# CMPEN 331

# Final Project Report

# Iangting Lin (ikl5009)

# Kaixuan Meng (kbm5393)

## Abstract

The idea of this project is to based on what we’ve learned throughout the semester and implement a pipeline CPU with Vivado. The project includes 5 stages of the pipeline and a testbench. We’ve learned how each stage works and the connection between them for a CPU in order to function.

## Introduction

## Capture

We are designing our code based on the diagram above. In the IF stage, we are only updating pc and fetching instruction. In ID stage, there are control unit, MUX, register file, and sign extention. In the EXE stage, we have MUX and ALU and in MEM we have the data memory. In WB stage, we have another MUX and the register file again to write back. In the cpu module, we connect all the variables to

## Verilog Design

module cpu(rst,clk,endnote);

input clk, rst;

output wire endnote;

wire mwreg,mm2reg;

wire[4:0]MEMrd\_out;

wire[31:0]MEMalur\_out,dataout;

wire endsignal,endsignalEXE,MEMendsignal;

wire [31:0]qa,qb,imm\_out,alu\_r,EXEqb\_out;

wire [31:0]instr;

wire wreg,m2reg,wmem,aluimm,ewreg,em2reg,ewmem;

wire [3:0]aluc;

wire[4:0]IDrd\_out,EXErd\_out;

IF aa(rst,clk,instr);

ID bb(rst,clk,instr,wreg,m2reg,wmem,aluc,aluimm,IDrd\_out,qa,qb,imm\_out,endsignal);

EXE cc(endsignal,rst,clk,wreg,m2reg,wmem,aluc,aluimm,IDrd\_out,qa,qb,imm\_out,ewreg,em2reg,ewmem,EXErd\_out,alu\_r,EXEqb\_out,endsignalEXE);

MEM dd(endsignalEXE,rst, clk,ewreg,em2reg,ewmem,EXErd\_out,alu\_r,EXEqb\_out,mwreg,mm2reg,MEMrd\_out,MEMalur\_out,dataout,MEMendsignal);

WB ee(MEMendsignal,rst,clk,mwreg,mm2reg,MEMrd\_out,MEMalur\_out,dataout,endnote);

endmodule

module IF(rst,clk,instr);

input clk, rst;

output reg [31:0] instr;

reg [31:0] pc;

reg [31:0] temp;

reg[5:0] newop, oldop;

reg [4:0]rd,rt,rs, oldrt;

integer i=0;

parameter INCREMENT\_AMOUNT = 32'd4;

reg [31:0] RAM[0:256];

parameter add=6'd0, addi=6'b001000, andi=6'b001100, ori=6'b001101, lw=6'b100011, sw=6'b101011;

initial begin

RAM[25]=32'h221820;

RAM[26]=32'h1232022;

RAM[27]=32'h692825;

RAM[28]=32'h693026;

RAM[29]=32'h693824;

RAM[30]=32'hFFFFFFFF;

end

always @(posedge clk or posedge rst)

begin

if (rst)

pc <= 32'd100;

else

if(i==0)

pc <= pc + INCREMENT\_AMOUNT;

end

always@(pc) begin

temp<=RAM[pc >>2];

end

always@(temp) begin

newop<=temp[31:26];

rt<=temp[20:16];

rs<=temp[25:21];

case(oldop)

add: begin

case(newop)

add: begin

if(rd==(rt|rs))

i<=2;

end

andi,ori,sw,lw:begin

if(rd==rs)

i<=2;

end

endcase

end

andi,ori,sw,lw:begin

case(newop)

add: begin

if(oldrt==(rt|rs))

i<=2;

end

andi,ori,sw,lw:begin

if(oldrt==rs)

i<=2;

end

endcase

end

endcase

end

always@(posedge clk) begin

if (i!=0) begin

instr<=32'hxxxxxxxx;

i<=i-1;

end

else begin

instr<=temp;

end

end

always@(instr) begin

rd<=temp[15:11];

oldop<=temp[31:26];

oldrt<=temp[20:16];

end

endmodule

module ID(reset,clock,IDreg,wreg,m2reg,wmem,aluc,aluimm,rd\_out,qa,qb,imm\_out,endsignal);

input reset;

input clock;

input [31:0]IDreg;

output wire wreg,m2reg,wmem,aluimm;

output wire [4:0] rd\_out;

output wire[3:0] aluc;

output wire [31:0] qa, qb,imm\_out;

output reg endsignal;

reg [31:0]Instr;

wire [5:0]op,func;

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

wire [15:0]imm;

wire regrt, RegWrite;

wire [31:0]WriteData;

always @(posedge clock)

begin

if(reset)

Instr<=32'hxxxxxxxx;

else

Instr<=IDreg;

if (Instr==32'hffffffff)

endsignal<=1;

end

assign RegWrite=1'b0;

assign WriteData=32'hxxxxxxxx;

assign WriteReg=5'bxxxxx;

assign op=Instr[31:26];

assign func=Instr[5:0];

assign rd=Instr[15:11];

assign rt=Instr[20:16];

assign rs=Instr[25:21];

assign imm=Instr[15:0];

Control\_Unit aaa(op,func,wreg,m2reg,wmem,aluc,aluimm, regrt);

MUX bbb(regrt, rd, rt,rd\_out);

registerfile ccc(rs,rt,WriteReg,WriteData,RegWrite,qa,qb,clock);

sign\_extension ddd(imm,imm\_out);

endmodule

module Control\_Unit(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;

parameter [3:0] ADD=4'b0010, SUB=4'b0110,AND=4'b0000, OR=4'b0001,XOR=4'b1111;

always @(op, func)

begin

case(op)

6'd0:begin

wreg=1; regrt=0; m2reg=0; aluimm=0; wmem=0;

case(func)

6'b100000: begin aluc=ADD; end //add

6'b100010:begin aluc=SUB;end //sub

6'b100100:begin aluc=AND;end//and

6'b100101:begin aluc=OR;end //or

6'b100110:begin aluc=XOR; end

6'b101010:begin aluc=ADD; end//slt

endcase

end

6'b001000:begin wreg=1;regrt=1;m2reg=0;aluimm=1;aluc=ADD; wmem=0; end //addi

6'b001100:begin wreg=1;regrt=1; m2reg=0;aluimm=0;aluc=AND; wmem=0; end //andi

6'b001101:begin wreg=1;regrt=1;m2reg=0;aluimm=1;aluc=OR; wmem=0; end //ori

6'b100011:begin wreg=1;regrt=1;m2reg=1;aluimm=1;aluc=ADD; wmem=0; end //lw

6'b101011:begin wreg=0;regrt=1'bx;;m2reg=1'bx;aluimm=1;;aluc=ADD; wmem=1; end //sw

6'bxxxxxx:begin wreg=1'bx;regrt=1'bx;;m2reg=1'bx;aluimm=1'bx;;aluc=4'bxxxx; wmem=1'bx; end

endcase

end

Endmodule

module MUX(regrt, rd, rt, out1);

input regrt;

input [4:0] rd, rt;

output reg [4:0] out1;

always@ (rd or rt or regrt) begin

if(regrt)

out1<=rt;

else

out1<=rd;

end

endmodule

module registerfile(Read1,Read2,WriteReg,WriteData,RegWrite, Data1, Data2, clk);

input [4:0] Read1, Read2, WriteReg;

input [31:0] WriteData;

input RegWrite, clk;

output [31:0] Data1, Data2;

reg [31:0] RF[31:0];

integer i;

initial begin

RF[0]=32'hA00000AA;

RF[1]=32'h10000011;

RF[2]=32'h20000022;

RF[3]=32'h30000033;

RF[4]=32'h40000044;

RF[5]=32'h50000055;

RF[6]=32'h60000066;

RF[7]=32'h70000077;

RF[8]=32'h80000088;

RF[9]=32'h90000099;

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

RF[i]=0;

end

assign Data1=RF[Read1];

assign Data2=RF[Read2];

always @(posedge clk)

begin

if(RegWrite)

RF[WriteReg]<=WriteData;

end

endmodule

module sign\_extension(imm, out1);

input [15:0] imm;

output reg [31:0] out1;

always @(imm)

begin

out1<={{16{imm[15]}}, imm[15:0]};

end

endmodule

module EXE(signal,rst,clk,wreg,m2reg,wmem,aluc,aluimm,rd\_in,qa\_in,qb\_in,imm\_in,ewreg,em2reg,ewmem,rd\_out,alu\_r,qb\_out, endsignal);

input signal;

input rst;

input clk;

input wreg,m2reg,wmem,aluimm;

input[3:0] aluc;

input[31:0] qa\_in,qb\_in,imm\_in;

input [4:0] rd\_in;

output reg ewreg,em2reg,ewmem;

output reg [4:0] rd\_out;

output reg[31:0] qb\_out;

output wire[31:0]alu\_r;

output reg endsignal;

reg [3:0] ealuc;

reg [31:0] qa\_wire, qb\_wire,imm\_wire;

reg ealuimm;

wire [31:0]mux\_r;

always @(posedge clk or posedge rst)

begin

if(rst) begin

ewreg<=1'bx;

em2reg<=1'bx;

ewmem<=1'bx;

ealuimm<=1'bx;

ealuc<=4'bxxxx;

rd\_out<=4'bxxxx;

qa\_wire<=32'hxxxxxxxx;

qb\_wire<=32'hxxxxxxxx;

imm\_wire<=32'hxxxxxxxx;

qb\_out<=32'hxxxxxxxx;

end

else begin

ewreg<=wreg;

em2reg<=m2reg;

ewmem<=wmem;

ealuimm<=aluimm;

ealuc<=aluc;

rd\_out<=rd\_in;

qa\_wire<=qa\_in;

qb\_wire<=qb\_in;

imm\_wire<=imm\_in;

qb\_out<=qb\_in;

endsignal<=signal;

end

end

MUX\_32bit xx(ealuimm,qb\_wire, imm\_wire, mux\_r);

ALU yy(ealuc,qa\_wire,mux\_r,alu\_r);

endmodule

module MUX\_32bit(c,a,b,r);

input c;

input [31:0] a,b;

output reg [31:0] r;

always@ (a or b or c) begin

if(c)

r<=b;

else

r<=a;

end

endmodule

module ALU(aluc,A, B, r);

input[3:0] aluc;

input [31:0] A,B;

output reg [31:0] r;

always @(aluc, A, B) begin

case (aluc)

4'hx:r<=32'hxxxxxxxx;

0: r <= A & B;

1: r <= A | B;

2: r <= A + B;

6: r <= A - B;

7: r <= A < B ? 1 : 0;

12: r <= ~(A | B); //result is nor default: ALUOut <= 0;

15: r<=(A+B)\*(~A+~B);

endcase

end

endmodule

module MEM(signal,rst,clk,ewreg,em2reg,ewmem,rd\_in,alur,qb\_in,mwreg,mm2reg,rd\_out,alur\_out,Data\_out,endsignal);

input signal;

input rst, clk;

input ewreg,em2reg,ewmem;

input [4:0] rd\_in;

input [31:0]alur;

input[31:0] qb\_in;

output reg mwreg;

output reg mm2reg;

output reg [4:0] rd\_out;

output reg[31:0]alur\_out;

output wire[31:0]Data\_out;

output reg endsignal;

reg mwmem;

reg [31:0]address,data\_in;

always @(posedge clk or posedge rst)

begin

if(rst) begin

mwreg<=1'bx;

mm2reg<=1'bx;

mwmem<=1'bx;

rd\_out<=4'bxxxx;

alur\_out<=32'hxxxxxxxx;

address<=32'hxxxxxxxx;

data\_in<=32'hxxxxxxxx;

end

else begin

mwreg<=ewreg;

mm2reg<=em2reg;

mwmem<=ewmem;

rd\_out<=rd\_in;

alur\_out<=alur;

address<=alur;

data\_in<=qb\_in;

endsignal<=signal;

end

end

DataMem zz(clk,mwmem,address,data\_in,Data\_out);

endmodule

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

input clk;

input we;

input [31:0] a;

input [31:0] di;

output [31:0] do;

reg [31:0] RAM[0:256];

integer i;

initial begin

for(i=0;i<256;i=i+1) begin

RAM[i]=0;end

RAM[0]=32'hA00000AA;

RAM[1]=32'h10000011;

RAM[2]=32'h20000022;

RAM[3]=32'h30000033;

RAM[4]=32'h40000044;

RAM[5]=32'h50000055;

RAM[6]=32'h60000066;

RAM[7]=32'h70000077;

RAM[8]=32'h80000088;

RAM[9]=32'h90000099;

end

assign do=RAM[a];

always @(posedge clk) begin

if(we)

RAM[a]<=di;

end

endmodule

module WB(signal,rst,clk,mwreg,mm2reg,rd\_in,alu\_in,memdata\_in,endsignal);

input signal;

input rst,clk;

input mwreg,mm2reg;

input [4:0] rd\_in;

input [31:0] alu\_in,memdata\_in;

output reg endsignal;

reg wwreg,wm2reg;

reg[4:0] rd\_out;

reg[31:0] alu\_out,memdata\_out;

wire[31:0] mux\_out;

wire[31:0] data1,data2;

always @(posedge clk or posedge rst)

begin

if(rst) begin

wwreg<=1'bx;

wm2reg<=1'bx;

rd\_out<=5'bxxxxx;

alu\_out<=32'hxxxxxxxx;

memdata\_out<=32'hxxxxxxxx;

end

else begin

wwreg<=mwreg;

wm2reg<=mm2reg;

rd\_out<=rd\_in;

alu\_out<=alu\_in;

memdata\_out<=memdata\_in;

endsignal<=signal;

end

end

MUX\_32bit eee(wm2reg,alu\_out,memdata\_out,mux\_out);

registerfile fff(5'bxxxxx,5'bxxxxx,rd\_out,mux\_out,wwreg,data1,data2,clk);

endmodule

module MUX\_32bit(c,a,b,r);

input c;

input [31:0] a,b;

output reg [31:0] r;

always@ (a or b or c) begin

if(c)

r<=b;

else

r<=a;

end

endmodule

module registerfile(Read1,Read2,WriteReg,WriteData,RegWrite, Data1, Data2, clk);

input [4:0] Read1, Read2, WriteReg;

input [31:0] WriteData;

input RegWrite, clk;

output [31:0] Data1, Data2;

reg [31:0] RF[31:0];

integer i;

initial begin

RF[0]=32'hA00000AA;

RF[1]=32'h10000011;

RF[2]=32'h20000022;

RF[3]=32'h30000033;

RF[4]=32'h40000044;

RF[5]=32'h50000055;

RF[6]=32'h60000066;

RF[7]=32'h70000077;

RF[8]=32'h80000088;

RF[9]=32'h90000099;

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

RF[i]=0;

end

assign Data1=RF[Read1];

assign Data2=RF[Read2];

always @(posedge clk)

begin

if(RegWrite)

RF[WriteReg]<=WriteData;

end

endmodule

`timescale 1ns/1ps

module testBench(

);

reg clk, rst;

wire endnote;

cpu a(rst,clk,endnote);

initial begin

#0 rst=1;

#5 rst=0;

#21 rst=0;

#40 rst=0;

end

initial begin

clk=1'b1;

forever #5 clk=~clk;

end

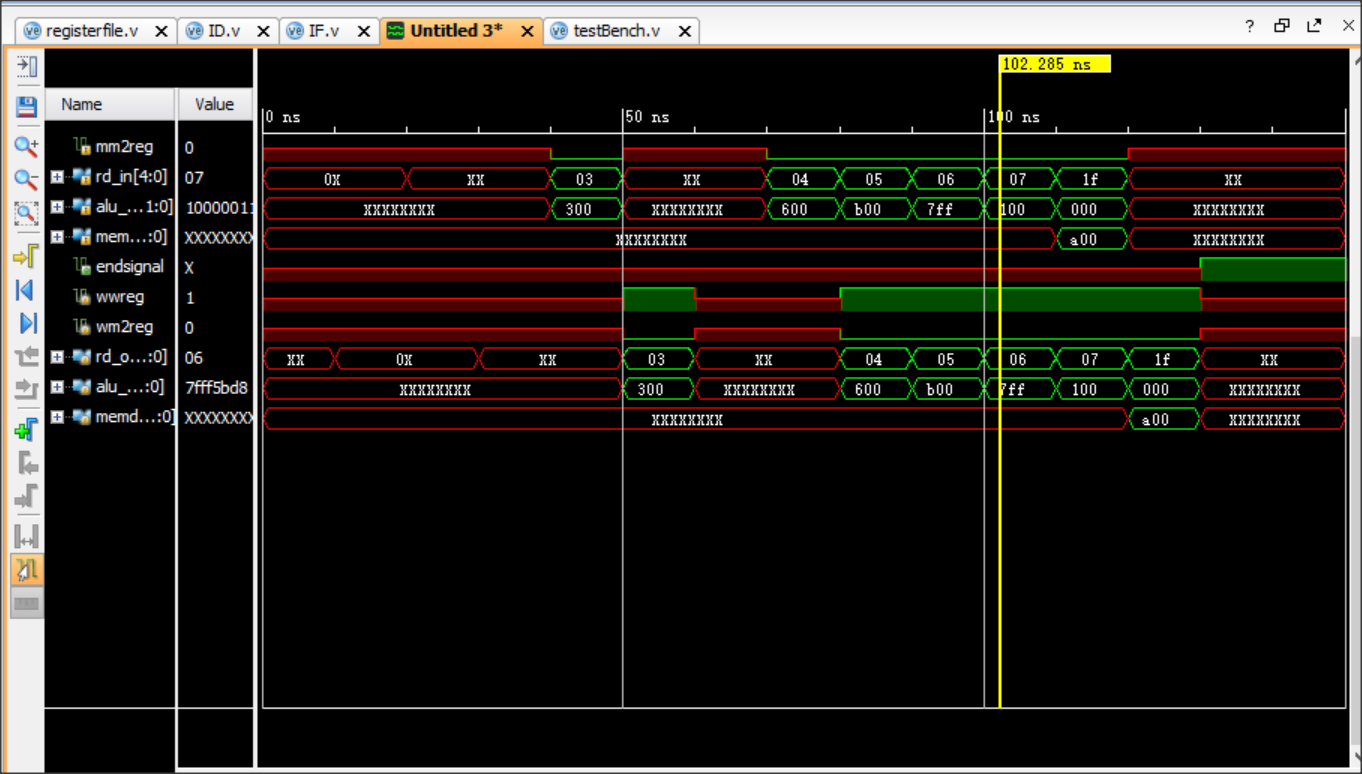
initial begin

#150 $finish;

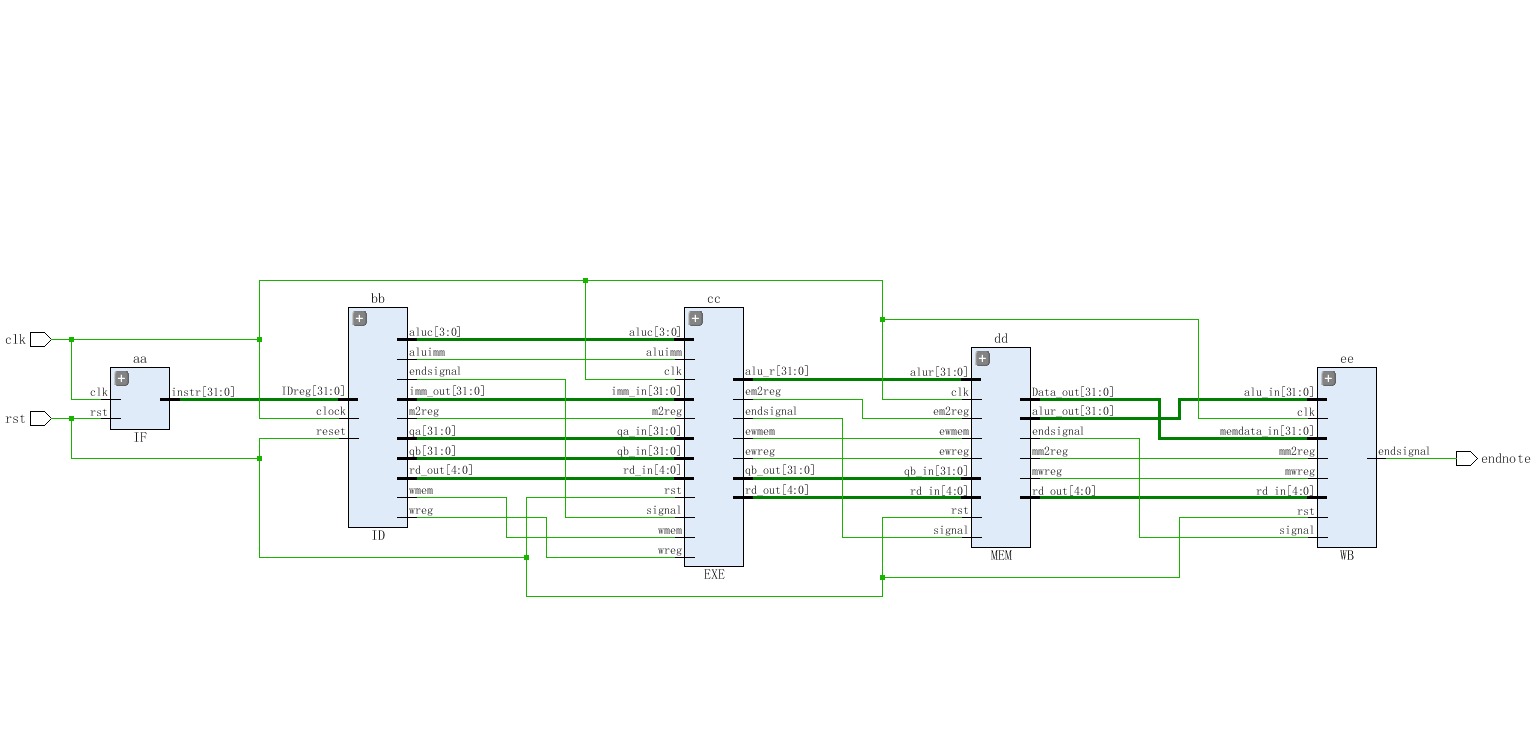
end

endmodule

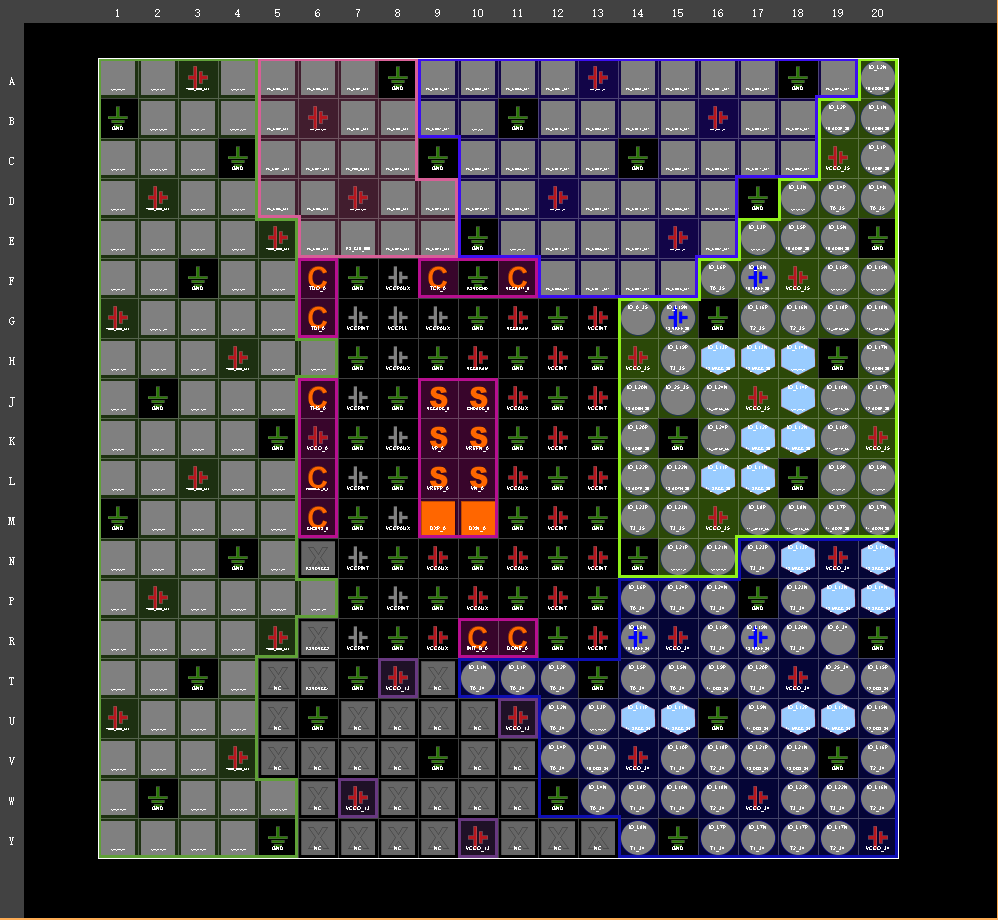
## Waveform:



## Schematic:



## I/O Planning:



## Floor planning:

## C:\Users\haocheng\Desktop\ffffloor.PNGffffloor

## ZYBO Results:

