Lab 03

CMPEN 331

Ryan Herman

Rqh5359

Design Code:

1. **PC Module:**

module PC(PCIn, PCOut, clk);

//inputs - clock, NPCin

input clk;

input [31:0] PCIn;

//outputs - NPOut

output reg [31:0] PCOut;

initial

begin

//Set first instance to 100

PCOut <= 100;

end

//On positive edge set new program count

always @ (posedge clk)

begin

PCOut <= PCIn;

end

endmodule

1. **Adder Module**

module AdderModule(PCIn, PCOut);

//inputs - Current Program Counter

input [31:0] PCIn;

//Outputs - Next Program Counter (CurrentPC + 4)

output reg [31:0] PCOut;

//Add 4 to get next PC

always @(\*)

begin

PCOut = PCIn + 4;

end

endmodule

1. **Instruction Memory Module**

module InstructionMemory(a, do);

//inputs - address (a)

input [31:0] a;

//outputs - instruction (do) //from hints video

output reg [31:0] do;

//Creat Intstruction Memory

reg [31:0] IM [0:511];

initial

begin

//lw binary code

IM[100] <= 32'b10001100000000100000000000000000;

IM[104] <= 32'b10001100000000110000000000000100;

end

always @(a)

begin

//instruction (do) = the instruction at IM[address]

do <= IM[a];

end

endmodule

1. **IF/ID Module**

module IF\_ID(do, RegOut, clk);

//inputs - Instruction (do), clock

input [31:0] do;

input clk;

//output - instruction register

output reg [31:0] RegOut;

initial

begin

//Set to 0

RegOut <= 0;

end

always @(posedge clk)

//Change RegOut Value on positive edge

begin

RegOut <= do;

end

endmodule

1. **Control Module**

module ControlUnit(Opcode, func, wreg, m2reg, wmem, aluc, aluimm);

//inputs - operation code

input[5:0] Opcode;

input[5:0] func; //Only needed for R-type

//outputs - RegWrite, MemtoReg, MemWrite, AlucCode (lw - 0010), aluimm

output reg [1:0] wreg;

output reg [1:0] m2reg;

output reg [1:0] wmem;

output reg [3:0] aluc;

output reg [1:0] aluimm;

always @ (\*)

begin

case (Opcode)

//lw case

6'b100011: begin

wreg = 1;

m2reg = 1;

wmem = 0;

aluc = 4'b0010; //ALU OPCode 2b'00 = 0010 (from table)

aluimm = 1;

end

endcase

//other cases would go below

end

endmodule

1. **Mux Module**

module Mux(rd, rt, selector, out);

input [4:0] rd, rt;

input [1:0] selector;

output reg [4:0] out;

//mux on selector

always @ (selector or rd or rt)

begin

out = (selector) ? rd : rt;

end

endmodule

1. **Register File Module**

module RegFile(rs, qa, clk);

input [4:0] rs;

input clk;

output reg [31:0] qa;

//create a register file

reg[31:0] RegFile [31:0];

//Initialize all to zero

initial

begin

RegFile[0] = 32'd0; RegFile[1] = 32'd0; RegFile[2] = 32'd0; RegFile[3] = 32'd0;

RegFile[4] = 32'd0; RegFile[5] = 32'd0; RegFile[6] = 32'd0; RegFile[7] = 32'd0;

RegFile[8] = 32'd0; RegFile[9] = 32'd0; RegFile[10] = 32'd0; RegFile[11] = 32'd0;

RegFile[12] = 32'd0; RegFile[13] = 32'd0; RegFile[14] = 32'd0; RegFile[15] = 32'd0;

RegFile[16] = 32'd0; RegFile[17] = 32'd0; RegFile[18] = 32'd0; RegFile[19] = 32'd0;

RegFile[20] = 32'd0; RegFile[21] = 32'd0; RegFile[22] = 32'd0; RegFile[23] = 32'd0;

RegFile[24] = 32'd0; RegFile[25] = 32'd0; RegFile[26] = 32'd0; RegFile[27] = 32'd0;

RegFile[28] = 32'd0; RegFile[29] = 32'd0; RegFile[30] = 32'd0; RegFile[31] = 32'd0;

end

//just output for lw instruction

always @ (posedge clk)

begin

qa = RegFile[rs];

end

endmodule

1. **Sign Extender Module**

module SignExtender(in, out);

input [15:0] in;

output reg [31:0] out;

//extend 16 bits --> 32 bits

always @ (\*)

begin

out <= {{16{in[15]}}, in};

end

endmodule

1. **ID/EXE Module**

module ID\_EXE(wreg, m2reg, wmem, aluc, aluimm, mux, qa, qb, extend, clk,

ewreg, em2reg, ewmem, ealuc, ealuimm, emux, eqa, eqb, eextend);

//input

input [1:0] wreg, m2reg, wmem, aluimm;

input [3:0] aluc;

input [4:0] mux;

input [31:0] qa, qb, extend;

input clk;

output reg [1:0] ewreg, em2reg, ewmem, ealuimm;

output reg [3:0] ealuc;

output reg [4:0] emux;

output reg [31:0] eqa, eqb, eextend;

//set bits for future use

always @ (posedge clk)

begin

ewreg = wreg;

em2reg = m2reg;

ewmem = wmem;

ealuimm = aluimm;

ealuc = aluc;

emux = mux;

eqa = qa;

eqb = qb;

eextend = extend;

end

endmodule

**Testbench Code:** // I had a lot of trouble debugging for a long time

`timescale 1ns / 1ps

module testbench();

reg clk\_tb;

reg [31:0] PCIn;

wire [31:0] PCOut;

wire [31:0] a;

wire [31:0] do;

wire [31:0] RegOut;

reg [5:0] Opcode, func;

wire [1:0] wreg, m2reg, wmem, aluimm;

wire [3:0] aluc;

reg [4:0] rd, rt;

reg [1:0] selector;

wire [4:0] out;

reg [4:0] rs;

wire [31:0] qa;

reg [15:0] in;

wire [31:0] out2;

wire [1:0] ewreg, em2reg, ewmem, ealuimm;

wire [3:0] ealuc;

wire [4:0] emux;

wire [31:0] eqa, eqb, eextend;

PC PC(PCIn, PCOut, clk\_tb);

AdderModule AdderModule(PCIn, PCOut);

InstructionMemory InstructionMemory(a, do);

IF\_ID IF\_ID(do, RegOut, clk);

ControlUnit ControlUnit(Opcode, func, wreg, m2reg, wmem, aluc, aluimm);

Mux Mux(rd, rt, selector, out);

RegFile RegFile(rs, qa, clk);

SignExtender SignExtender(in, out2);

ID\_EXE ID\_EXE(wreg, m2reg, wmem, aluc, aluimm, mux, qa, qb, extend, clk,

ewreg, em2reg, ewmem, ealuc, ealuimm, emux, eqa, eqb, eextend);

initial

begin

clk\_tb = 0;

end

always @ (\*)

begin

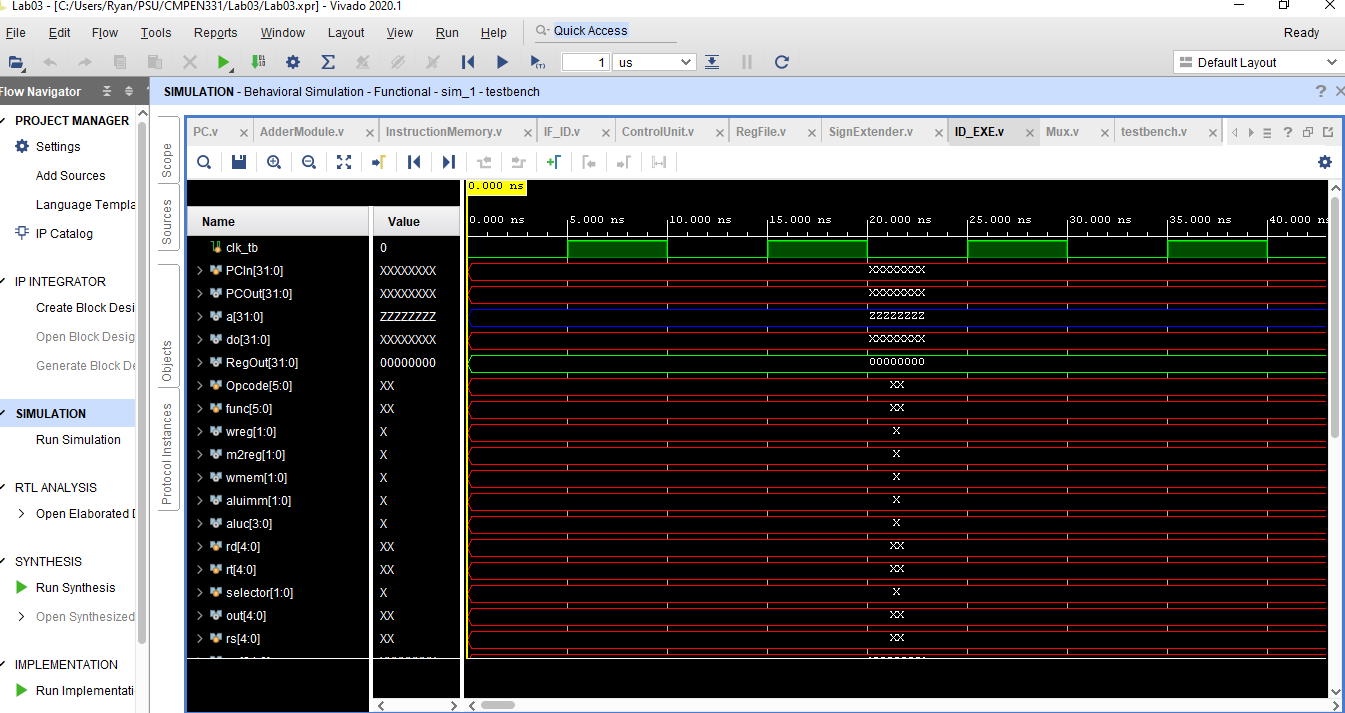
#5;

clk\_tb = ~clk\_tb;

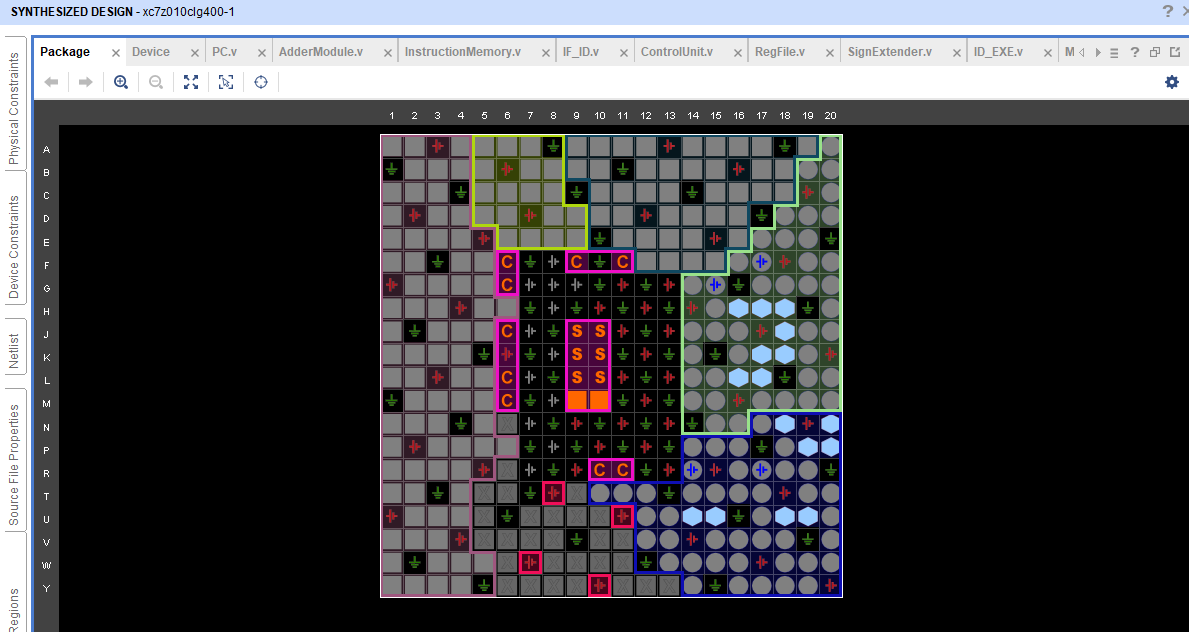
end

endmodule

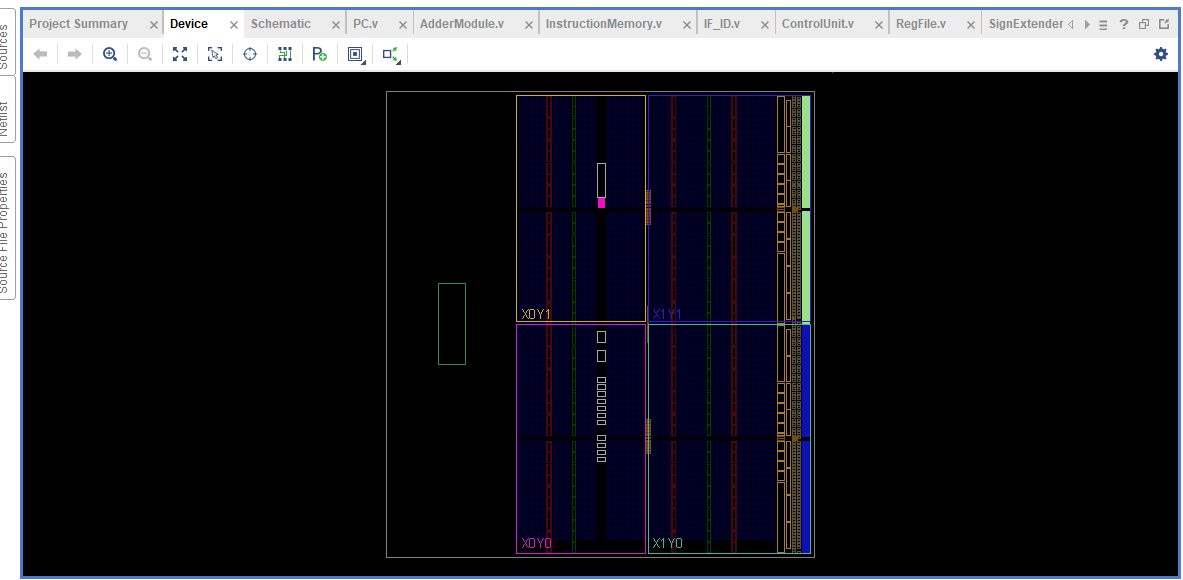
Waveform:



I/O Planning:



Floorplanning:



Schematic:

