**Final Project**

**Rachel Brooks**

**Compeng 331**

**8/14/2020**

**Design Code:**

`timescale 1ns / 1ps

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

// Company: CompEng 331 Lab3

// Engineer: Rachel Brooks

// Create Date: 07/01/2020 11:04:19 PM

// Module Name: ID and IF STAGES

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

//module 1 : pc - set current pc to next pc at edge of CPU

module pc(input [31:0] nextpc, output reg [31:0] outpc, input clk,input wpcir);

always @ (posedge clk)

begin

outpc <= nextpc;

end

endmodule

//module 2: adder - take in PC and output next PC (PC+4)

module adder(input [31:0] inpc,output reg [31:0] outpc);

initial

begin

outpc <=100; //starts at 100

end

always @(inpc)

begin

outpc <= inpc + 4; //adds 4 to the pc for the next one

end

endmodule

//module 3: IF/ID - takes in d0 and outputs it at edge of clk

module IF\_ID(input clk, input [31:0] d, output reg [31:0] out);

always @(posedge clk)

begin

out <= d;

end

endmodule

//module 4: Instruction Memory - takes in an address and outputs a 32 bit instruction d0

module instMem(input [31:0] add, output reg [31:0] instr);

reg [31:0] MEM[0:127];

initial

begin

MEM[100] = 32'b00000000001000100001100000100000;

MEM[104] = 32'b00000001001000110010000000100010;

MEM[108] = 32'b00000000011010010010100000100101;

MEM[112] = 32'b00000000011010010011000000100110;

MEM[116] = 32'b00000000011010010011100000100100;

end

always @(add) //at address put it in for instructions

begin

instr <= MEM[add] ;

end

endmodule

//module 5: RegFile - takes in source register and outputs data in register

module regFile(input clk, input [4:0] rs, input [4:0] rt, input we, input [4:0] wn,input [31:0] d, output reg [31:0] qa, output reg [31:0] qb);

reg [31:0] rf[31:0];

initial

begin

///rf[i] <= 32'd0;

rf[0] = 32'h00000000;

rf[1] = 32'hA00000AA;

rf[2] = 32'h10000011;

rf[3] = 32'h20000022;

rf[4] = 32'h30000033;

rf[5] = 32'h40000044;

rf[6] = 32'h50000055;

rf[7] = 32'h60000066;

rf[8] = 32'h70000077;

rf[9] = 32'h80000088;

rf[10] = 32'h90000099;

end

always @(\*)

begin

qa <= rf[rs];

qb <= rf[rt];

end

always @ (posedge clk)

begin

if(we)

begin

rf[wn] <= d;

end

end

endmodule

//module 6: Sign Extension - immediate (imm) is sign-extended into 32 bits

module signEx(input [15:0] in, output reg [31:0] signedout);

always @(in)

begin

signedout = { {16{in[15]}}, in[15:0]};

end

endmodule

//module 7: Control Unit - Use the op and func to help determine output (incorporates ALU)

module controlUnit(input [5:0] op, input [5:0] func, input [5:0] rs, input [5:0] rt, input [4:0] mrn, input [4:0] ern, input mm2reg, input mwreg, input em2reg, input ewreg,output reg wreg, output reg m2reg, output reg wmem, output reg [3:0] aluc, output reg aluimm, output reg regrt, output reg [1:0] fwda, output reg [1:0] fwdb);

//need add,sub, and, or ,xor, lw

always @(\*)

begin

if (op == 6'b000000)

begin

if (func == 6'b100000) //add

begin

regrt <= 0;

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluimm <= 0;

aluc <= 4'b0010;

end

else if (func == 6'b100010) begin //subtraction

regrt <= 0;

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluimm <=0;

aluc <=4'b0101;

end

else if (func == 6'b100100) begin //and

regrt <= 0;

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluimm <=0;

aluc <=4'b0000;

end

else if (func == 6'b100101)

begin

regrt <= 0;

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluimm <=0;

aluc <=4'b0001;

end

else if (func == 6'b100110)

begin

regrt <= 0;

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluimm <=0;

aluc <=4'b0011;

end

end

else if (op == 6'b100011)

begin //lw

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 1;

regrt <= 1;

end

if(rs==ern)

begin

fwda<=2'b01;

end

else if(rs==mrn & ~m2reg)

begin

fwda<=2'b10;

end

else if (m2reg & rs==mrn)

begin

fwda<=2'b11;

end

else

begin

fwda<=2'b00;

end

if(rt==ern)

begin

fwdb<=2'b01;

end

else if(rt==mrn & ~m2reg)

begin

fwdb<=2'b11;

end

else if(rt==mrn & m2reg)

begin

fwdb<=2'b11;

end

else

begin

fwdb<=2'b00;

end

end

endmodule

//module 8 : Multiplexer - 2:1 mux w selector

module mux(input [31:0] rd, input [31:0] rt, input regrt, output reg [31:0] out);

always @(\*)

begin

out <= (regrt) ? rt:rd;

end

endmodule

//module 9:ID/EXE

module ID\_EXE(input clk, input wreg, input m2reg,input wmem, input [3:0] aluc, input aluimm, input [4:0] mux, input [31:0] qa, input [31:0] qb,

input [31:0] exout, output reg ewreg, output reg em2reg,output reg ewmem, output reg [3:0] ealuc, output reg ealuimm, output reg [4:0] emux,

output reg [31:0] eqa, output reg [31:0] eqb, output reg [31:0] eexout);

always @(posedge clk)

begin

ewreg<=wreg;

em2reg<=m2reg;

ewmem<=wmem;

ealuc<=aluc;

ealuimm<=aluimm;

emux<=mux;

eqa<=qa;

eqb<=qb;

eexout<=exout;

end

endmodule

//module 10:ALU - performs additions

module ALU(input [31:0] a, input [31:0] b, input [3:0] ealuc, output reg [31:0] r);

always @ (\*)

begin

if(ealuc==4'b0010) //add

begin

r <= a+b;

end

else if(ealuc==4'b0101) //sub

begin

r <= a-b;

end

else if(ealuc==4'b0001) //or

begin

r <= a | b;

end

else if(ealuc==4'b0011) //xor

begin

r <= a^b;

end

else if(ealuc==4'b0000) //and

begin

r <= a&b;

end

end

endmodule

//module 11:Multiplexer in EXE - slects immediate

module exe\_mux(input [31:0] eqb, input [31:0] exout , input ealuimm, output reg [31:0] b);

always @(\*)

begin

b <= (ealuimm) ? exout:eqb;

end

endmodule

//module 12: EXE/MEM

module EXE\_MEM(input clk, input ewreg, input em2reg, input ewmem, input [4:0] emux, input [31:0] r, input [31:0] qb,

output reg mwreg, output reg mm2reg, output reg mwmem, output reg [4:0] mmux, output reg [31:0] mr, output reg [31:0] di);

always @(posedge clk)

begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

mmux <= emux;

mr <=r;

di <= qb;

end

endmodule

//Module 13: Datamemory - reads data memory

module Datamem(input [31:0] a, input [31:0] di, input we, output reg [31:0] do);

reg [31:0] MEM[0:127];

initial

begin

MEM[0] = 32'hA00000AA;

MEM[4] = 32'h10000011;

MEM[8] = 32'h20000022;

MEM[12] = 32'h30000033;

MEM[16] = 32'h40000044;

MEM[20] = 32'h50000055;

MEM[24] = 32'h60000066;

MEM[28] = 32'h70000077;

MEM[32] = 32'h80000088;

MEM[36] = 32'h90000099;

end

always @ (\*)

begin

do <= MEM[a];

end

endmodule

//Module 14:MEM/WB

module MEM\_WB(input clk, input mwreg, input mm2reg, input [4:0] mmux, input [31:0] mr, input [31:0] do,

output reg wwreg, output reg wm2reg, output reg [4:0] wmux,output reg [31:0] wr, output reg [31:0] wdo);

always @(posedge clk)

begin

wwreg <= mwreg;

wm2reg <= mm2reg;

wmux <= mmux;

wr <= mr;

wdo <= do;

end

endmodule

//Module 15: WB stage - the memory data is selected and will be written into the register file at the end of the cycle

module wb\_mux(input [31:0] wr,input [31:0] wdo,input wm2reg, output reg [31:0] wb\_d);

always @(\*)

begin

wb\_d <= (wm2reg) ? wdo:wr;

end

endmodule

module fwda\_sel(input [1:0] fwda, input [31:0] qa, input [31:0] aluout, input [31:0] ealuout, input [31:0] do, output reg [31:0] out);

always @(\*) begin

if (fwda ==2'b00)

begin

out <= qa;

end

else if (fwda == 2'b01)

begin

out <= aluout;

end

else if (fwda == 2'b10)

begin

out <= ealuout;

end

else

begin

out <= do;

end

end

endmodule

module fwdb\_sel(input [1:0] fwdb, input [31:0] qb, input [31:0] aluout, input [31:0] ealuout, input [31:0] do, output reg [31:0] out);

always @(\*)

begin

if (fwdb ==2'b00)

begin

out <= qb;

end

else if (fwdb == 2'b01)

begin

out <= aluout;

end

else if (fwdb == 2'b10)

begin

out <= ealuout;

end

else

begin

out <= do;

end

end

endmodule

**Test Bench:**

**`timescale 1ns / 1ps**

**//////////////////////////////////////////////////////////////////////////////////**

**// Company: Compeng 331 Lab3**

**// Engineer: Rachel Brooks**

**// Create Date: 07/01/2020 11:03:44 PM**

**//Module Name: testbench**

**//////////////////////////////////////////////////////////////////////////////////**

**module testbench();**

**reg clk;**

**//pc and adder**

**wire [31:0] pc;**

**wire [31:0] pcout;**

**//instruction mem**

**wire [31:0] instr;**

**wire [31:0] outif;**

**//CU outputs**

**wire wreg;**

**wire m2reg;**

**wire wmem;**

**wire [3:0] aluc;**

**wire aluimm;**

**//mux input**

**wire regrt;**

**//muxoutput**

**wire [4:0] muxo;**

**//regfile output**

**wire [31:0] qa;**

**wire [31:0] qb;**

**//signextension output**

**wire [31:0] exout;**

**// id exe output**

**wire em2reg;**

**wire ewreg;**

**wire ewmem;**

**wire [3:0] ealuc;**

**wire ealuimm;**

**//from muxout**

**wire [4:0] emuxo;**

**//from regfile**

**wire [31:0] eqa;**

**wire [31:0] eqb;**

**//sign extension**

**wire [31:0] eexout;**

**//mux output**

**wire [31:0] exemuxo;**

**//alu output**

**wire [31:0] aluout;**

**//exemem output**

**wire mwreg;**

**wire mm2reg;**

**wire mwmem;**

**wire [4:0] mmux;**

**wire [31:0] maluout;**

**wire [31:0] mqb;**

**//datamem output**

**wire [31:0] do;**

**//memwb output**

**wire wwreg;**

**wire wm2reg;**

**wire [4:0] wmux;**

**wire [31:0] waluout;**

**wire [31:0] wdo;**

**wire [31:0] wb\_d;**

**wire [1:0] fwda;**

**wire [31:0]fwda\_out;**

**wire [1:0] fwdb;**

**wire [31:0]fwdb\_out;**

**pc pc\_tb(pc, pcout, clk,); //pc to next pc**

**adder adder\_tb(pcout, pc); //uses pc output and gets next pc**

**IF\_ID IF\_ID\_tb(clk, instr,outif); //gets instruction and outputs it at clk edge (need clk)**

**instMem instMem\_tb(pcout, instr); //in address from adder and out instruction**

**regFile regFile\_tb(clk, outif[25:21], outif[20:16], wwreg, wmux, wb\_d, qa, qb);**

**signEx signEx\_tb(outif, exout);**

**controlUnit controlUnit\_tb(outif[31:26], outif[5:0], outif[25:21], outif[20:16],mmux,emuxo, mm2reg,mwreg, em2reg, ewreg, wreg, m2reg, wmem, aluc, aluimm, regrt,fwda,fwdb);**

**mux mux\_tb(outif[15:11], outif[20:16], regrt, muxo);**

**ID\_EXE ID\_EXE\_tb(clk, wreg, m2reg,wmem, aluc, aluimm, muxo, fwda\_out, fwdb\_out, exout, ewreg, em2reg,ewmem, ealuc, ealuimm, emuxo, eqa, eqb, eexout);**

**exe\_mux exe\_mux\_tb(eqb,eexout,ealuimm,exemuxo);**

**ALU ALU\_tb(eqa,exemuxo,ealuc,aluout);**

**EXE\_MEM EXE\_MEM\_TB(clk, ewreg, em2reg, ewmem, emuxo, aluout, eqb, mwreg, mm2reg, mwmem, mmux, maluout, mqb);**

**Datamem Datamem\_tb(maluout, mqb, mwmem ,do);**

**MEM\_WB MEM\_WB\_tb(clk, mwreg,mm2reg,mmux, maluout,do,wwreg,wm2reg,wmux,waluout,wdo);**

**wb\_mux wb\_mux\_tb(waluout,wdo,wm2reg,wb\_d);**

**fwda\_sel fwda\_sel\_tb(fwda, qa, aluout, maluout, do,fwda\_out);**

**fwdb\_sel fwdb\_sel\_tb(fwdb, qb, aluout, maluout, do,fwdb\_out);**

**initial begin**

**clk = 0;**

**end**

**always begin**

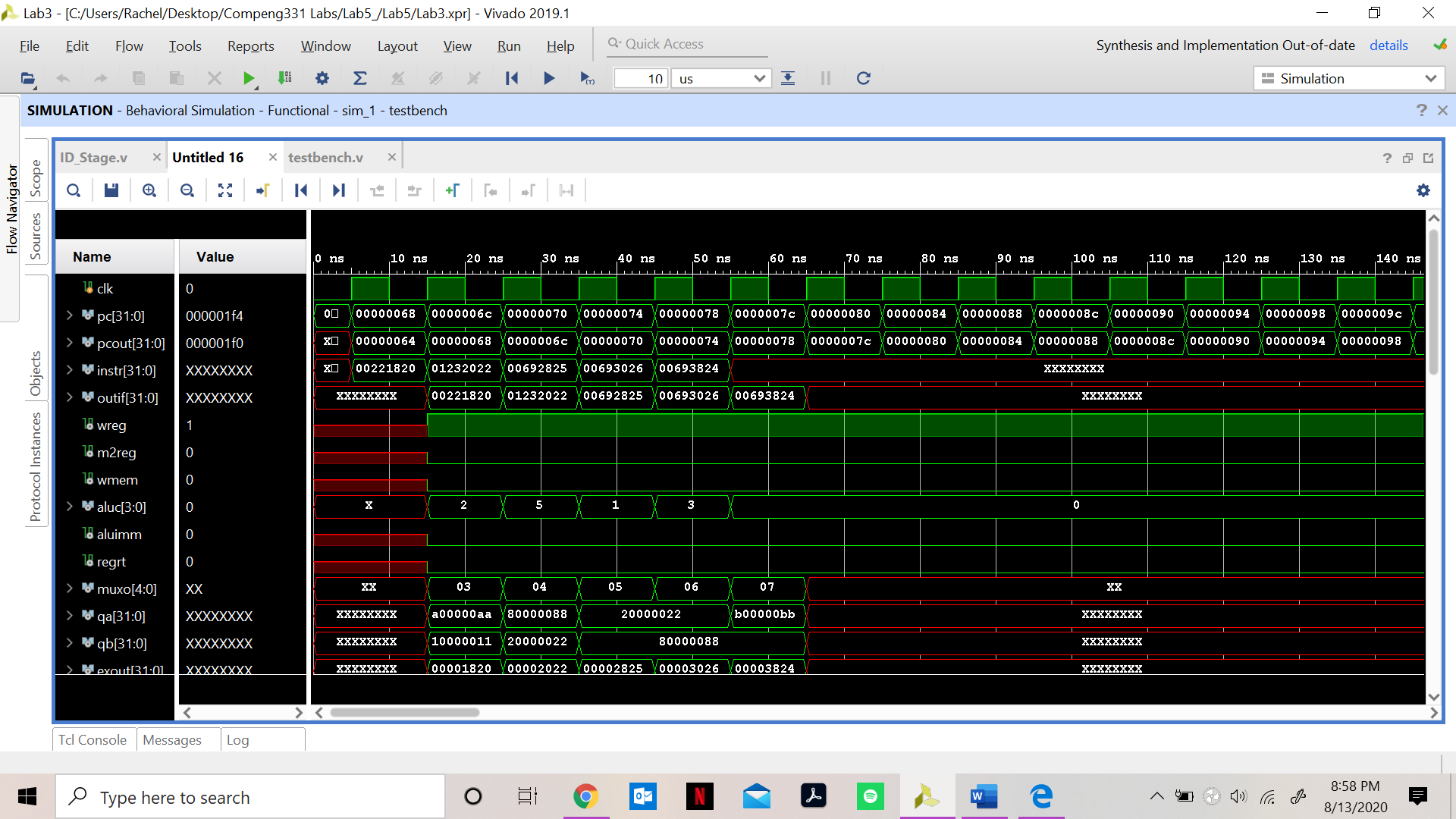
**#5;**

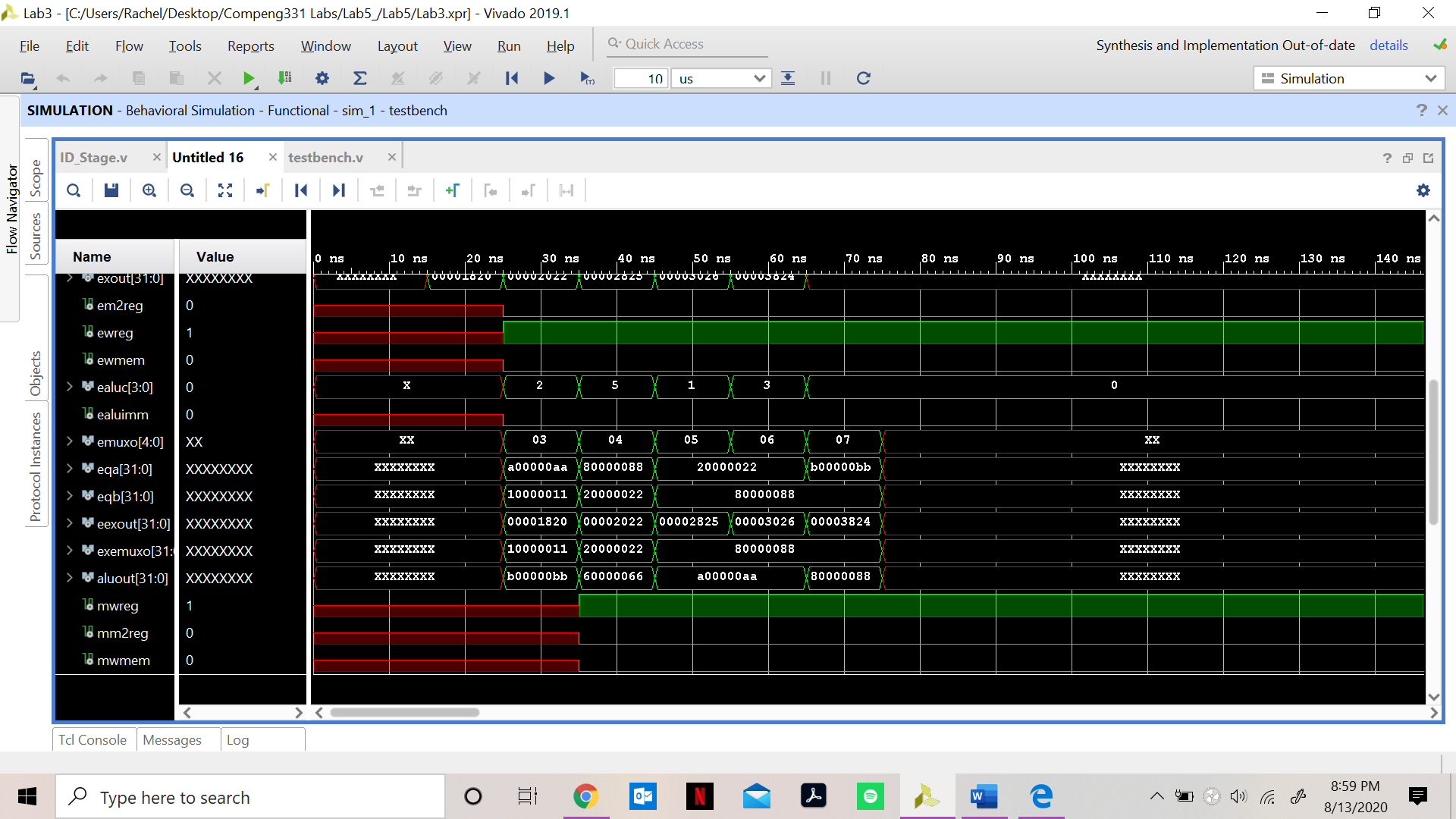
**clk = ~clk;**

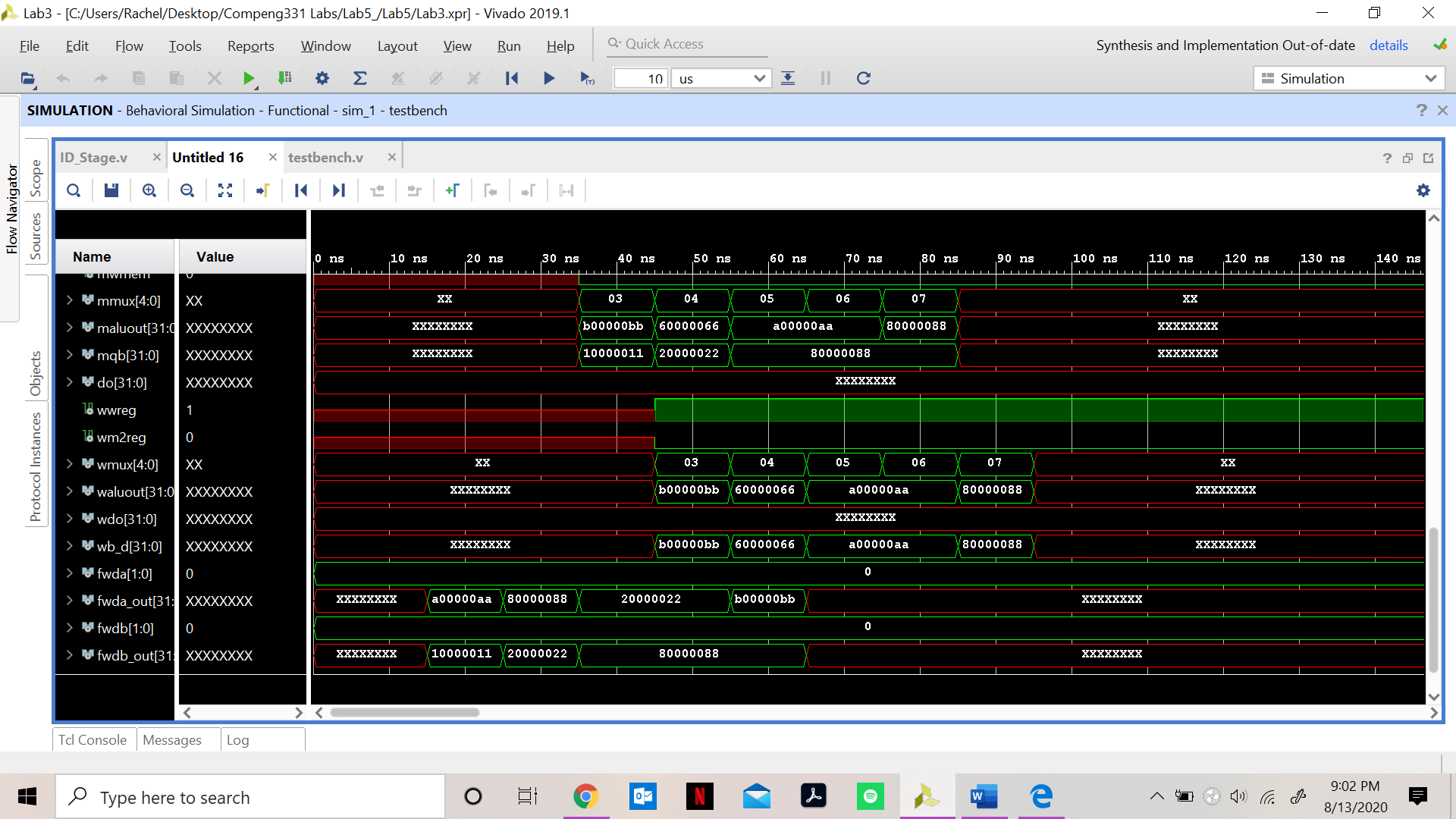
**end**

**endmodule**

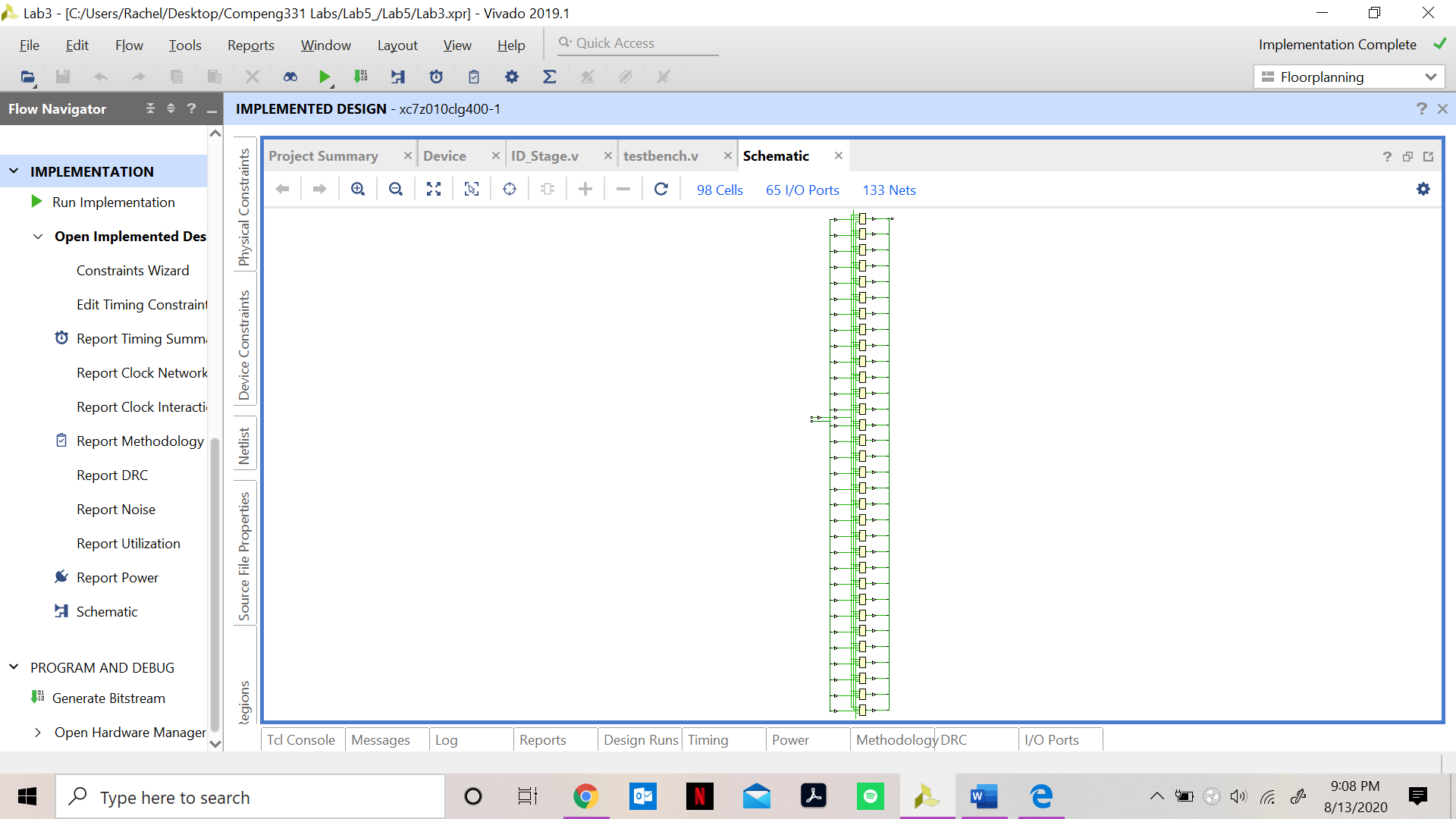
**Wave Forms:**

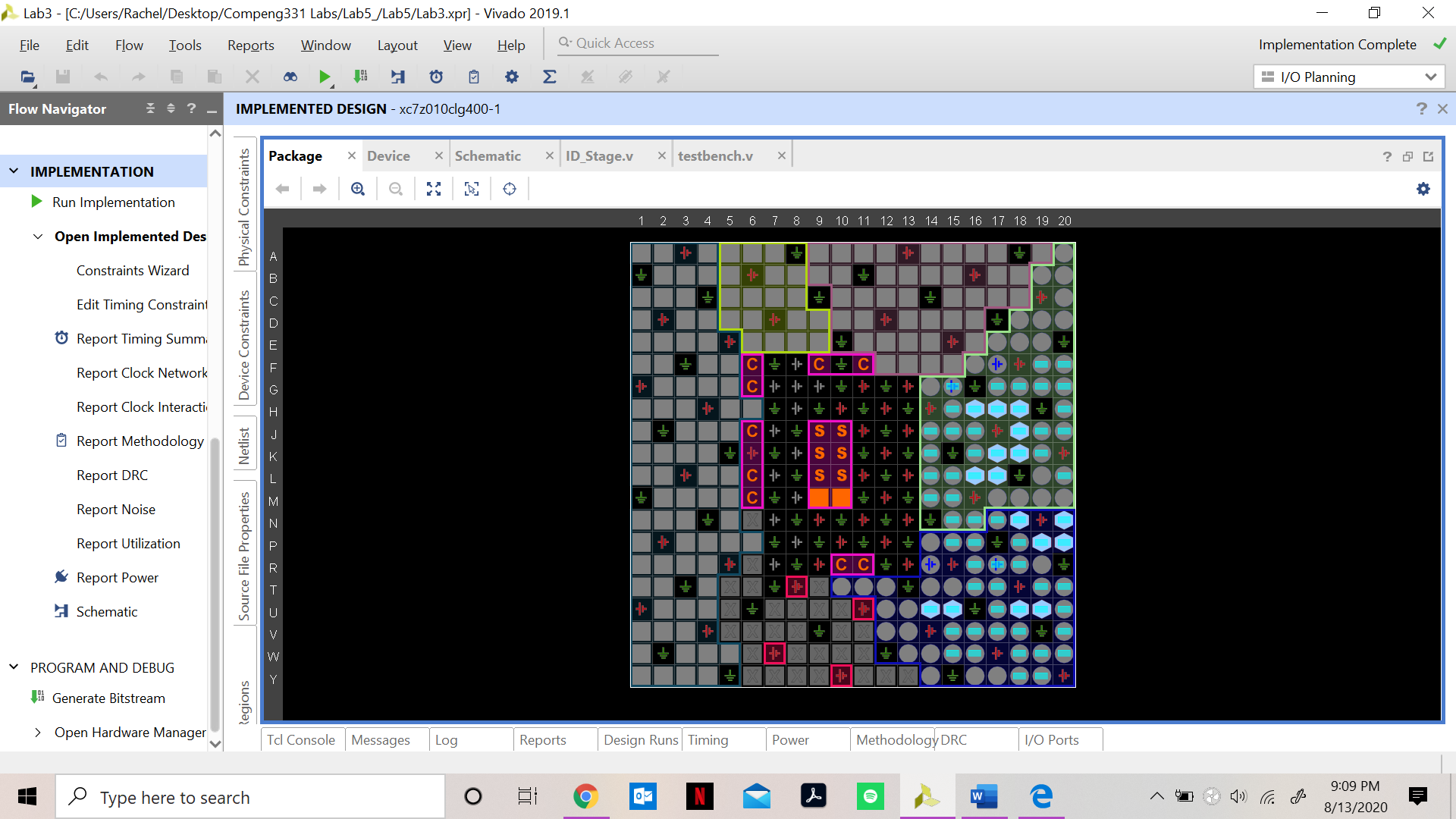


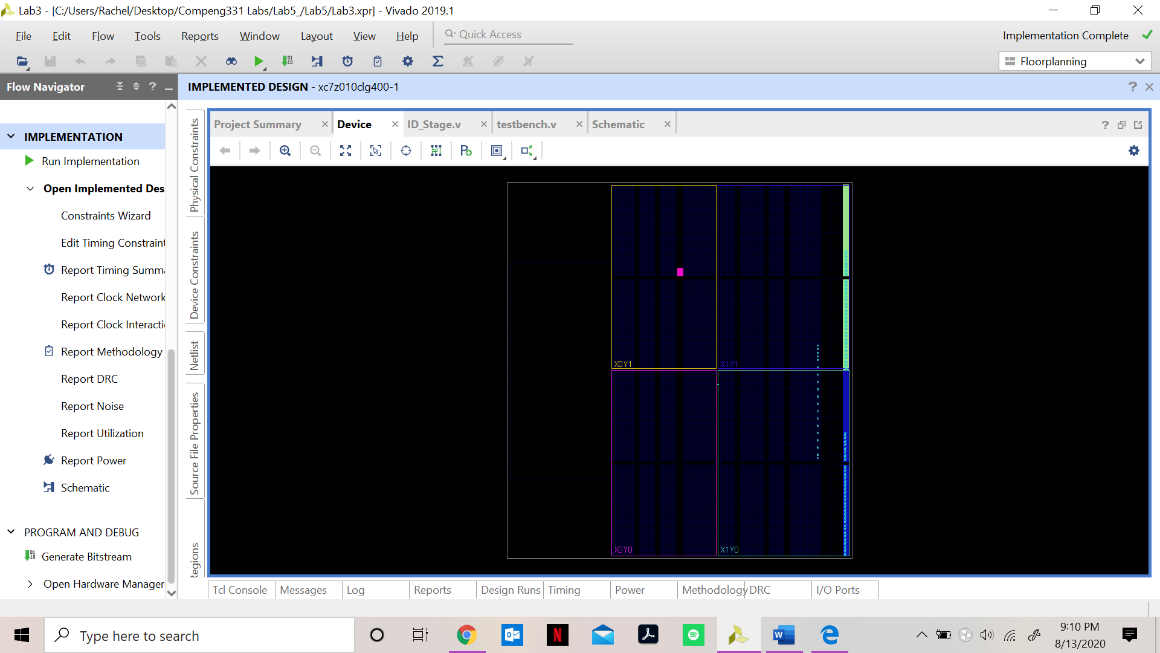




**Design Schematic:**



**I/O Planning:**

**Floor Planning:**