Name: Junzhe Wu Net id: Junzhew3

1. How much data can be sent per second using okWireOut?
   32*4000(CPS)*32(bit)/8/1000 = 512KB
2. How much data can be sent per second using okWireIn?
   32*5000(CPS)*32(bit)/8/1000 = 640KB
3. Look at the Project Summary window. How many resources on the FPGA are used to implement your code? How does this compare to lab 1?

Resource	Utilization	Available	Utilization%
LUT	846	47200	1.79
LUTRAM	32	19000	0.17
FF	807	94400	0.85
BRAM	2	105	1.90
IO	55	285	19.3
BUFG	3	32	9.38
MMCM	1	6	16.67

   (1) We did not use LUTRAM, BRAM and MMCM in the lab1.
   (2) We use more resources of LUT, FF, IO and BUFG in the lab2, especially LUT and FF.
4. What is the maximum allowed values for both variable_1 and variable_2 such that the addition of these two number is correct using the example code provided in this lab? What happens if you go over these maximum values? Double check this in your code to make sure your answer is correct.
   (1) The **result_wire** which store the addition of **variable_1** and **variable_2** is an 32-bit wire. The maximum allowed value of **result_wire** = $2^{32} - 1$ = 4294967295, so the maximum allowed value of the addition of **variable_1** and **variable_2** is 4294967295.
   (2) If I go over the maximum value, the **result_wire** can not store the value of the addition anymore and it will generate some errors in the running time.
5. Include a printout of both codes with the final report.

(1) Verilog code:

```
`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 [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 = 1;
    wire [endPt_count*65-1:0] okEHx;
    okWireOR # (.N(endPt_count)) wireOR (okEH, okEHx);

    // Clock
    wire clk;
    wire [33:0] clkdiv1;
    wire [1:0] reset0;
    reg [1:0] reset1;
    reg [33:0] clkdiv;
    reg slow_clk;
    reg [7:0] counter;

    IBUFGDS osc_clk(
```

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

    initial begin
        clkdiv = 0;
        reset1=1;
        slow_clk = 0;
    end

    // 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 == clkdiv1) begin
            slow_clk <= ~slow_clk;
            clkdiv <= 0;
        end
    end

    assign led = ~counter;
    //The main code will run fr0m the slow clock.  The rest of the code
will be in this section.
    //The counter will increment when button 0 is pressed and on the
rising edge of the slow clk
    //The counter will decrement when button 0 is pressed and on the
rising edge of the slow clk


    always @(posedge slow_clk) begin
        if (reset0==1'b1) begin
            counter <= 8'h00;
        end
        else if (button [2] == 1'b0 && button [3] == 1'b1 && button [0]
== 1'b1 && button [1] == 1'b1 && reset0 == 1'b0) begin
            counter <= counter + 2'b10;
        end
        else if (button [3] == 1'b0  && button [2] == 1'b1 && button [0]
== 1'b1 && button [1] == 1'b1 && reset0 == 1'b0) begin
            counter <= counter - 2'b10;
```

```
        end
        else if (button [0] == 1'b0  && button [3] == 1'b1 && button [2]
== 1'b1 && button [1] == 1'b1 && reset0 == 1'b0) begin
            counter <= 8'hff;
        end
        else if (button [1] == 1'b0  && button [3] == 1'b1 && button [0]
== 1'b1 && button [2] == 1'b1 && reset0 == 1'b0) begin
            counter <= 8'h00;
        end
    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(clkdiv1));

    //  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(reset0));


    // 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'

    // 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 = variable_1 - variable_2;
       result_register=counter;
    end
```

```
    // 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_register));


    // 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
```

(2) Python code:

```
# import various libraries necessery to run your Python code
import time   # time related library
import sys    # system related library
ok_loc = 'C:\\Program Files\\Opal
Kelly\\FrontPanelUSB\\API\\Python\\3.6\\x64'
sys.path.append(ok_loc)   # add the path of the OK library
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 communicaiton with the
OK board
ConfigStatus=dev.ConfigureFPGA("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("-----------------------------------------------------")

#%%
# Define the two variables that will send data to the FPGA
# We will use WireIn instructions to send data to the FPGA

clkdiv = 100000
print("clkdiv is initilized to " + str(int(clkdiv)))
dev.SetWireInValue(0x00, clkdiv) #Input data for Variable 1 using
mamoery spacee 0x00
dev.UpdateWireIns()  # Update the WireIns

'''
variable_2 = 14; # # variable_2 is initilized to digitla number 14
print("Variable 2 is initilized to " + str(int(variable_2)))

dev.SetWireInValue(0x01, variable_2) #Input data for Variable 2 using
mamoery spacee 0x01

#%%
# We will read data from the FPGA in two different variables
# Since we are using a slow clock on the FPGA to compute the results
# we need to wait for the resutl to be computed
#time.sleep(0.5)
'''

# First recieve data from the FPGA by using UpdateWireOuts

dev.UpdateWireOuts()
pre_result = dev.GetWireOutValue(0x20)  # Transfer the recived data in
result_sum variable
#result_difference = dev.GetWireOutValue(0x21)  # Transfer teh recived
data in result_difference variable
print("The counter is " + str(int(pre_result)))
#print("The difference between the two numbers is " +
str(int(result_difference)))
```

```
while 1:
    reset=0
    dev.UpdateWireOuts()
    result=dev.GetWireOutValue(0x20)
    if result!=pre_result:
        print("The counter is " + str(int(result)))
        pre_result=result
    if result>=100:
        reset = 1;
        #print("reset counter")
    dev.SetWireInValue(0x01, reset)
    dev.UpdateWireIns()
    time.sleep(0.1)



dev.Close
```