Armin Zou & Wenxuan Xu

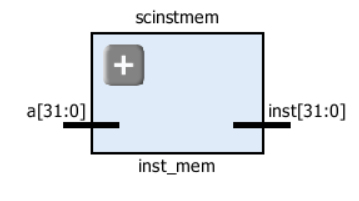
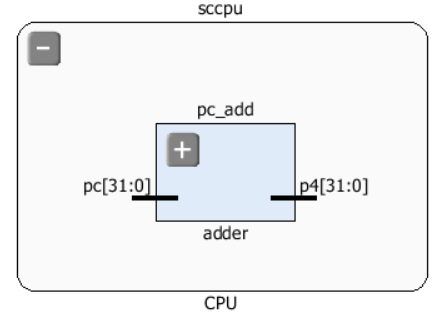
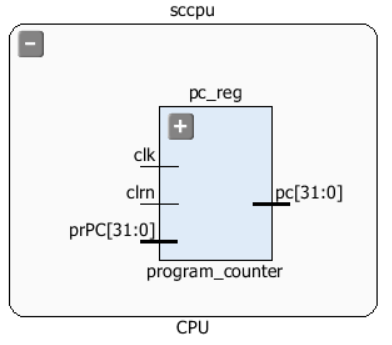
**Final Project Report**

**Abstract:**

In this project, we are implementing a Single-Cycle CPU using the Xilinx design package for FPGAs for the R-type, I-type and J-type instruction. The single-cycle computer is designed with a single-cycle CPU and two memory modules, each instruction is executed in 1 clock cycle and each component can be used only once. Furthermore, we are testing our design by generated the bit stream file and programmed it into the ZyboBoard. In order to test our design, chosen sw0 of the logic slide switches as an input to control the beginning of fetching of the instructions, and chosen LED 0 of the logic LEDs to light on. To verify the result, the right-most LED 0 would light on if the right-most switch sw0 is turned on.

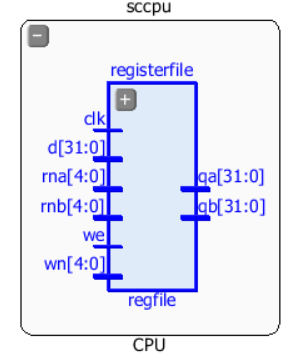
**Introduction:**

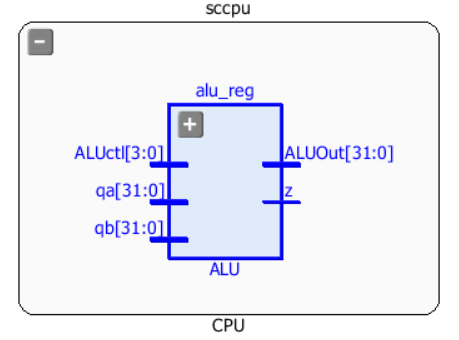
The Single-cycle implementation design is well-structured for R-type, I-type and J-type instruction. The single-cycle CPU is a collection of state and combinational logic which consist of different components (Control unit, program counter, etc.).

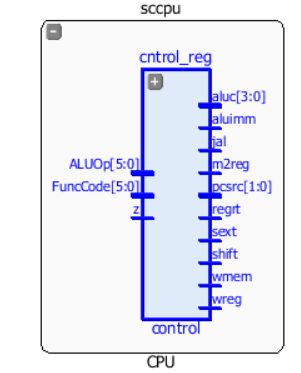
* The common elements for all instructions are instruction memory, PC counter, Adder, they are performing the instruction fetch that used by all instructions. 

The PC gives address to instruction memory, the Memory outputs instruction contents, and the adder increment PC to the next instruction address.

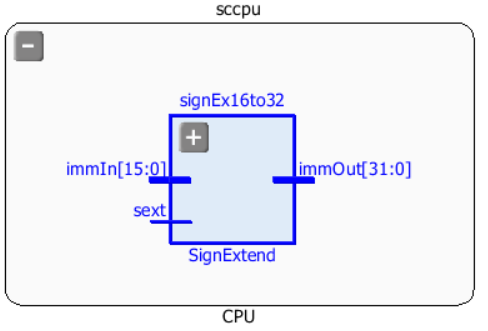
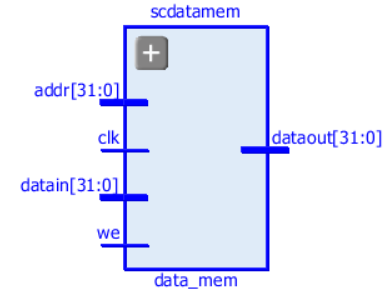
* R-type instruction access the following components:
* Registerfile: 2 read registers rt and rd, 1 write register rd, and write data register for the inputs, and 2 read data values for the outpus.



* ALU: Performs arithmetic/logical operations for the instructions. 
* Control unit: ALU control selects operation (aluc).

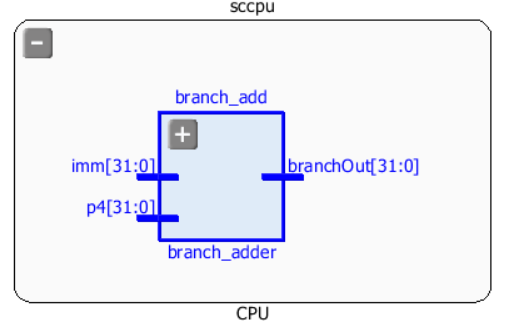
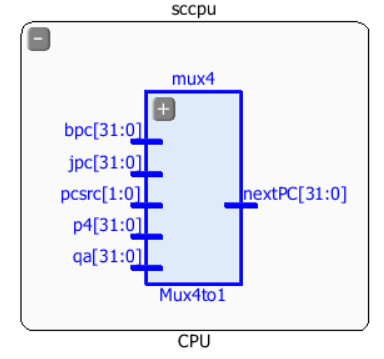


* I-type instruction:
* Load and Store: Using read data 1 from Registerfile as base register value to ALU, and read data 2 from Registerfile as the data value to be stored in data memory, MemRead, MemWrite controls determine whether to read or write data memory. Also the components Signextend and Data Memory for load and store instruction.

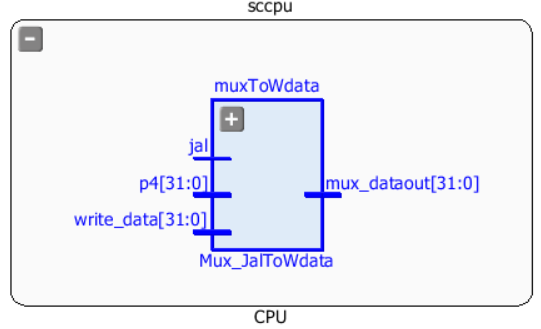
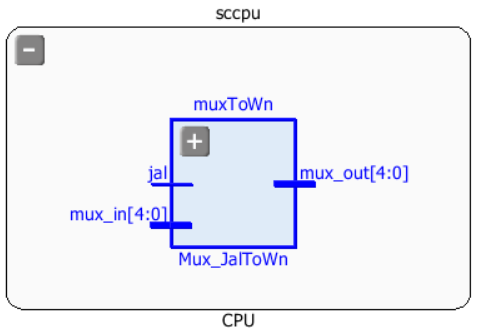
* Branch: Compare contents of 2 registers.

1. Shift 16-bit offset left by 2 bits to get word address.
2. Using shifted offset value to value of PC + 4 to get branch target address.
3. Update PC with branch target if operands are equal (branch is taken).
4. Also modify instruction fetch datapath to allow PC to be updated with new value.

* J-type instruction:
* Jump requires different address calculation.

Replace lower 28 bits of PC with 26 bits from instruction, shifted left 2 bits and to be added using input (jpc) from multiplexer 4to1. Also the control signal jal determined the jump and link instruction with two multiplexers to pass the values for the instructions into registerfile.

Overall, the advantage of single-cycle implementation is simple to design and easy to understand.

Write a report that contains the following:

a.   Your Verilog® design code. Use: i.   Device Family: Zynq-7000 ii.   Device: XC7Z010- -1CLG400C or ZyboBoard

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/27/2016 06:37:05 PM

// Design Name:

// Module Name: sccomp

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module sccomp(clk,clrn,finish);

input clk,clrn;

// output [31:0] pc; // uncomment to display output in simulation waveform

// output [31:0] inst; // uncomment to display output in simulation waveform

// output [31:0] alu; // uncomment to display output in simulation waveform

// output [31:0] memout; // uncomment to display output in simulation waveform

output reg finish;

wire [31:0] inst;

wire [31:0] pc;

wire [31:0] alu;

wire [31:0] memout;

wire [31:0] data;

wire wmem;

CPU sccpu(clk,clrn,inst,memout,pc,wmem,alu,data);

inst\_mem scinstmem (pc,inst);

data\_mem scdatamem (alu,data,memout,wmem,clk);

always@(clrn,pc,finish)

if(clrn == 0)

finish = 0;

else if(pc == 32'h0000005c)

finish = 1;

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 11:00:56 PM

// Design Name:

// Module Name: CPU

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module CPU(clk,clrn,inst,mem,pc,wmem,alu\_out,data);

input clk,clrn; // clock and reset

input [31:0] inst; // inst from inst memory

input [31:0] mem; // data from data memory

output [31:0] pc; // program counter;

output [31:0] alu\_out; // alu output

output [31:0] data; // data to data memory

output wmem; // write data memory

// instruction fields

wire [5:0] op = inst[31:26];

wire [4:0] rs = inst[25:21];

wire [4:0] rt = inst[20:16];

wire [4:0] rd = inst[15:11];

wire [5:0] func = inst[05:00];

wire [15:0] imm = inst[15:00];

wire [25:0] addr = inst[25:00];

// control signals

wire [3:0] aluc;

wire [1:0] pcsrc;

wire wreg,regrt,m2reg,shift,aluimm,sext,z,jal;

// datapath wires

wire [31:0] sa = {27'b0,inst[10:06]};

wire [31:0] p4;

wire [31:0] nxpc,pc;

wire [31:0] write\_data;

wire [31:0] qa,qb;

wire [31:0] alua;

wire [31:0] imm\_32;

wire [31:0] alub;

wire [31:0] alu\_out;

wire [31:0] imm\_shift = {imm\_32[29:0],2'b00};

wire [31:0] branchOut;

wire [31:0] jpc = {p4[31:28],addr,2'b00};

wire [4:0] wn;

wire [4:0] jal\_datain;

wire [31:0] mux\_Writedata;

program\_counter pc\_reg(nxpc,clk,clrn,pc); // Program Counter Register

adder pc\_add(pc,p4);

control cntrol\_reg(op,func,aluc,regrt,pcsrc,wreg,aluimm,wmem,m2reg,shift,z,sext,jal); // control unit

Mux\_regfile muxToRegfile(rd,rt,regrt,jal\_datain); // MultiPlexer for Regfile wn

Mux\_JalToWn muxToWn(jal\_datain,jal,wn); // Muxtilplexer for jal to wn

regfile registerfile(rs,rt,wn,write\_data,wreg,qa,qb,clk); // Register File

Mux\_Alua muxToAlua(qa,sa,shift,alua); // Multiplexer for Alua

SignExtend signEx16to32(imm,sext,imm\_32); // SignExtension 16to32

Mux\_Alub muxToAlub(qb,imm\_32,aluimm,alub); // Multiplexer for Alub

ALU alu\_reg(aluc,alua,alub,alu\_out,z); // ALU

Mux\_WData muxToData(alu\_out,mem,m2reg,mux\_Writedata);

Mux\_JalToWdata muxToWdata(mux\_Writedata,p4,jal,write\_data);

branch\_adder branch\_add(p4,imm\_shift,branchOut); // Branch Addder

Mux4to1 mux4(p4,branchOut,qa,jpc,pcsrc,nxpc); // 4to1 Multiplexer

assign data = qb;

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/13/2016 06:16:42 PM

// Design Name:

// Module Name: program\_counter

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module program\_counter(prPC,clk,clrn,pc);

input [31:0] prPC;

input clk;

input clrn;

output reg [31:0] pc;

always@(posedge clk or posedge clrn)

begin

if(clrn == 1)

pc <= 0;

else

pc <= prPC;

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/13/2016 08:42:02 PM

// Design Name:

// Module Name: adder

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module adder(pc,p4);

input [31:0]pc;

output reg [31:0] p4;

always@(\*)begin

p4 <= pc + 4;

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/13/2016 06:13:46 PM

// Design Name:

// Module Name: control

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module control(ALUOp,FuncCode,aluc,regrt,pcsrc,wreg,aluimm,wmem,m2reg,shift,z,sext,jal);

input [5:0] ALUOp;

input [5:0] FuncCode;

input z;

output reg regrt, wreg,aluimm,wmem,m2reg,shift,sext,jal;

output reg [3:0] aluc;

output reg [1:0] pcsrc;

always@(\*)

begin

case(ALUOp)

6'b000000:begin

case(FuncCode)

6'b100000:begin // add

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0000;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b100010:begin //subtract

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0100;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b100100:begin // and

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0001;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b100101:begin // or

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0101;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b100110:begin // xor

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0010; // xor

wmem <= 0;

pcsrc <= 2'b00;

end

6'b000000:begin // sll

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 1;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0011;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b000010:begin // srl

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 1;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0111;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b000011:begin // sra

wreg <= 1;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 1;

aluimm <= 0;

sext <= 0;

aluc <= 4'b1111;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b001000:begin // jr

wreg <= 0;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0000;

wmem <= 0;

pcsrc <= 2'b10;

end

endcase

end

6'b001000:begin // addi

wreg <= 1;

regrt <= 1;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 1;

sext <= 1;

aluc <= 4'b0000;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b001100:begin //andi

wreg <= 1;

regrt <= 1;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 1;

sext <= 0;

aluc <= 4'b0001;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b001101:begin // ori

wreg <= 1;

regrt <= 1;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 1;

sext <= 0;

aluc <= 4'b0101;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b001110:begin // xori

wreg <= 1;

regrt <= 1;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 1;

sext <= 0;

aluc <= 4'b0010;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b100011:begin // lw

wreg <= 1;

regrt <= 1;

jal <= 0;

m2reg <= 1;

shift <= 0;

aluimm <= 1;

sext <= 1;

aluc <= 4'b0000;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b101011:begin // sw

wreg <= 0;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 1;

sext <= 1;

aluc <= 4'b0000;

wmem <= 1;

pcsrc <= 2'b00;

end

6'b000100:begin // beq

wreg <= 0;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 1;

aluc <= 4'b0010;

wmem <= 0;

case(z)

0:pcsrc <= 2'b00;

1:pcsrc <= 2'b01;

endcase

end

6'b000101:begin // bne

wreg <= 0;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 1;

aluc <= 4'b0010;

wmem <= 0;

case(z)

0:pcsrc <= 2'b01;

1:pcsrc <= 2'b00;

endcase

end

6'b001111:begin // lui

wreg <= 1;

regrt <= 1;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 1;

sext <= 0;

aluc <= 4'b0110;

wmem <= 0;

pcsrc <= 2'b00;

end

6'b000010:begin // jump

wreg <= 0;

regrt <= 0;

jal <= 0;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0000;

wmem <= 0;

pcsrc <= 2'b11;

end

6'b000011:begin // jal

wreg <= 1;

regrt <= 0;

jal <= 1;

m2reg <= 0;

shift <= 0;

aluimm <= 0;

sext <= 0;

aluc <= 4'b0000;

wmem <= 0;

pcsrc <= 2'b11;

end

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 09:24:02 PM

// Design Name:

// Module Name: Mux\_regfile

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux\_regfile(rd,rt,sel,mux\_out);

input [4:0] rd;

input [4:0] rt;

input sel;

output reg [4:0] mux\_out;

always@(\*)begin

case(sel)

0:mux\_out <= rd;

1:mux\_out <= rt;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/14/2016 01:53:38 PM

// Design Name:

// Module Name: Mux\_JalToWn

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux\_JalToWn(mux\_in,jal,mux\_out);

input [4:0] mux\_in;

input jal;

output reg [4:0] mux\_out;

always@(\*)begin

case(jal)

0:mux\_out <= mux\_in;

1:mux\_out <= 5'b11111;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/13/2016 06:14:56 PM

// Design Name:

// Module Name: regfile

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module regfile(rna,rnb,wn,d,we,qa,qb,clk);

input [31:0] d;

input [4:0] rna;

input [4:0] rnb;

input [4:0] wn;

input we;

input clk;

output [31:0] qa, qb;

reg [31:0] RAM [0:31];

initial begin

RAM[0] = 32'h00000000;

RAM[1] = 32'hA00000AA;

RAM[2] = 32'h10000011;

RAM[3] = 32'h20000022;

RAM[4] = 32'h30000033;

RAM[5] = 32'h40000044;

RAM[6] = 32'h50000055;

RAM[7] = 32'h60000066;

RAM[8] = 32'h70000077;

RAM[9] = 32'h80000088;

RAM[10] = 32'h90000099;

end

assign qa = (rna == 0)? 0 : RAM[rna];

assign qb = (rna == 0)? 0 : RAM[rnb];

always@(posedge clk)

begin

if ((wn!= 0)&& we)

RAM[wn] <= d;

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 09:29:05 PM

// Design Name:

// Module Name: Mux\_Alua

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux\_Alua(qa,sa,sel,mux\_out);

input [31:0] qa;

input [31:0] sa;

input sel;

output reg [31:0] mux\_out;

always@(\*)begin

case(sel)

0:mux\_out <= qa;

1:mux\_out <= sa;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 10:49:36 PM

// Design Name:

// Module Name: SignExtend

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module SignExtend(immIn,sext,immOut);

input [15:0] immIn;

input sext;

output reg [31:0] immOut;

always@(\*)begin

case(sext)

0:immOut <= {16'b0000000000000000,immIn[15:0]};

1:immOut <= {{16{immIn[15]}},immIn[15:0]};

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 09:30:03 PM

// Design Name:

// Module Name: Mux\_Alub

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux\_Alub(qb,imm,sel,mux\_out);

input [31:0] qb;

input [31:0] imm;

input sel;

output reg [31:0] mux\_out;

always@(\*)begin

case(sel)

0:mux\_out <= qb;

1:mux\_out <= imm;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/13/2016 06:13:12 PM

// Design Name:

// Module Name: ALU

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module ALU(ALUctl,qa,qb,ALUOut,z);

input [3:0] ALUctl;

input [31:0] qa,qb;

output reg [31:0] ALUOut;

output z;

assign z = (ALUOut==0); //Zero is true if ALUOut is 0

always@(ALUctl,qa,qb)

begin

case(ALUctl)

4'b0000:ALUOut <= qa + qb; // ADD

4'b0100:ALUOut <= qa - qb; //SUBTRACT

4'b0001:ALUOut <= qa & qb; // AND

4'b0101:ALUOut <= qa | qb; // OR

4'b0010:ALUOut <= qa ^ qb; // XOR

4'b0011:ALUOut <= qb << qa; // SLL

4'b0111:ALUOut <= qb >> qa; // SRL

4'b0110:ALUOut <= qb << 16; // LUI

default:ALUOut <= 0;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/03/2016 05:13:31 PM

// Design Name:

// Module Name: Mux\_WData

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux\_WData(aluOut,do,sel,write\_data);

input [31:0] aluOut;

input [31:0] do;

input sel;

output reg [31:0] write\_data;

always@(\*)begin

case(sel)

0:write\_data <= aluOut;

1:write\_data <= do;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/14/2016 02:06:59 PM

// Design Name:

// Module Name: Mux\_JalToWdata

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux\_JalToWdata(write\_data,p4,jal,mux\_dataout);

input [31:0] write\_data;

input [31:0] p4;

input jal;

output reg [31:0] mux\_dataout;

always@(\*)begin

case(jal)

0:mux\_dataout <= write\_data;

1:mux\_dataout <= p4;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 10:21:32 PM

// Design Name:

// Module Name: branch\_adder

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module branch\_adder(p4,imm,branchOut);

input [31:0] p4;

input [31:0] imm;

output reg [31:0] branchOut;

always@(\*)begin

branchOut <= p4 + imm;

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/27/2016 05:09:26 PM

// Design Name:

// Module Name: Mux4to1

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Mux4to1(p4,bpc,qa,jpc,pcsrc,nextPC);

input [31:0] p4;

input [31:0] bpc;

input [31:0] qa;

input [31:0] jpc;

input [1:0] pcsrc;

output reg [31:0] nextPC;

always@(\*)begin

case(pcsrc)

2'b00: nextPC = p4;

2'b01: nextPC = bpc;

2'b10: nextPC = qa;

2'b11: nextPC = jpc;

endcase

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/13/2016 06:11:44 PM

// Design Name:

// Module Name: inst\_mem

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module inst\_mem(a,inst);

input [31:0] a;

output [31:0] inst;

wire [31:0] rom [0:31];// (pc) label instruction

assign rom[5'h00] = 32'h3c010000; // (00) main: lui $1, 0

assign rom[5'h01] = 32'h34240050; // (04) ori $4, $1, 80

assign rom[5'h02] = 32'h20050004; // (08) addi $5, $0, 4

assign rom[5'h03] = 32'h0c000018; // (0c) call: jal sum

assign rom[5'h04] = 32'hac820000; // (10) sw $2, 0($4)

assign rom[5'h05] = 32'h8c890000; // (14) lw $9, 0($4)

assign rom[5'h06] = 32'h01244022; // (18) sub $8, $9, $4

assign rom[5'h07] = 32'h20050003; // (1c) addi $5, $0, 3

assign rom[5'h08] = 32'h20a5ffff; // (20) loop2: addi $5, $5, -1

assign rom[5'h09] = 32'h34a8ffff; // (24) ori $8, $5, 0xffff

assign rom[5'h0A] = 32'h39085555; // (28) xori $8, $8, 0x5555

assign rom[5'h0B] = 32'h2009ffff; // (2c) addi $9, $0, -1

assign rom[5'h0C] = 32'h312affff; // (30) andi $10,$9, 0xffff

assign rom[5'h0D] = 32'h01493025; // (34) or $6, $10, $9

assign rom[5'h0E] = 32'h01494026; // (38) xor $8, $10, $9

assign rom[5'h0F] = 32'h01463824; // (3c) and $7, $10, $6

assign rom[5'h10] = 32'h10a00001; // (40) beq $5, $0, shift

assign rom[5'h11] = 32'h08000008; // (44) j loop2

assign rom[5'h12] = 32'h2005ffff; // (48) shift: addi $5, $0, -1

assign rom[5'h13] = 32'h000543c0; // (4c) sll $8, $5, 15

assign rom[5'h14] = 32'h00084400; // (50) sll $8, $8, 16

assign rom[5'h15] = 32'h00084403; // (54) sra $8, $8, 16

assign rom[5'h16] = 32'h000843c2; // (58) srl $8, $8, 15

assign rom[5'h17] = 32'h08000017; // (5c) finish: j finish

assign rom[5'h18] = 32'h00004020; // (60) sum: add $8, $0, $0

assign rom[5'h19] = 32'h8c890000; // (64) loop: lw $9, 0($4)

assign rom[5'h1A] = 32'h20840004; // (68) addi $4, $4, 4

assign rom[5'h1B] = 32'h01094020; // (6c) add $8, $8, $9

assign rom[5'h1C] = 32'h20a5ffff; // (70) addi $5, $5, -1

assign rom[5'h1D] = 32'h14a0fffb; // (74) bne $5, $0, loop

assign rom[5'h1E] = 32'h00081000; // (78) sll $2, $8, 0

assign rom[5'h1F] = 32'h03e00008; // (7c) jr $31

assign inst = rom[a[6:2]];

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/02/2016 11:04:12 PM

// Design Name:

// Module Name: data\_mem

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module data\_mem(addr,datain,dataout,we,clk);

input clk;

input we;

input [31:0] addr;

input [31:0] datain;

output [31:0] dataout;

reg [31:0] ram [0:31];

assign dataout = ram[addr[6:2]];

always@(posedge clk)

if(we) ram[addr[6:2]] = datain;

integer i;

initial begin // initialize memory

for (i = 0; i < 32; i = i + 1)

ram[i] = 0;

// ram[word\_addr] = data // (byte\_addr)

ram[5'h14] = 32'h000000a3; // (50hex)

ram[5'h15] = 32'h00000027; // (54hex)

ram[5'h16] = 32'h00000079; // (58hex)

ram[5'h17] = 32'h00000115; // (5chex)

ram[5'h18] = 32'h00000258; // the sum stored by sw instruction

end

endmodule

b.   Your Verilog Test Bench code. Add “`timescale 1ns/1ps” as the first line of your test bench file.

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 11/27/2016 07:03:02 PM

// Design Name:

// Module Name: sccomp\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module sccomp\_tb;

reg clk = 1,clrn = 1;

wire [31:0] pc;

// wire [31:0] inst;

wire [31:0] aluout;

wire [31:0] memout;

sccomp sccomp\_tb(clk,clrn,pc,aluout,memout);

always

#10 clk = ~clk;

initial

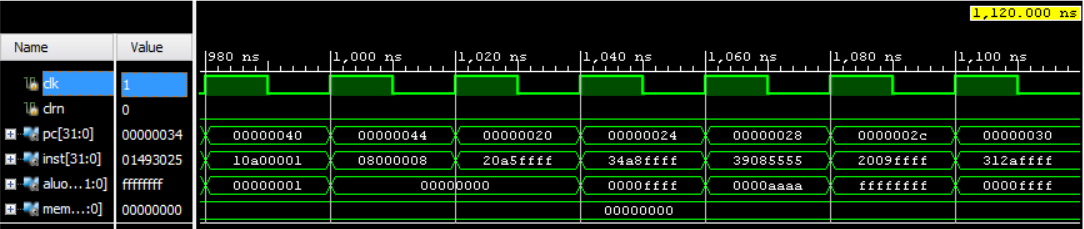
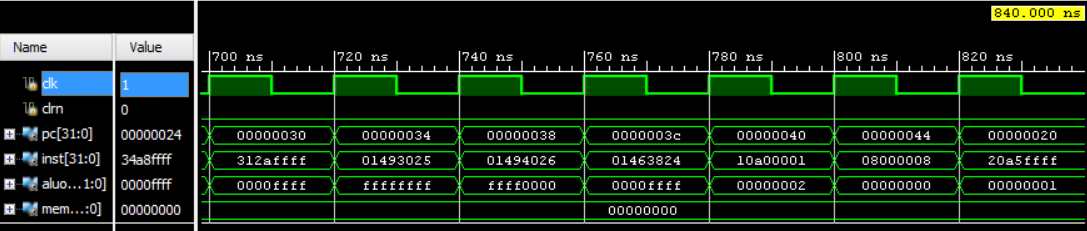
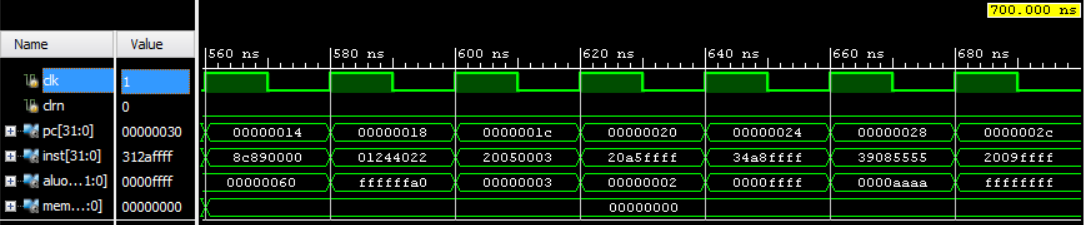
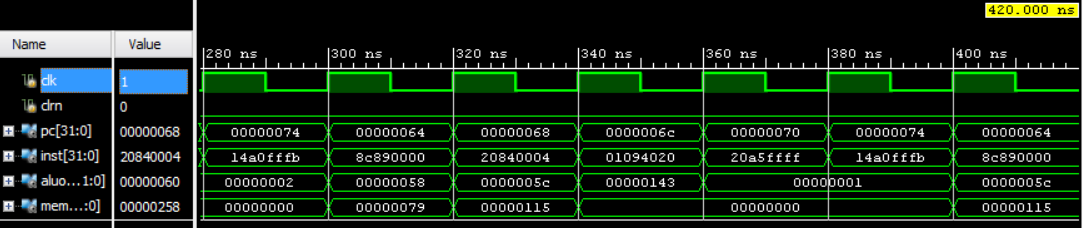
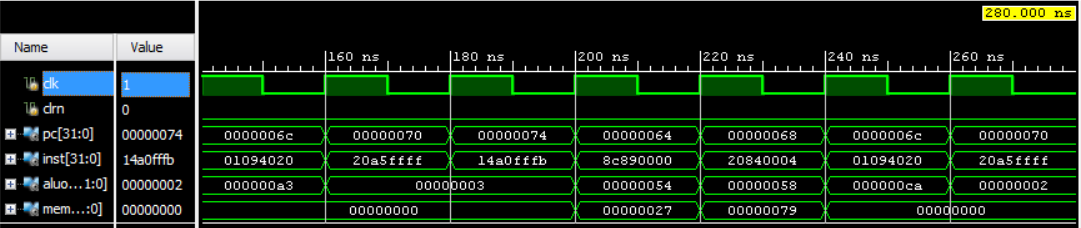
begin

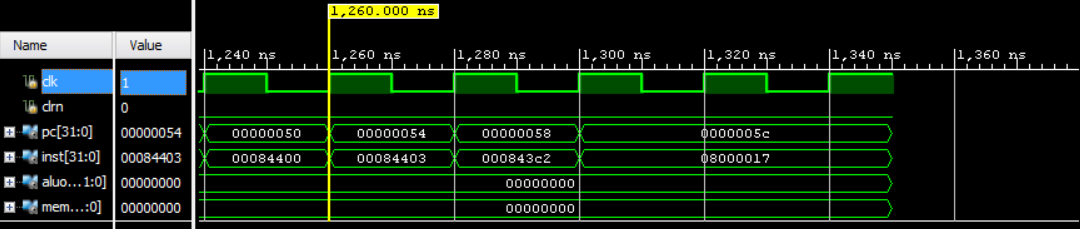
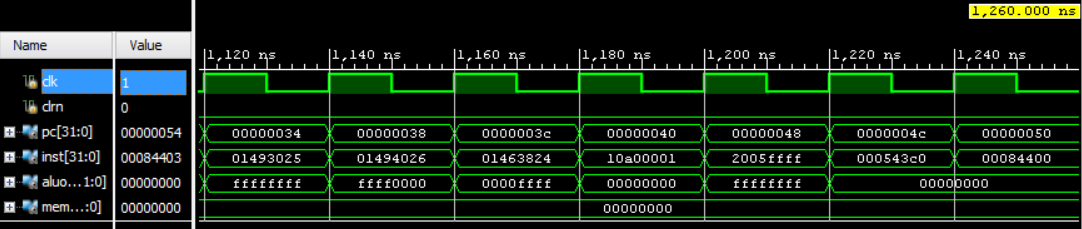
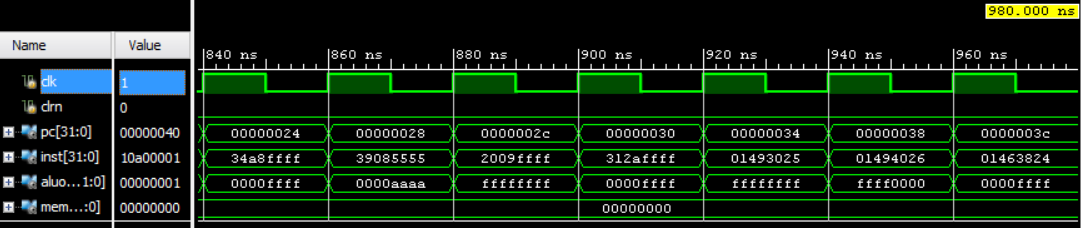
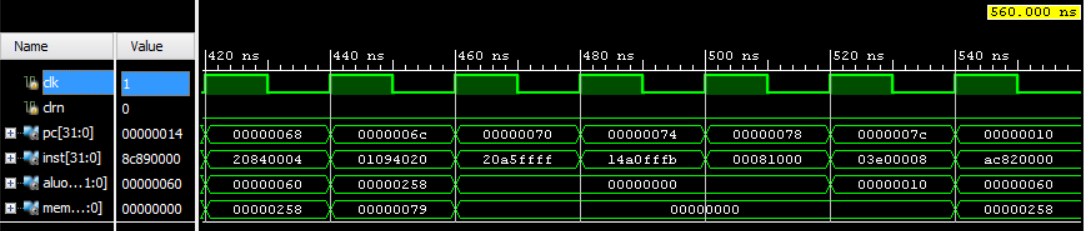
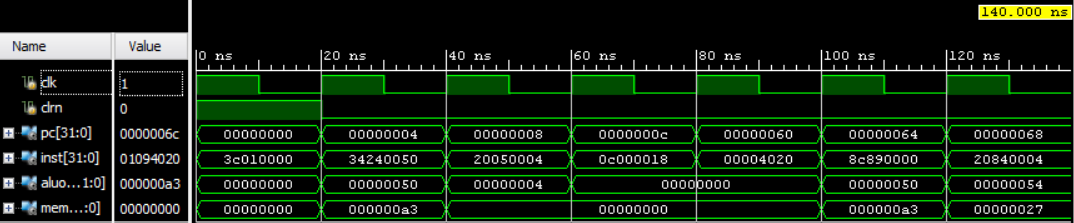
#20 clrn = 0;

end

endmodule

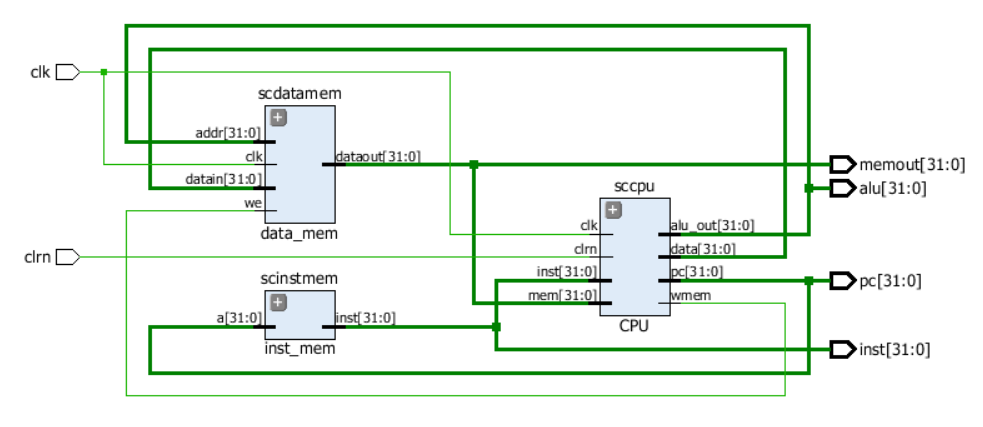
**c.   The waveforms resulting from the verification of your design. Figure 3 shows the simulation till ~ 275 ns, you need to show the simulation until 1350 ns. The clock period is 20 ns as shown in Figure 3. The memout signal in Figure 3 is the dataout signal in Figure 2.**



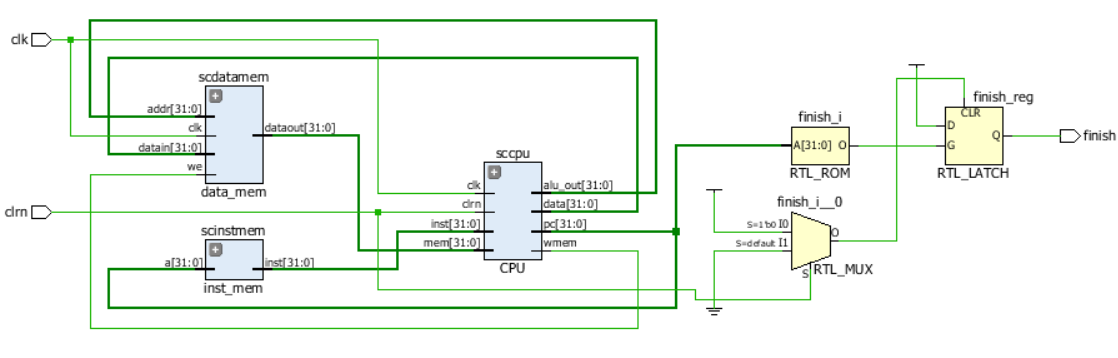


**d.   The design schematics from the Xilinx synthesis of your design. Do not use any area constraints.**

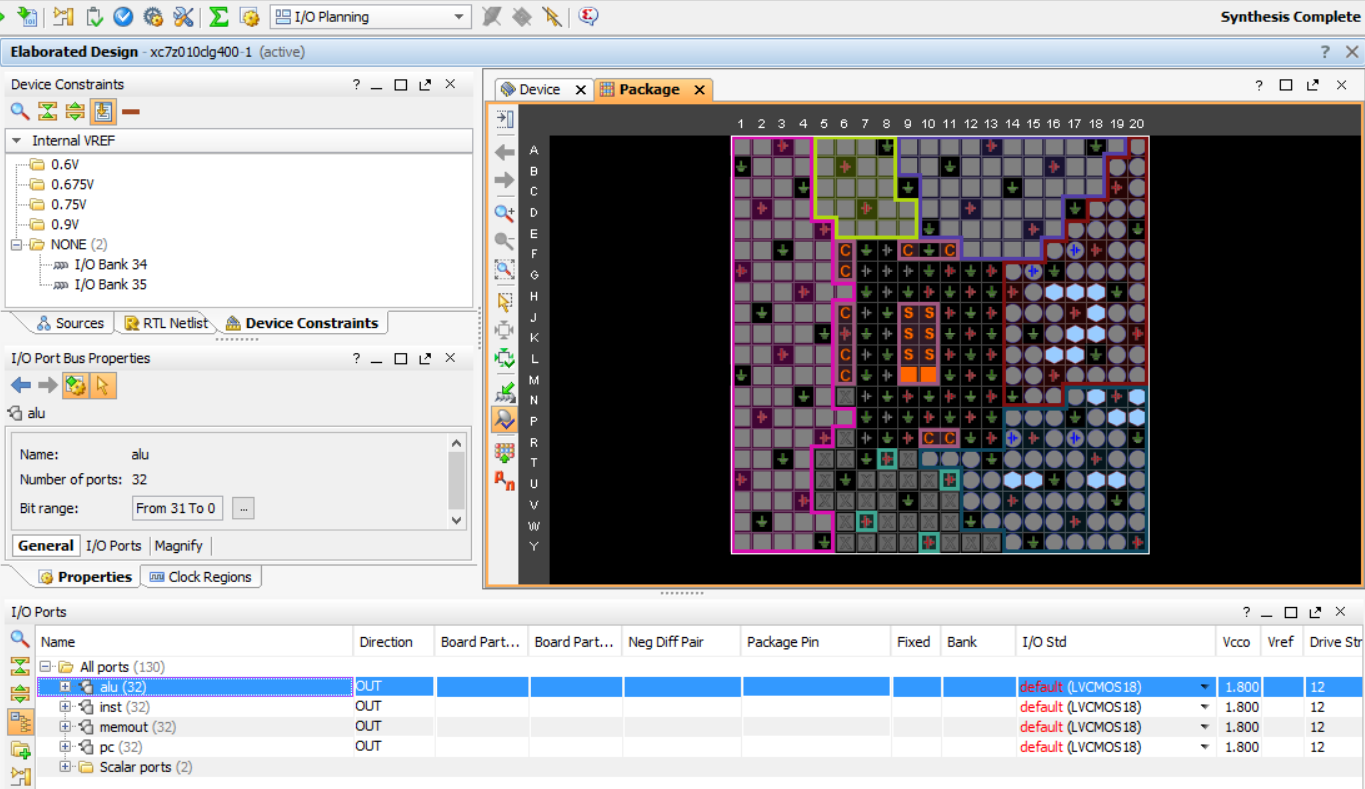
Design schematic for all output signals shown in the simulation waveform:



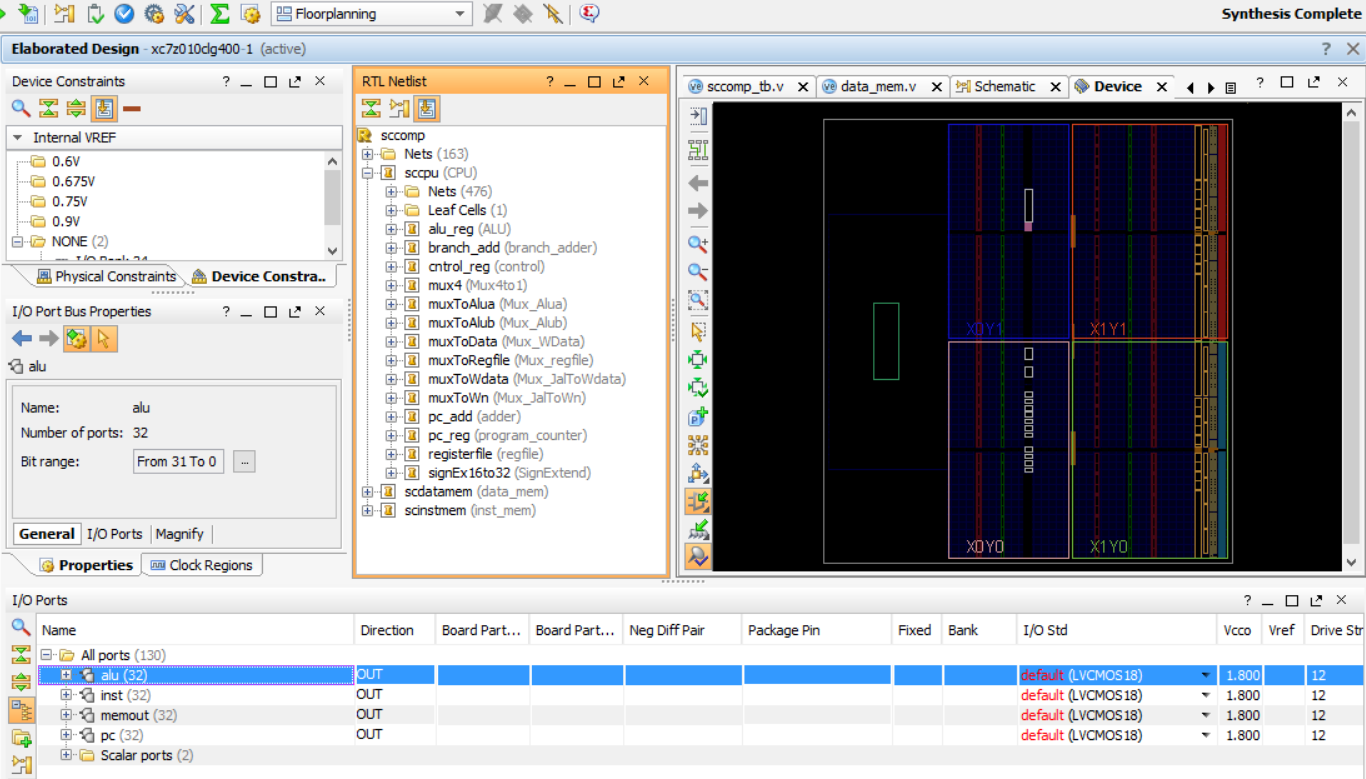
Design schematic for ZyboBoard with an input “clrn” to control the beginning of fetching of the instruction, and an output “finish” to control the logic LED to light on:



**e.   Snapshot of the I/O Planning**

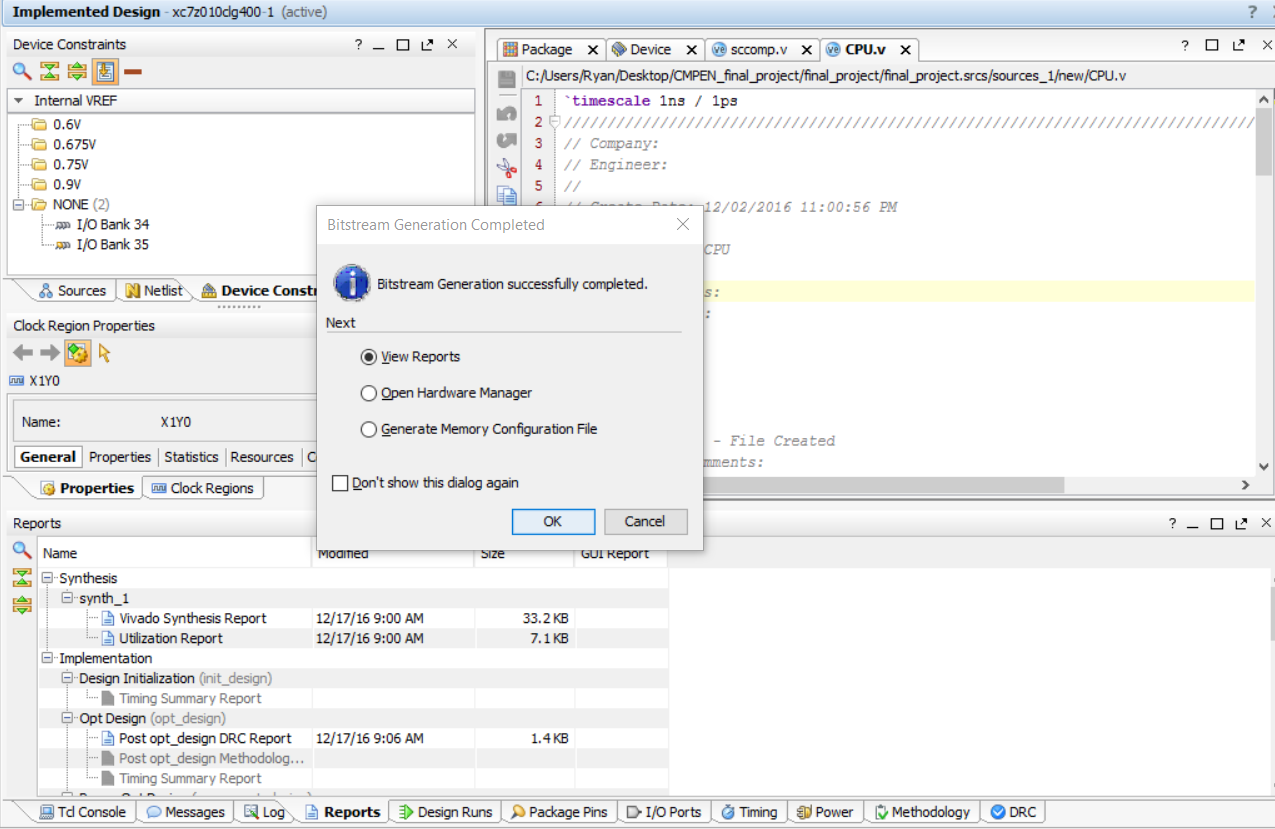


**f.   Snapshot of the floor planning**



**g.   Generate the bitstream.**

**h.   The design should be free from errors when synthesized, implemented and generated of bit stream. i.   Connect the board and power it ON. Open a hardware session, and program the FPGA. Make sure that the micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Make sure that the jumper on the board is set to select USB power.**



**j.   Select the Open Hardware Manager option and click OK.**

**k.   Click on the Open target link, then Auto Connect from the dropdown menu.**

**l.   The Hardware Session status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.**

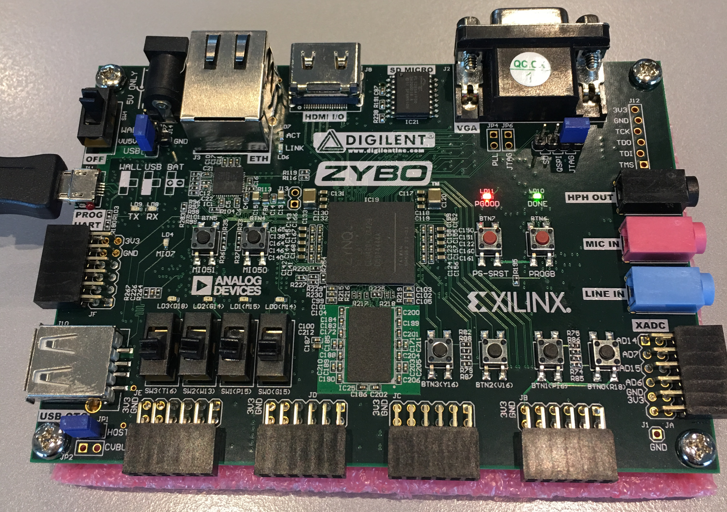
**m.  Select the device in the Hardware Device Properties, and verify that the (.bit) file is selected as the programming file in the General tab.**

**n.   Choose sw0 of the logic slide switches (item number 9, page 7, lecture 21) as an input to control the beginning of fetching of the instructions.**

**o.   After finishing the last instruction, choose LED 0 of the logic LEDs (item number 8, page 7, lecture 21) to light on.**

**p.   You need to take pictures of the ZyboBoard to show that the download is done without problems and to show the configurations described in items n and o. Failed to do that will result in 15 points deduction of your project grade.**

ZyboBoard with download done successfully:



ZyboBoard with configurations that a sw0 of the logic slide switches as an input to control the beginning of fetching of the instruction, and a LED 0 of the logic LEDs to light on.

In the fact that sw0 is turned on, the result of LED 0 is light on means the device is programmed well and download is done without any problems.

