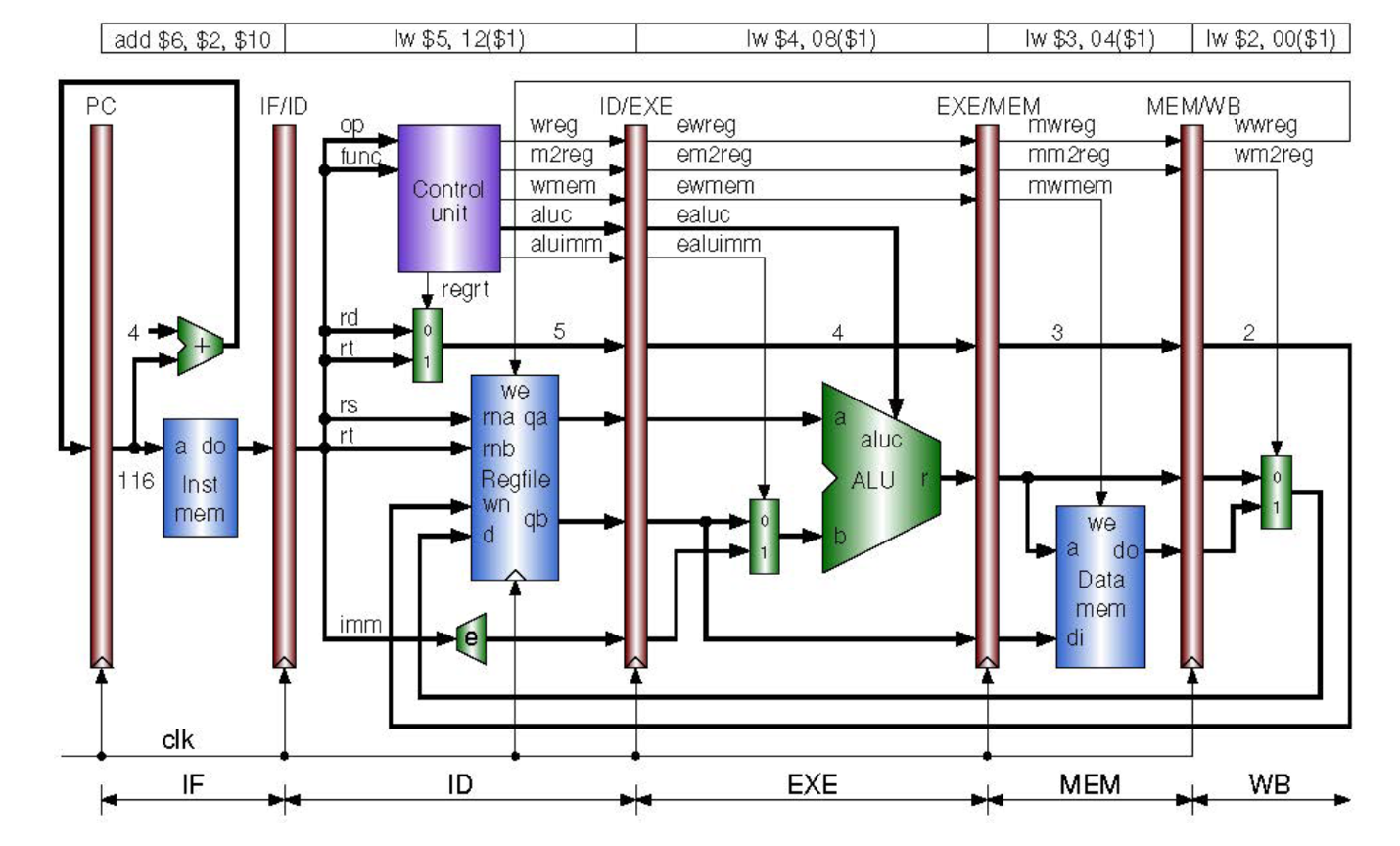
May 4, 2017

**ABSTRACTION:**

In this project, we are programming a five-stage MIPS pipelining CPU using Verilog and downloading it to a ZYBO-7000 development FPGA. We use switch 0 on the ZYBO board to start fetching the instruction and use LED 0 to report when processing is done. We tested out five R-TYPE instructions to examine our program and added stalls to prevent potential data hazard.

**INTRODUCTION:**



Five-stage pipelining uses five “pipelines” to store instruction/data so that it promises processing multiple instructions on the same clock cycle. In our design, we have five Verilog modules, PC, IF/ID, ID/EXE, EXE/MEM, MEM/WB to stimulate MIPS pipelining so that: 1. Instruction is fetched in PC stage; 2: Instruction is decoded and supporting data is read and written in the register; 3: instruction is processed and results are sent to the fourth stage; 4: data is written to or read from the data memory; 5: LW like instruction data is written back to stage 2. All instructions for the next cycle are stored in PC, IF/ID, ID/EXE, EXE/MEM, MEM/WB.

By using this methodology, we can maximize the use of each part of the CPU and process as many as 5 instructions in one clock cycle. However, correct data might not be available before the write back (5) stage, thus we add stalls to hold the pc counter in order to solve potential data hazard.

VERILOG IMPLEMENTATION

1. Device used: XC7Z010- -1CLG400C
2. Verilog Code

`timescale 1ns / 1ps

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

// Company: Penn State

// Engineer: Tianjian Gao

//

// Create Date: 04/16/2017 8:00:00 PM

// Design Name: MIPS Mini CPU DESIGN

// Module Name: All Five Stages

// Project Name: CMsPEN 331 LAB6

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module miniCPU(clk, sw, led);

input clk;

input sw;

output led;

wire [31:0] pc\_in, pc\_out, inst;

wire [31:0] ir;

wire [5:0] op, func;

wire [4:0] rs, rt, rd;

wire [15:0] imm;

assign op = ir[31:26];

assign rs = ir[25:21];

assign rt = ir[20:16];

assign rd = ir[15:11];

assign imm = ir[15:0];

assign func = ir[5:0];

wire wreg, m2reg, wmem,aluimm;

wire [3:0] aluc;

wire [4:0] mux5;

wire [31:0] eimm;

wire [31:0] qa, qb;

wire regrt;

assign eimm = {{16{imm[15]}},imm[15:0]};

wire ewreg, em2reg, ewmem, ealuimm;

wire [3:0] ealuc;

wire [4:0] mux4;

wire [31:0] aluout;

wire [31:0] exqa, exqb, exeimm;

wire [31:0] aluin\_b;

wire mwreg, mm2reg;

wire [4:0] mux3;

wire [31:0] aluout\_mem;

wire [31:0] do;

wire [31:0] di;

wire mwmem;

wire wwreg, wm2reg;

wire [4:0] mux2;

wire [31:0] mem\_mux0, mem\_mux1;

wire [31:0] wout;

wire start;

assign start = sw;

wire done;

assign led = done;

//Stage1

PC pc1 (.clk(clk),.start(start),.pc(pc\_out),.next\_pc(pc\_in),.done(done));

INCREMENT\_PC inc\_pc (.pc(pc\_out), .next\_pc(pc\_in));

InstMem fetch(.pc(pc\_out),.inst(inst));

//Stage2

IFID ifid (.clk(clk), .inst\_in(inst), .inst\_out(ir));

ControlUnit ctrlu(.op(op),.func(func),.wreg(wreg),.m2reg(m2reg),.wmem(wmem),

.aluc(aluc),.aluimm(aluimm),.regrt(regrt));

MUX5 mux\_id(.select(regrt),.a(rd),.b(rt),.out(mux5));

Register regout(.clk(clk), .we(wwreg), .rna(rs), .rnb(rt), .wn(mux2), .d(wout), .qa(qa), .qb(qb));

//Note: in the final project, RegFile is moved to the write back stage, but in verilog it works the same nonetheless.

//Stage3

IDEXE idexe(.clk(clk),.wreg(wreg), .m2reg(m2reg), .wmem(wmem), .aluc(aluc), .aluimm(aluimm), .mux5(mux5),

.qa(qa), .qb(qb), .eimm(eimm), .ewreg(ewreg), .em2reg(em2reg), .ewmem(ewmem), .ealuc(ealuc), .ealuimm(ealuimm), .mux4(mux4),

.exqa(exqa), .exqb(exqb), .exeimm(exeimm));

MUX32 mux\_exe(.select(ealuimm),.a(exqb),.b(exeimm),.out(aluin\_b));

ALU alu (.ALUCtrl(ealuc), .A(exqa) , .B(aluin\_b), .ALUOut(aluout));

//Stage4

EXEMEM exemem(.clk(clk), .ewreg(ewreg), .em2reg(em2reg), .ewmem(ewmem), .mux4(mux4), .aluout(aluout), .exqb(exqb),

.mwreg(mwreg), .mm2reg(mm2reg), .mwmem(mwmem), .mux3(mux3), .aluout\_mem(aluout\_mem), .di(di));

DataMem datamem(.we(mwmem), .a(aluout\_mem), .di(di), .do(do));

//Stage5

MEMWB memwb (.clk(clk), .mwreg(mwreg), .mm2reg(mm2reg), .mux3(mux3), .aluout\_mem(aluout\_mem), .do(do),

.wwreg(wwreg), .wm2reg(wm2reg), .mux2(mux2), .mem\_mux0(mem\_mux0),.mem\_mux1(mem\_mux1));

MUX32 mux\_wb(.select(wm2reg),.a(mem\_mux0),.b(mem\_mux1),.out(wout));

endmodule

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

module PC(clk, start, pc, next\_pc, done);

input clk, start;

input [31:0] next\_pc;

output reg [31:0] pc;

output reg done;

integer stall = 0;

integer first = 1;

always@(posedge clk)

begin

if (start == 0)

begin

pc <= 0;

first <= 1;

done <= 0;

end

else

begin

if (first == 1)

begin

pc <= 96;

first <= 0;

end

else if (pc == 100 && stall == 0)

begin

pc <= 0;

stall <= stall + 1;

end

else if (pc == 0 && stall < 2)

begin

stall <= stall + 1;

end

else if (pc == 0 && stall == 2)

begin

pc <= 104;

stall <= stall + 1;

end

else

begin

pc <= next\_pc;

end

if (pc >= 136)

begin

done <= 1;

end

end

end

endmodule

module INCREMENT\_PC(pc, next\_pc);

input [31:0] pc;

output [31:0] next\_pc;

assign next\_pc = pc + 4;

endmodule

module IFID(clk, inst\_in, inst\_out);

input clk;

input [31:0] inst\_in;

output reg [31:0] inst\_out;

always@(posedge clk)

begin

inst\_out <= inst\_in;

end

endmodule

module IDEXE(clk,wreg, m2reg, wmem, aluc, aluimm, mux5, qa, qb, eimm,

ewreg, em2reg, ewmem, ealuc, ealuimm, mux4, exqa, exqb, exeimm);

input clk, wreg, m2reg, wmem, aluimm;

input [3:0] aluc;

input [4:0] mux5;

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

output reg ewreg, em2reg, ewmem, ealuimm;

output reg [3:0] ealuc;

output reg [31:0] exqa, exqb, exeimm;

output reg [4:0] mux4;

always @(posedge clk)

begin

ewreg <= wreg;

em2reg <= m2reg;

ewmem <= wmem;

ealuimm <= aluimm;

ealuc <= aluc;

mux4 <= mux5;

exqa <= qa;

exqb <= qb;

exeimm <= eimm;

end

endmodule

module EXEMEM(clk, ewreg, em2reg, ewmem, mux4, aluout, exqb,

mwreg, mm2reg, mwmem, mux3, aluout\_mem, di);

input clk, ewreg, em2reg, ewmem;

input [4:0] mux4;

input [31:0] aluout, exqb;

output reg mwreg, mm2reg, mwmem;

output reg [4:0] mux3;

output reg [31:0] aluout\_mem, di;

always @(posedge clk)

begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

mux3 <= mux4;

aluout\_mem <= aluout;

di <= exqb;

end

endmodule

module MEMWB(clk, mwreg, mm2reg, mux3, aluout\_mem, do,

wwreg, wm2reg, mux2, mem\_mux0,mem\_mux1);

input clk, mwreg, mm2reg;

input [4:0] mux3;

input [31:0] aluout\_mem, do;

output reg wwreg, wm2reg;

output reg [4:0] mux2;

output reg [31:0] mem\_mux0, mem\_mux1;

always@(posedge clk)

begin

wwreg <= mwreg;

wm2reg <= mm2reg;

mux2 <= mux3;

mem\_mux0 <= aluout\_mem;

mem\_mux1 <= do;

end

endmodule

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

module MUX5(select, a, b, out);

input select;

input [4:0] a, b;

output reg [4:0] out;

always @(select or a or b)

begin

out <= select ? b : a;

end

endmodule

module MUX32(select, a, b, out);

input select;

input [31:0] a, b;

output reg [31:0] out;

always @(select or a or b)

begin

out <= select ? b : a;

end

endmodule

module InstMem(pc, inst);

input [31:0] pc;

output reg [31:0] inst;

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

initial begin

mem[25] = 32'b00000000001000100001100000100000;

mem[26] = 32'b00000001001000110010000000100010;

mem[27] = 32'b00000000011010010010100000100101;

mem[28] = 32'b00000000011010010011000000100110;

mem[29] = 32'b00000000011010010011100000100100;

end

always@(pc)

begin

inst <= mem[pc>>2];

end

endmodule

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

input [5:0] op, func;

output reg wreg, m2reg, wmem, aluimm, regrt;

output reg [3:0] aluc;

always@(op or func)begin

if (op==6'd35) begin

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0000;

aluimm <= 1;

regrt <= 1;

end

else if (op==6'd0 && func == 6'd32) begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0000;

aluimm <= 0;

regrt <= 0;

end

else if (op==6'd0 && func == 6'd34) begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0001;

aluimm <= 0;

regrt <= 0;

end

else if (op==6'd0 && func == 6'd37) begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 0;

regrt <= 0;

end

else if (op==6'd0 && func == 6'd38) begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0011;

aluimm <= 0;

regrt <= 0;

end

else if (op==6'd0 && func == 6'd36) begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0100;

aluimm <= 0;

regrt <= 0;

end

end

endmodule

module Register(clk, we, rna, rnb, wn, d, qa, qb); //..correct

input clk, we;

input [4:0]rna, rnb;

input [4:0] wn;

input [31:0] d;

output [31:0] qa, qb;

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

integer it;

initial begin

for(it = 0; it < 31; it = it + 1)

RegFile[it] = 0;

RegFile[0] = 32'hA00000AA;

RegFile[1] = 32'h10000011;

RegFile[2] = 32'h20000022;

RegFile[3] = 32'h30000033;

RegFile[4] = 32'h40000044;

RegFile[5] = 32'h50000055;

RegFile[6] = 32'h60000066;

RegFile[7] = 32'h70000077;

RegFile[8] = 32'h80000088;

RegFile[9] = 32'h90000099;

end

assign qa = RegFile[rna];

assign qb = RegFile[rnb];

always@(posedge clk) begin

if (we)

RegFile[wn] <= d;

end

endmodule

module ALU (ALUCtrl, A, B, ALUOut);

input [3:0] ALUCtrl;

input [31:0] A, B;

output reg [31:0] ALUOut;

always @(A or B or ALUCtrl)

begin

case (ALUCtrl)

4'b0000: ALUOut <= A + B;

4'b0001: ALUOut <= A - B;

4'b0010: ALUOut <= A | B;

4'b0011: ALUOut <= A ^ B;

4'B0100: ALUOut <= A & B;

endcase

end

endmodule

module DataMem(we, a, di, do);

input [31:0] a, di;

output reg [31:0] do;

input we;

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

integer it;

initial begin

for(it = 0; it < 256; it = it + 1)

DataMem[it] = 0;

end

always@(\*)

begin

if (we) begin

DataMem[a>>2] <= di;

do <= 0;

end

else begin

do <= DataMem[a>>2][31:0];

end

end

endmodule

1. Testbench Code

`timescale 1ns / 1ps

module testbench();

reg clk;

reg sw;

wire led;

miniCPU cpu(.clk(clk),.sw(sw),.led(led));

initial begin

sw = 0;

clk = 1;

#200 sw = 1;

end

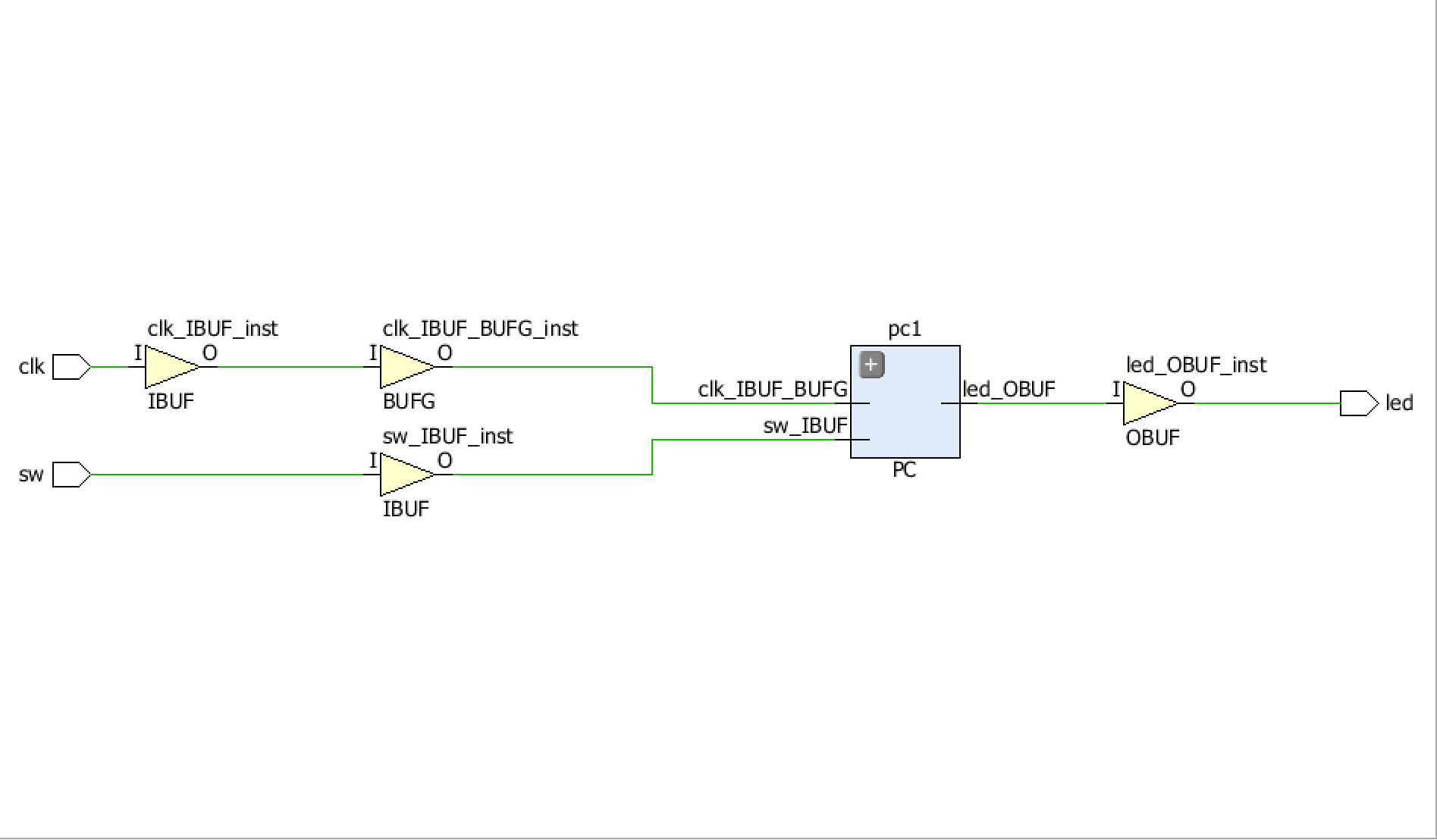
always begin

#5 clk=~clk;

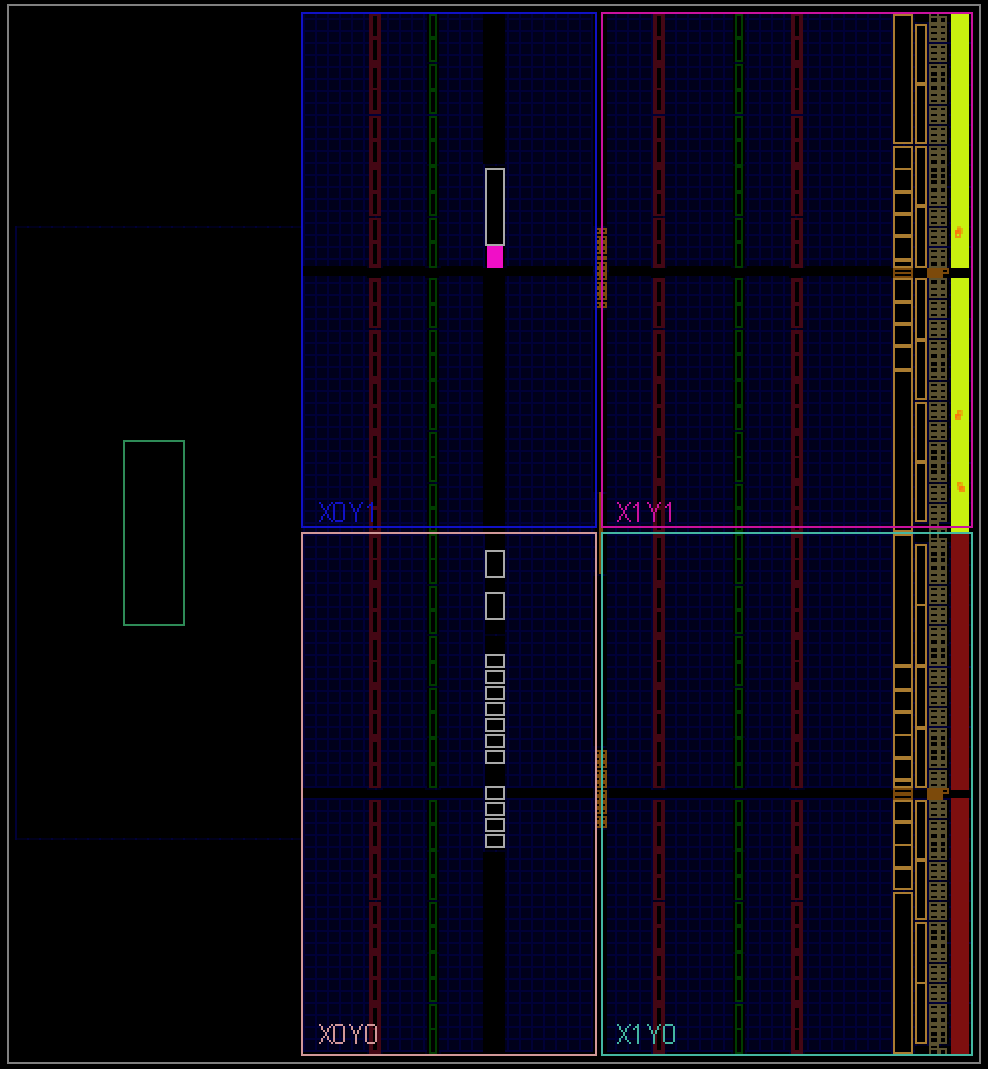
end

endmodule

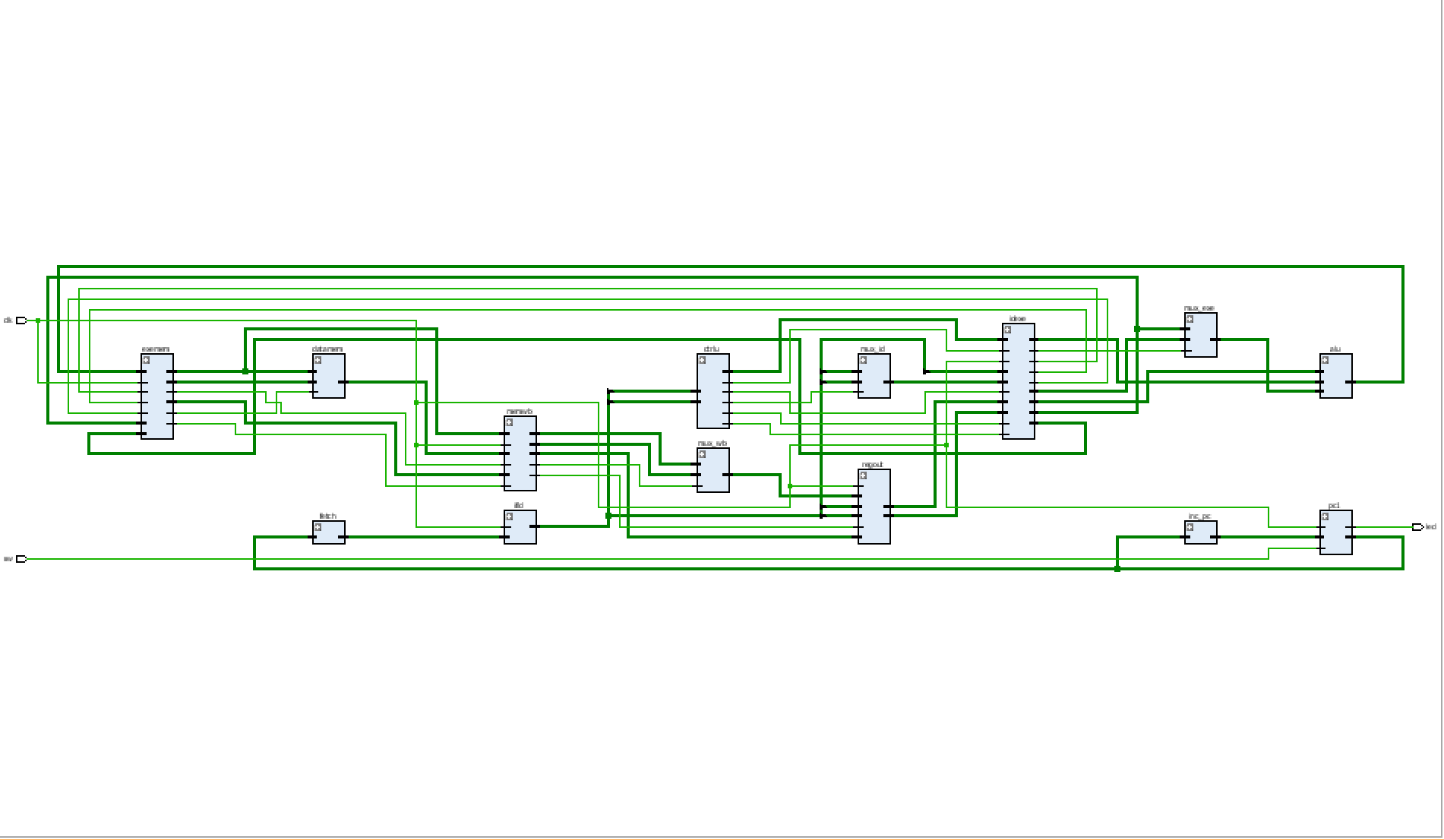
1. I/O PLANNING



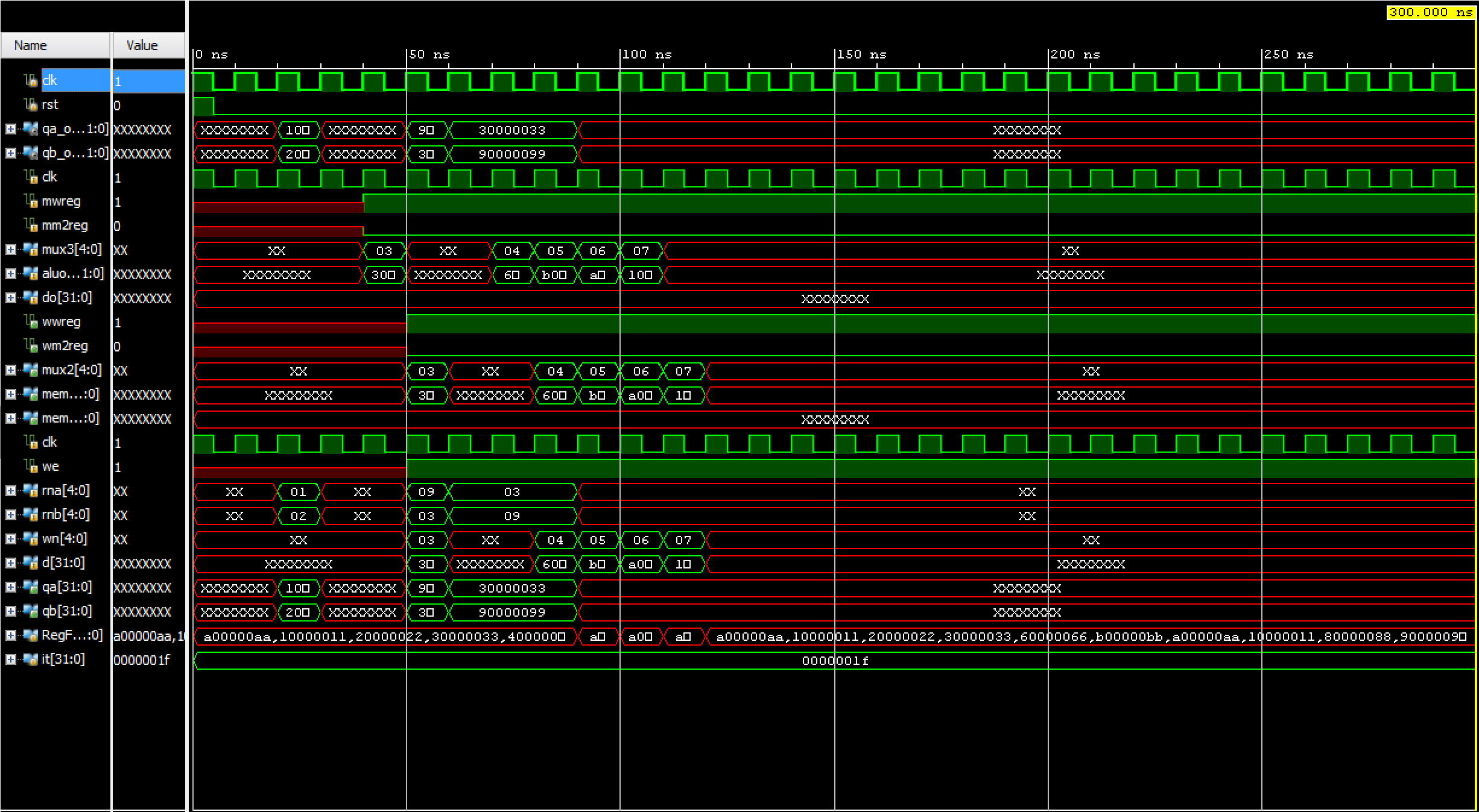
1. FLOOR PLANNING



1. SCHEMATIC

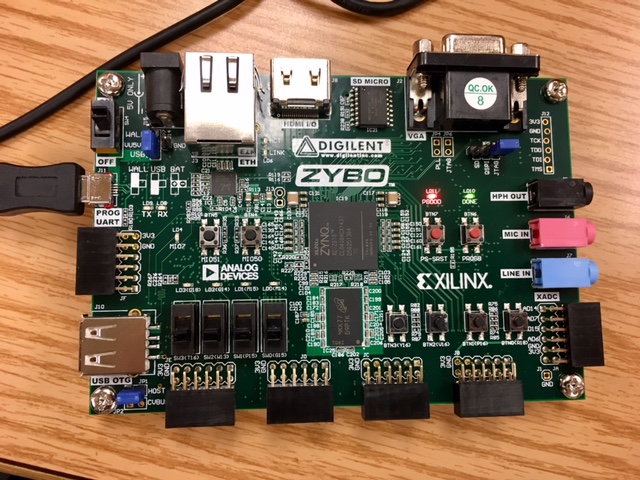


1. WAVEFROM



1. FPGA PROGRAMMING RESULT

SWITCH OFF:



SWITCH ON, PROGRAM BEGINS. LED ON.

