full),

        .prog\_full(prog\_full),

        .empty(FIFO\_empty)

    );

    okBTPipeOut CounterToPC (

        .okHE(okHE),

        .okEH(okEHx[ 1\*65 +: 65 ]),

        .ep\_addr(8'ha0),

        .ep\_datain(FIFO\_data\_out),

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

        .ep\_blockstrobe(FIFO\_BT),

        .ep\_ready(prog\_full)

    );

endmodule

CVM300\_driver.v:

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 2024/11/03 11:05:11

// Design Name:

// Module Name: CVM300\_driver

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module CVM300\_driver(

    input clk,

    input  CVM300\_CLK\_OUT,

    output CVM300\_CLK\_IN,

    output CVM300\_SYS\_RES\_N,

    output CVM300\_FRAME\_REQ,

    output CVM300\_SPI\_EN,

    output CVM300\_SPI\_CLK,

    input  CVM300\_SPI\_OUT,

    output CVM300\_SPI\_IN,

    input  CVM300\_LVAL,

    input  CVM300\_DVAL,

    input [9:0] CVM300\_D,

    input  [31:0]PC\_rx,

    output [31:0]PC\_tx,

    input  [31:0]PC\_command,

    input  [31:0]PC\_addr,

    input  [31:0]PC\_val,

    output  FIFO\_wr\_clk,

    output  FIFO\_read\_reset,

    output  FIFO\_write\_reset,

    output  FIFO\_wr\_enable,

    output  [31:0]FIFO\_data\_in,

    input   FIFO\_full, // currently not used

    input   FIFO\_BT,

    output USB\_ready

    );

    wire [3:0] SPI\_state;

    wire busy;

    wire command\_read;

    wire rx\_read;

    wire tx\_read;

    wire [1:0] Spi\_rw;

    wire [7:0] Spi\_rx\_reg;

    wire [7:0] Spi\_tx\_reg;

    wire [9:0] controller\_state;

    //SPI SERDES

    SPI\_driver SPI\_driver(

    .clk(clk),

    .cur\_state(SPI\_state),

    .SPI\_MISO(CVM300\_SPI\_OUT),

    .SPI\_MOSI(CVM300\_SPI\_IN),

    .SPI\_CLK(CVM300\_SPI\_CLK),

    .SPI\_EN(CVM300\_SPI\_EN),

    .busy(busy),

    .command\_read(command\_read),

    .rx\_read(rx\_read),

    .tx\_read(tx\_read),

    .Spi\_rw(Spi\_rw),

    .Spi\_rx\_reg(Spi\_rx\_reg),

    .Spi\_tx\_reg(Spi\_tx\_reg)

    );

    //SPI controller

    SPI\_controller SPI\_controller(

    .clk(clk),

    .PC\_rx(PC\_rx),

    .PC\_addr(PC\_addr),

    .PC\_val(PC\_val),

    .PC\_tx(PC\_tx),

    .command\_read(command\_read),

    .rx\_read(rx\_read),

    .tx\_read(tx\_read),

    .rw(Spi\_rw),

    .tx\_byte(Spi\_tx\_reg),

    .rx\_byte(Spi\_rx\_reg),

    .busy(busy),

    .cur\_state(controller\_state)

    );

    // Reset control////////////////////////////////////////////////////////////////

    // Clock generation//////////////////////////////////////////////////////////////////

    reg CVM\_Clk;

    reg [23:0] ClkDivCVM = 24'd0;

    assign CVM300\_CLK\_IN = CVM\_Clk;

    always @(posedge clk) begin

        if (ClkDivCVM == 4) begin

            CVM\_Clk <= !CVM\_Clk;

            ClkDivCVM <= 0;

        end else begin

            ClkDivCVM <= ClkDivCVM + 1'b1;

        end

    end

    // Frame request && Reset set/////////////////////////

    reg reset = 1;

    reg started = 0;

    reg frame\_request = 0;

    reg [15:0] HS\_counter = 0;

    always @(posedge CVM\_Clk) begin

        if (PC\_command[0] == 1) begin

            reset <= 1;

            started <= 1;

            HS\_counter <= 16'd1;

        end

        else begin

            if (started) reset <= 0;

            else reset <= 1;

        end

        if (PC\_command[16:1] == HS\_counter && started == 1'b1) begin

            HS\_counter <= HS\_counter + 1;

            frame\_request <= 1'b1;

        end

        else begin

            frame\_request <= 1'b0;

        end

    end

    assign CVM300\_SYS\_RES\_N = ~reset;

    assign CVM300\_FRAME\_REQ = frame\_request;

    assign FIFO\_read\_reset = frame\_request;

    assign FIFO\_write\_reset = frame\_request;

    reg[31:0] FIFO\_data\_in\_reg;

    reg FIFO\_ready\_reg;

    reg FIFO\_wrena\_reg;

    reg read\_flag=1'b0;

    reg CMV\_clk\_CDC;

    reg CMV\_clk;

    reg CMV\_clk\_reg;

    always @(posedge clk) begin

        CMV\_clk\_CDC <= CVM300\_CLK\_OUT;

        CMV\_clk <= CMV\_clk\_CDC;

        CMV\_clk\_reg <= CMV\_clk;

        if (CMV\_clk ==1'b0 && CMV\_clk\_reg == 1'b1) begin

            if (CVM300\_LVAL == 1'b1 && CVM300\_DVAL == 1'b1) begin

                FIFO\_data\_in\_reg[9:0] <= CVM300\_D;

                FIFO\_data\_in\_reg[31:10] <= 0;

                read\_flag <= 1'b1;

                FIFO\_wrena\_reg <= 1'b1;

            end

        end

        else FIFO\_wrena\_reg <= 1'b0;

        if (CVM300\_LVAL == 1'b0 && CVM300\_DVAL == 1'b0) begin

            if (read\_flag == 1'b1) FIFO\_ready\_reg <= 1'b1;

            read\_flag <= 1'b0;

        end

        if (FIFO\_BT == 1'b1) FIFO\_ready\_reg <= 1'b0;

    end

    assign FIFO\_data\_in = FIFO\_data\_in\_reg;

    assign FIFO\_wr\_enable = FIFO\_wrena\_reg;

    assign FIFO\_wr\_clk = clk;

    assign USB\_ready = FIFO\_ready\_reg;

endmodule