**Penn State University**

**School of Electrical Engineering and Computer Science**

**CMPEN331 – Computer Organization and Design**

**Section 002**

**Instructor: Dr. Mohamed Almekkawy**

**Lab 5**

**Produced by Hongshuo Wang**

* Verilog Source

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 04/20/2019 05:08:48 PM

// Design Name:

// Module Name: lab5

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module PCAdder(

input [7:0] currentPC,

output reg [7:0] upDatedPC);

parameter INCREMENT\_AMOUNT = 8'd4;

initial upDatedPC = 8'd100;

always @(\*)

begin

upDatedPC <= currentPC + INCREMENT\_AMOUNT;

end

endmodule

module instMem(

input [7:0] a,

output reg[31:0] do

);

reg [31:0] IM [0:255];

parameter FIRST\_LOCATION = 8'd100;

parameter SECOND\_LOCATION = 8'd104;

parameter THIRD\_LOCATION = 8'd108;

parameter FORTH\_LOCATION = 8'd112;

parameter FIFTH\_LOCATION = 8'd116;

initial begin

IM[FIRST\_LOCATION] = 32'b10001100001000100000000000000000;

IM[SECOND\_LOCATION] = 32'b10001100001000110000000000000100;

IM[THIRD\_LOCATION] = 32'b10001100001001000000000000001000;

IM[FORTH\_LOCATION] = 32'b10001100001001010000000000001100;

IM[FIFTH\_LOCATION] = 32'b00000000010010100011000000100000;

end

always @(\*)

begin

do <= IM[a];

end

endmodule

module controlUnit(

input [31:0] do,

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

output reg [3:0] aluc

);

parameter LW = 6'b100011;

parameter R = 6'b000000;

parameter SW = 6'b101011;

parameter BEQ = 6'b000100;

parameter ADD = 6'b100000;

parameter SUB = 6'b100010;

parameter AND = 6'b100100;

parameter OR = 6'b100101;

parameter XOR = 6'b100110;

parameter SLT = 6'b101010;

wire [5:0] op, func;

assign op = do[31:26];

assign func = do[5:0];

always @(\*)

begin

case (op)

LW: begin

wreg <= 1;

m2reg <= 1;

wmem <= 0;

regrt <= 1;

aluimm <= 1;

aluc <= 4'b0010;

end

SW: begin

wreg <= 0;

wmem <= 1;

aluc <= 4'b0010;

aluimm <= 1;

end

BEQ: begin

aluimm <= 0;

wreg <= 0;

wmem <= 0;

aluc <= 4'b0110;

end

R: begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

regrt <= 1;

aluimm <= 0;

case (func)

ADD: begin

aluc <= 4'b0010;

end

SUB:begin

aluc <= 4'b0110;

end

AND: begin

aluc <= 4'b0000;

end

OR: begin

aluc <= 4'b0001;

end

SLT: begin

aluc <= 4'b0111;

end

XOR: begin

aluc <= 4'b1100;

end

endcase

end

endcase

end

endmodule

module rtrdMultiplier(

input [31:0] do,

input regrt,

output reg[4:0] outputReg

);

wire [4:0] rd,rt;

assign rd = do[15:11];

assign rt = do[20:16];

always @(\*)

begin

if (regrt == 1)

outputReg <= rt;

else

outputReg <= rd;

end

endmodule

module regfile(

input [31:0] do, d,

input [4:0] wn,

input we, clk,

output [31:0] qa, qb

);

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

integer i;

initial

begin

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

RF[i] <= 32'd0;

end

wire [4:0] rna, rnb;

assign rna = do[25:21];

assign rnb = do[20:16];

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

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

always @(posedge clk)

begin

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

RF[wn] <= d;

end

endmodule

module e(

input [31:0] do,

output reg [31:0] extendedImm

);

wire [15:0] imm;

assign imm = do[15:0];

always @(\*)

begin

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

end

endmodule

module PCReg(

input [7:0] inputPC,

input clk,

output reg[7:0] outputPC

);

always @(posedge clk)

begin

outputPC <= inputPC;

end

endmodule

module IF\_IDReg(

input [31:0] inputDo,

input clk,

output reg [31:0] outputDo

);

always @(posedge clk)

begin

outputDo <= inputDo;

end

endmodule

module ID\_EXEReg(

input inputWreg, inputM2reg, inputWmem, inputAluimm, clk,

input [3:0] inputAluc,

input [4:0] inputReg,

input [31:0] inputQa, inputQb, inputImm,

output reg outputWreg, outputM2reg, outputWmem, outputAluimm,

output reg [31:0] outputQa, outputQb, outputImm,

output reg [3:0] outputAluc,

output reg [4:0] outputReg

);

always @(posedge clk)

begin

outputWreg <= inputWreg;

outputM2reg <= inputM2reg;

outputWmem <= inputWmem;

outputAluimm <= inputAluimm;

outputQa <= inputQa;

outputQb <= inputQb;

outputImm <= inputImm;

outputAluc <= inputAluc;

outputReg <= inputReg;

end

endmodule

module EaluimmMultiplexer(

input ealuimm,

input [31:0] qb, imm,

output reg [31:0] b

);

always @(\*)

begin

if (ealuimm == 0)

b <= qb;

else

b <= imm;

end

endmodule

module ALU(

input [3:0] aluc,

input [31:0] a, b,

output reg [31:0] r

);

parameter ADD = 4'b0010;

parameter SUB = 4'b0110;

parameter AND = 4'b0000;

parameter OR = 4'b0001;

parameter SLT = 4'b0111;

parameter XOR = 4'b1100;

always @(\*)

begin

case (aluc)

ADD: begin

r <= a + b;

end

SUB: begin

r <= a - b;

end

AND: begin

r <= a & b;

end

OR: begin

r <= a | b;

end

XOR: begin

r <= ~(a | b);

end

SLT: begin

r <= a < b ? 1 : 0;

end

endcase

end

endmodule

module EXE\_MEM(

input inputEwreg, inputEm2reg, inputEwmem, clk,

input [4:0] inputEreg,

input [31:0] inputR, inputEqb,

output reg outputEwreg, outputEm2reg, outputEwmem,

output reg [4:0] outputEreg,

output reg [31:0] outputR, outputEqb

);

always @(posedge clk)

begin

outputEwreg <= inputEwreg;

outputEm2reg <= inputEm2reg;

outputEwmem <= inputEwmem;

outputEreg <= inputEreg;

outputR <= inputR;

outputEqb <= inputEqb;

end

endmodule

module Datamem(

input [31:0] a, di,

input we,

output reg [31:0] do

);

reg [31:0] mem [0:1023];

initial begin

$readmemh("lab5.mem", mem);

end

always @(\*)

begin

if (we == 1)

begin

mem[a >> 2] <= di;

end

else

begin

do <= mem[a >> 2];

end

end

endmodule

module MEM\_WB(

input inputMwreg, inputMm2reg, clk,

input [4:0] inputMReg,

input [31:0] inputMr, inputMdo,

output reg outputMwreg, outputMm2reg,

output reg [4:0] outputMReg,

output reg [31:0] outputMr, outputMdo

);

always @(posedge clk)

begin

outputMwreg <= inputMwreg;

outputMm2reg <= inputMm2reg;

outputMr <= inputMr;

outputMReg <= inputMReg;

outputMdo <= inputMdo;

end

endmodule

module wm2regMultiplexer(

input [31:0] winputMem, winputALU,

input wm2reg,

output reg [31:0] wOutput

);

always @(\*)

begin

if (wm2reg == 1)

wOutput <= winputMem;

else

wOutput <= winputALU;

end

endmodule

* TestBench

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 04/20/2019 05:11:46 PM

// Design Name:

// Module Name: TestBench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module testBench();

wire [7:0] inputPC\_tb;

wire [31:0] inputDo\_tb, outputDo\_tb, inputQa\_tb, inputQb\_tb, eqa\_tb, eqb\_tb, inputImm\_tb, eimm\_tb;

wire inputWreg\_tb, inputM2reg\_tb, inputWmem\_tb, inputAluimm\_tb, regrt\_tb;

wire ealuimm\_tb;

wire [3:0] inputAluc\_tb,ealuc\_tb;

wire [4:0] inputReg\_tb;

wire [7:0] outputPC\_tb;

reg clk\_tb;

wire ewreg\_tb, em2reg\_tb, ewmem\_tb;

wire mwreg\_tb, mm2reg\_tb, mwmem\_tb;

wire wwreg\_tb, wm2reg\_tb;

wire [4:0] eReg\_tb, mReg\_tb, wReg\_tb;

wire [31:0] er\_tb, mr\_tb, mdi\_tb, wr\_tb, wdo\_tb, mdo\_tb, eb\_tb, w2regoutput\_tb;

PCAdder PCAdder\_tb(.currentPC(outputPC\_tb), .upDatedPC(inputPC\_tb));

PCReg PCReg\_tb(.inputPC(inputPC\_tb), .clk(clk\_tb), .outputPC(outputPC\_tb));

instMem instMem\_tb(.a(outputPC\_tb), .do(inputDo\_tb));

IF\_IDReg IF\_IDReg\_tb(.inputDo(inputDo\_tb), .clk(clk\_tb), .outputDo(outputDo\_tb));

controlUnit controlUnit\_tb(.do(outputDo\_tb), .regrt(regrt\_tb), .wreg(inputWreg\_tb), .m2reg(inputM2reg\_tb), .wmem(inputWmem\_tb), .aluimm(inputAluimm\_tb), .aluc(inputAluc\_tb));

rtrdMultiplier rtrdMultiplier\_tb(.do(outputDo\_tb), .regrt(regrt\_tb), .outputReg(inputReg\_tb));

ID\_EXEReg ID\_EXEReg\_tb

(

.inputWreg(inputWreg\_tb),

.inputM2reg(inputM2reg\_tb),

.inputWmem(inputWmem\_tb),

.inputAluimm(inputAluimm\_tb),

.clk(clk\_tb),

.inputAluc(inputAluc\_tb),

.inputReg(inputReg\_tb),

.inputQa(inputQa\_tb),

.inputQb(inputQb\_tb),

.inputImm(inputImm\_tb),

.outputWreg(ewreg\_tb),

.outputM2reg(em2reg\_tb),

.outputWmem(ewmem\_tb),

.outputAluimm(ealuimm\_tb),

.outputQa(eqa\_tb),

.outputQb(eqb\_tb),

.outputImm(eimm\_tb),

.outputAluc(ealuc\_tb),

.outputReg(eReg\_tb)

);

regfile regfile\_tb(.do(outputDo\_tb), .d(w2regoutput\_tb), .wn(wReg\_tb), .we(wwreg\_tb), .clk(clk\_tb), .qa(inputQa\_tb), .qb(inputQb\_tb));

e e\_tb(.do(outputDo\_tb), .extendedImm(inputImm\_tb));

wm2regMultiplexer wm2regMultiplexer\_tb(wdo\_tb, wr\_tb, wm2reg\_tb, w2regoutput\_tb);

EXE\_MEM EXE\_MEM\_tb

(ewreg\_tb, em2reg\_tb, ewmem\_tb, clk\_tb, eReg\_tb, er\_tb, eqb\_tb,

mwreg\_tb, mm2reg\_tb, mwmem\_tb, mReg\_tb, mr\_tb, mdi\_tb);

MEM\_WB MEM\_WB\_tb

(mwreg\_tb, mm2reg\_tb, clk\_tb, mReg\_tb, mr\_tb, mdo\_tb, wwreg\_tb, wm2reg\_tb, wReg\_tb, wr\_tb, wdo\_tb);

EaluimmMultiplexer ealuimmMultiplexer\_tb

(ealuimm\_tb, eqb\_tb, eimm\_tb, eb\_tb);

ALU ALU\_tb

(ealuc\_tb, eqa\_tb, eb\_tb, er\_tb);

Datamem Datamem\_tb

(mr\_tb, mdi\_tb, mwmem\_tb, mdo\_tb);

initial begin

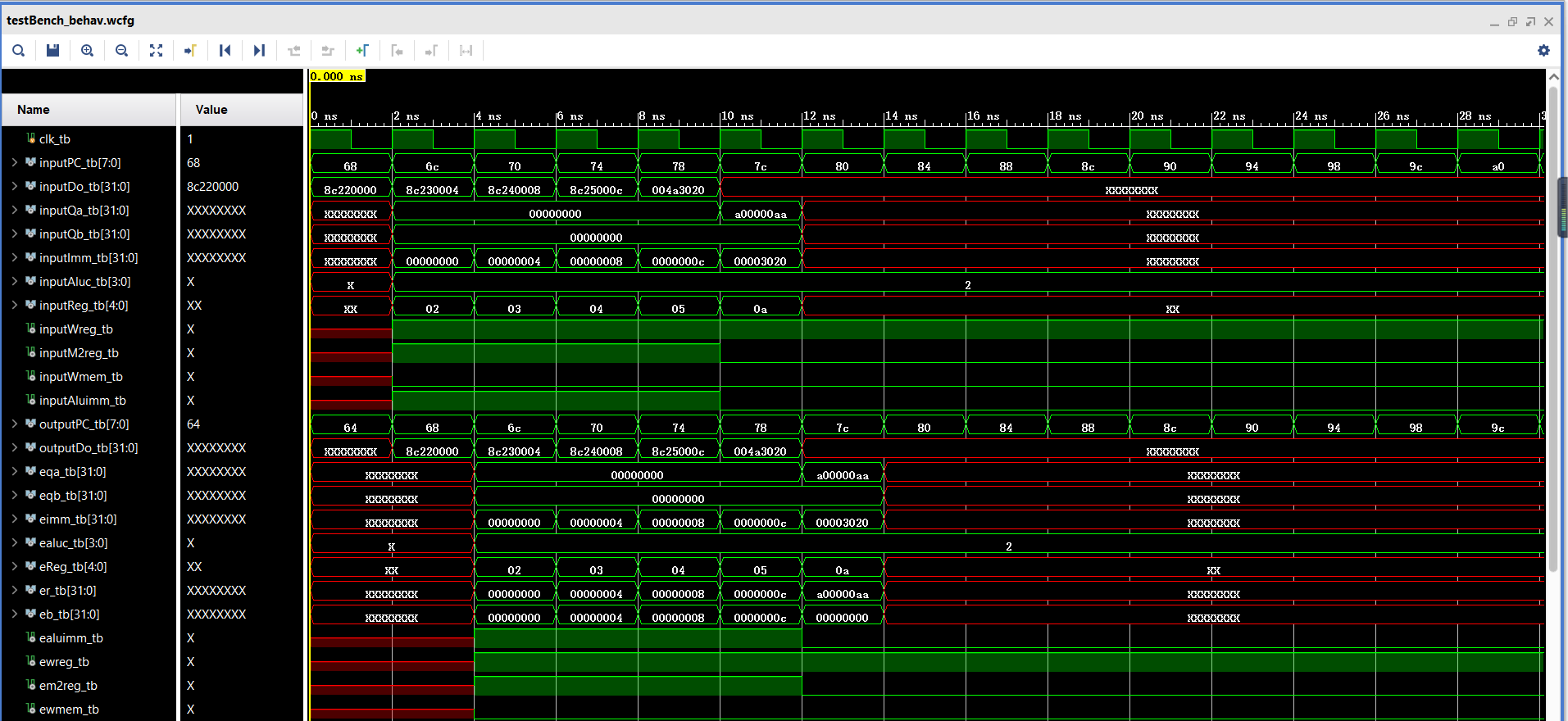
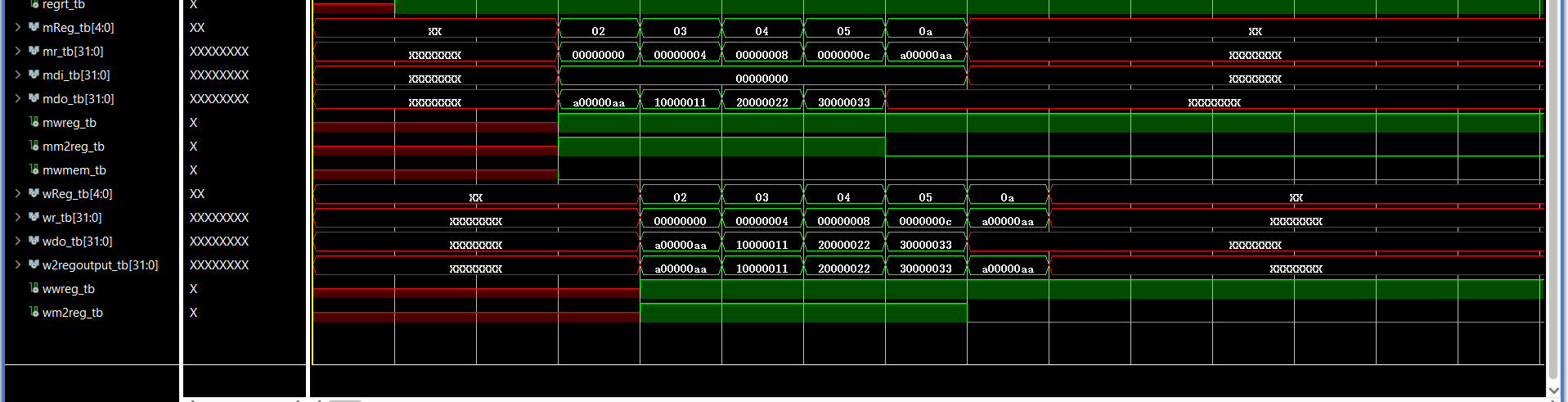
clk\_tb <= 1;

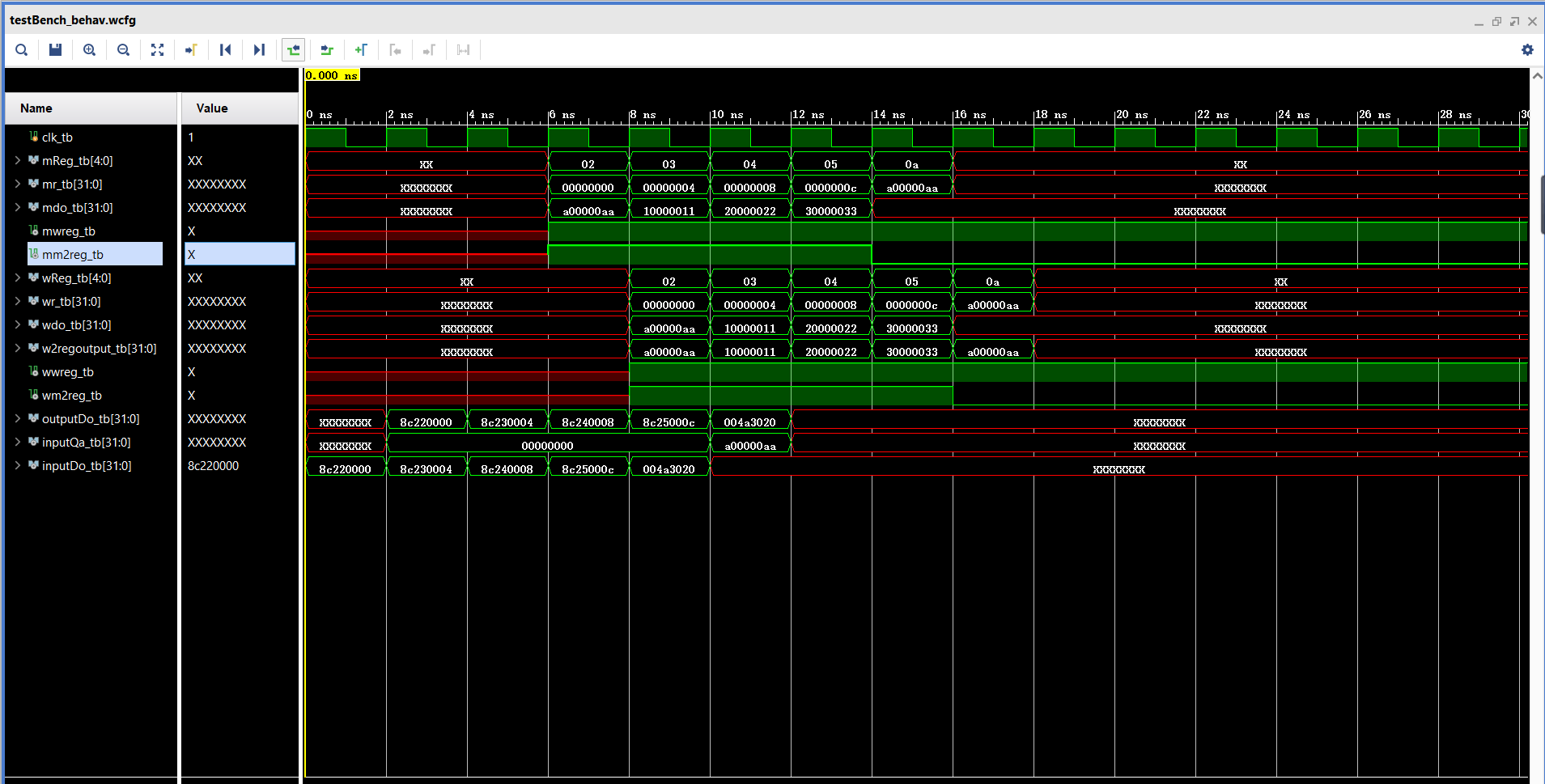
end

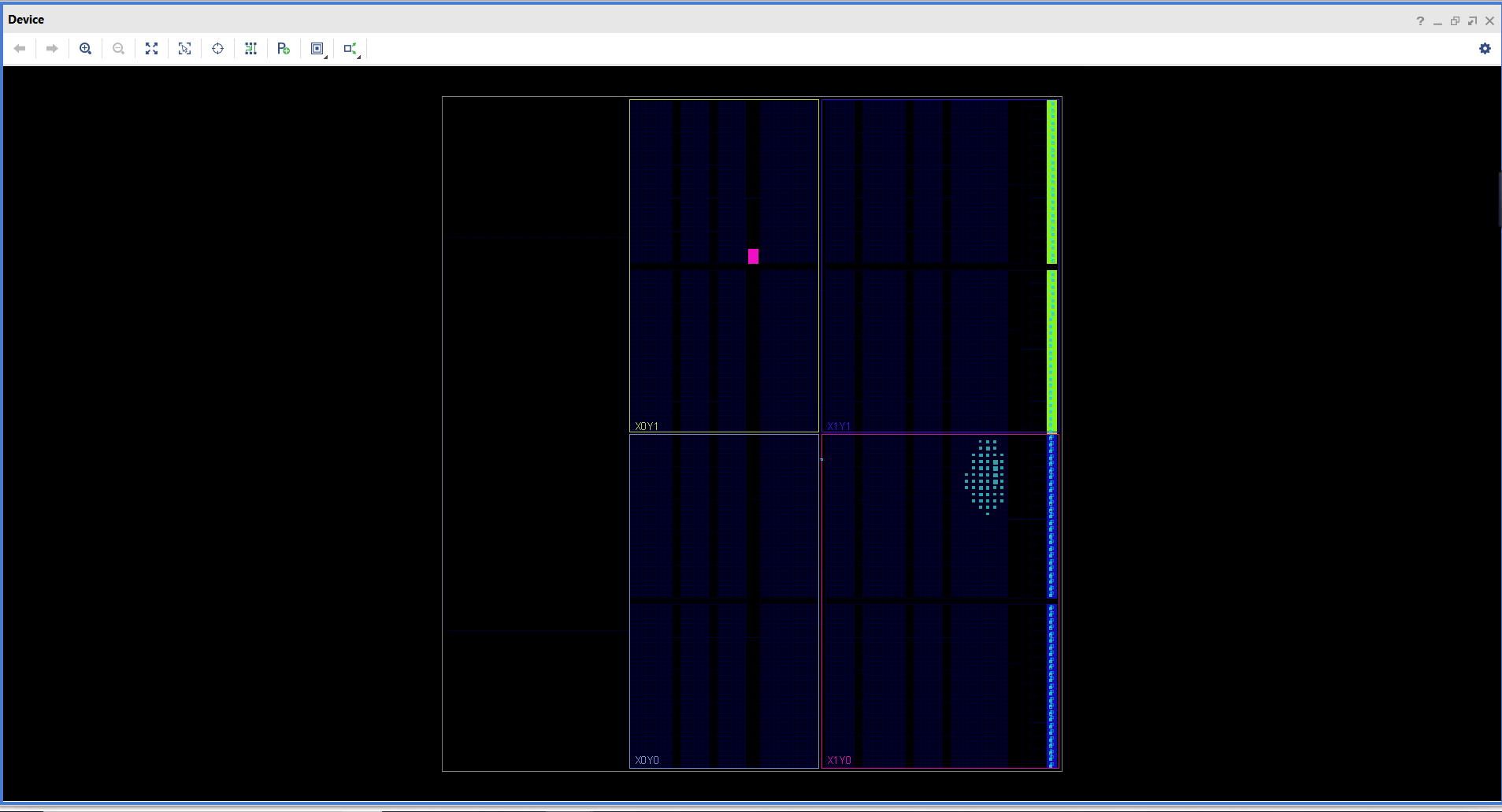
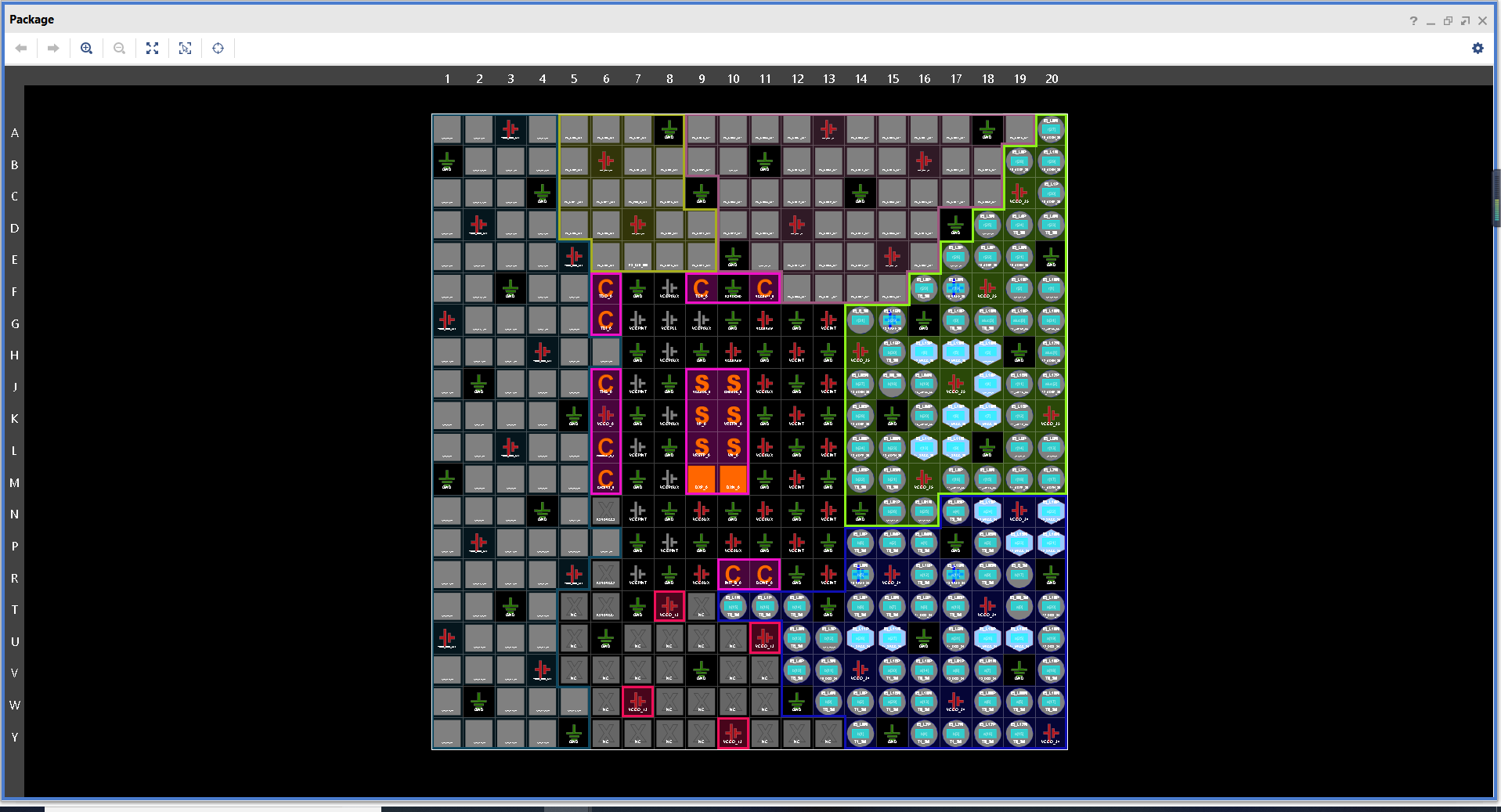
always begin

#1 clk\_tb <= ~clk\_tb;

End

* Waveform
* Whole Form



* + Simple form
* Floor planning
* I/O planning
* Schematics

