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

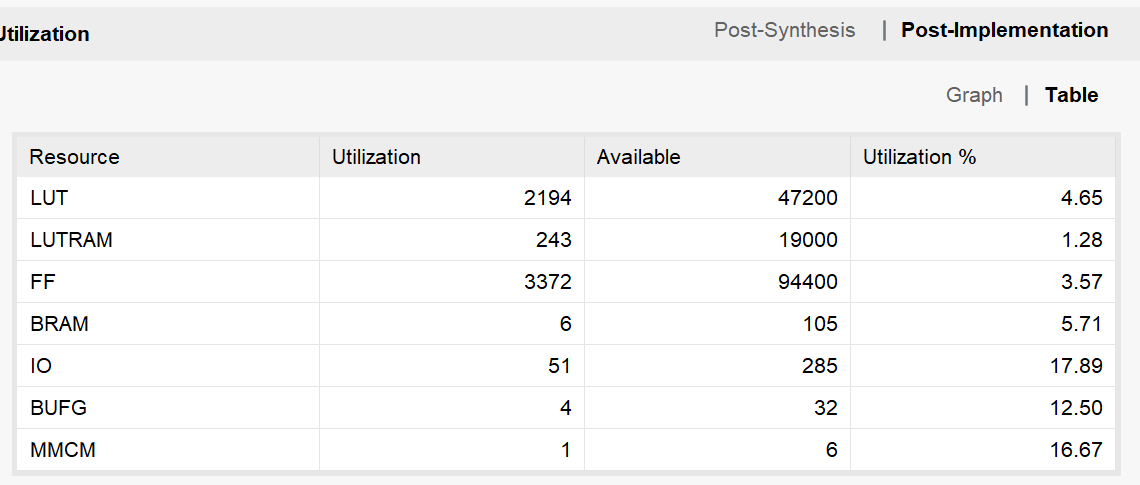
Question 1.

Each okWireOut captures at 200MHz(host interface clock) with 32 bits width thus it can transmit 800Mbytes/s

Question 2.

Each okWireIn updates at 200MHz(host interface clock) with 32 bits width thus it can transmit 800Mbytes/s

Question 3.



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)