Post Lab Question by Ethan Zhou (yz69) & Qiran Pang (qpang2)

1.1 Measured Performance (CPS = Calls Per Second)

API CALL	USB 3.0 (CPS)
UpdateWireIns	5,000+
UpdateWireOuts	4,000+
ActivateTriggerIn	8,000+
UpdateTriggerOuts	4,000+

Question 1.

From documentation, we know the calls per second of UpdateWireOuts for USB 3.0 is 4000+, therefore, with 32 bits width for each WireOut, it can transmit 16000+ Bytes/s.

Question 2.

From documentation, we know the calls per second of UpdateWireIns for USB 3.0 is 5000+, therefore, with 32 bits width for each WireOut, it can transmit 20000+ Bytes/s.

Question 3.

Utilization — Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	2194	47200	4.65
LUTRAM	243	19000	1.28
FF	3372	94400	3.57
BRAM	6	105	5.71
IO	51	285	17.89
BUFG	4	32	12.50
MMCM	1	6	16.67

The design utilizes more resources from every category, this is to accommodate the implementation of logics and registers related to the interface.

Question 4.

The maximum allowed result value would be 4294967295, thus to ensure the result is accurate, variable_1 and variable_2 should not exceed half of this which would be 2147483647. If the maximum value is exceeded, the result over overflow (subtracts 4294967295 from the arithmetic result)

Question 5.

Verilog : `timescale 1ns / 1ps

```
module lab2_example(
    input   wire    [4:0] okUH,
    output  wire    [2:0] okHU,
    inout   wire    [31:0] okUHU,
    inout   wire    okAA,
    input   wire    sys_clkn,
    input   wire    sys_clkp,
    input   wire    reset,
    // Your signals go here
    input [3:0] button,
    output reg [7:0] led
);

  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

  //Declare your registers or wires to send or recieve data
  wire [31:0] variable_1, variable_2;      //signals that are outputs from a module must be wires
  wire [31:0] result_wire;              //signals that go into modules can be wires or registers
  reg  [31:0] result_register;          //signals that go into modules can be wires or registers
```

```
//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 2 output endpoints, hence endPt_count = 2.
localparam  endPt_count = 2;
wire [endPt_count*65-1:0] okEHx;
okWireOR # (.N(endPt_count)) wireOR (okEH, okEHx);

// Clock
wire clk;
reg [31:0] clkdiv;
reg [31:0] div_var;
reg slow_clk;
reg [7:0] counter;

IBUFGDS osc_clk(
```

```
    .O(clk),
    .I(sys_clkp),
    .IB(sys_clkn)
);

initial begin
    clkdiv = 0;
    slow_clk = 0;
end

//ILA probes
ila_0 ila(
    .clk(clk),
    .probe0(variable_1),
    .probe1(variable_2),
    .probe2(result_wire),
    .probe3(result_register)
    );

// This code creates a slow clock from the high speed Clk signal
// You will use the slow clock to run your finite state machine
// The slow clock is derived from the fast 200 MHz clock by dividing it 10,000,000 time and another 2x
// Hence, the slow clock will run at 10 Hz
always @(posedge clk) begin
    clkdiv <= clkdiv + 1'b1;
    if (clkdiv == div_var) begin
        slow_clk <= ~slow_clk;
        clkdiv <= 0;
```

```
   end
end

always @ (posedge clk) begin
   div_var <= variable_2;
   case (variable_1)
      0 : begin
         led <= {8{1'b1}};
         end
      1 : begin
         led <= {8{1'b0}};
         end
      default: begin
         led <= ~counter;
         end
      endcase
    end


//The main code will run fr0m the slow clock.  The rest of the code will be in this section.
//The counter will decrement when button 0 is pressed and on the rising edge of the slow clk
//Otherwise the counter will increment
always @(posedge slow_clk) begin
   case (variable_1)
      2: begin
         counter <= counter + 2;
         end
      3: begin
         counter <= counter - 2;
```

```
            end
        5: begin
            counter <= 0;
            end
        default: begin
            counter <= counter;
            end
        endcase
    end

    //  variable_1 is a wire that contains data sent from the PC to FPGA.
    //  The data is communicated via memeory location 0x00
    okWireIn wire10 (   .okHE(okHE),
                        .ep_addr(8'h00),
                        .ep_dataout(variable_1));

    //  variable_2 is a wire that contains data sent from the PC to FPGA.
    //  The data is communicated via memeory location 0x01
    okWireIn wire11 (   .okHE(okHE),
                        .ep_addr(8'h01),
                        .ep_dataout(variable_2));

    // Variable 1 and 2 are added together and the result is stored in a wire named: result_wire
    // Since we are using a wire to store the result, we do not need a clock signal and
    // we will use an assign statement
    assign result_wire = variable_1 + variable_2;    // Left-Side of 'assign' statement must be a
'wire'

    // result_wire is transmited to the PC via address 0x20
```

```
   okWireOut wire20 (  .okHE(okHE),
                       .okEH(okEHx[ 0*65 +: 65 ]),
                       .ep_addr(8'h20),
                       .ep_datain(result_wire));

   // Variable 1 and 2 are subtracted and the result is stored in a register named: result_register
   // Since we are using a register to store the result, we not need a clock signal and
   // we will use an always statement examening the clock state
   always @ (posedge(slow_clk)) begin
      result_register = counter;
   end

   // result_wire is transmited to the PC via address 0x21
   okWireOut wire21 (  .okHE(okHE),
                       .okEH(okEHx[ 1*65 +: 65 ]),
                       .ep_addr(8'h21),
                       .ep_datain(result_register));
Endmodule
```

```
Python : # -*- coding: utf-8 -*-

#%%
# import various libraries necessary to run your Python code
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"
```

```
sys.path.append(ok_sdk_loc)   # add the path of the OK library
os.add_dll_directory(ok_dll_loc)

import ok     # OpalKelly library

#%%
# 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("../FPGA/bit/lab2_example.bit"); # Configure the FPGA with
this bit file

# 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("--------------------------------------------------")
#%%
control_variable = 2; # control_variable is initialized to digital number 2
clock_divider = 10000000; # # clock_divider is initialized to digital number 10000000
while(True):
    dev.UpdateWireOuts()
    counter = dev.GetWireOutValue(0x21)  # Transfer the received data in result_sum variable
    # result_difference = dev.GetWireOutValue(0x21)  # Transfer the received data in
result_difference variable
    print("The counter value is " + str(int(counter)))
    # print("The difference between the two numbers is " + str(int(result_difference)))

    print("clock_divider is initialized to " + str(int(clock_divider)))
    if counter == 100:
        print("control_variable is initialized to " + str(5))
        dev.SetWireInValue(0x00, 5) #Input data for Variable 1 using memory space 0x00
        dev.UpdateWireIns()  # Update the WireIns
        time.sleep(0.5)

    else:
        print("control_variable is initialized to " + str(int(control_variable)))
        dev.SetWireInValue(0x00, control_variable) #Input data for Variable 1 using memory space
0x00
    dev.SetWireInValue(0x01, clock_divider) #Input data for Variable 2 using memory space 0x01
    dev.UpdateWireIns()  # Update the WireIns
    time.sleep(0.05)
```