CMPEN 331

Lab 5 Report

Mrigank Doshy

Access ID: msd5399

The Pennsylvania State University

1. programCounterAdder

```
`timescale 1ns / 1ps
module programCounterAdder(
    input [31:0] nextPC,
    output reg [31:0] PC
    );

initial begin
    PC=32'd100;
end

always @(*)
    begin
        PC<=nextPC+4;
    end
endmodule
```

2. instructionMemory

```
`timescale 1ns / 1ps
module instructionMemory(
    input [31:0] PC,
    output reg [31:0] instruction
    );

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

initial begin
    instMemory[100] = 32'b10001100001000100000000000000000;
    instMemory[104] = 32'b10001100001000110000000000000100;
    instMemory[108] = 32'b10001100001001000000000000001000;
    instMemory[112] = 32'b10001100001001010000000000001100;
    instMemory[116] = 32'b00000000010010100011000000100000;
end
always @(PC)
    begin
        instruction<=instMemory[PC];
    end
endmodule
```

3. controlUnit

```
`timescale 1ns / 1ps

module controlUnit(
    input [5:0]op,
    input [5:0]func,
    output reg wreg,
    output reg m2reg,
    output reg wmem,
    output reg [3:0]aluc,
    output reg aluimm,
    output reg regrt
    );


always @(op, func)
    begin
        case(op)
            6'b100011: //lw
                begin
                    wreg<=1'b1;
                    m2reg<=1'b1;
                    wmem<=1'b0;
                    aluc<=4'b0010;
                    aluimm<=1'b1;
```

```
        regrt<=1'b1;
    end
6'b101011: //sw
    begin
        wreg<=1'b0;
        m2reg<=1'b0;
        wmem<=1'b1;
        aluc<=4'b0010;
        aluimm<=1'b1;
        regrt<=1'bx;
    end
6'b101011: //beq
    begin
        wreg<=1'b0;
        m2reg<=1'bx;
        wmem<=1'b0;
        aluc<=4'b0110;
        aluimm<=1'b0;
        regrt<=1'bx;
    end
6'b000000: //r
    begin
        case(func)
            6'b100000:
```

```
                begin
                    wreg<=1'b1;
                    m2reg<=1'b0;
                    wmem<=1'b0;
                    aluc<=4'b0010;
                    aluimm<=1'b0;
                    regrt<=1'b0;
                end
            endcase
        end
    endcase
end
endmodule
```

4. mux1

```
`timescale 1ns / 1ps
module mux1(
    input [0:4] rd,
    input [0:4] rt,
    input regrt,
    output reg [4:0] muxRegister
    );

    always@ (*)
        begin
            case(regrt)
                1'b0: muxRegister<=rd;
                1'b1: muxRegister<=rt;
            endcase
        end
endmodule
```

5. signExtension

```
`timescale 1ns / 1ps
module signExtension(
    input [31:0] imm,
    output [31:0] extendedImm
    );

    assign extendedImm = {{16{imm[15]}},imm[15:0]};
endmodule
```

6. registerFile

```
`timescale 1ns / 1ps
module registerFile(
    input clk,
    input [4:0] rna,
    input [4:0] rnb,
    input [4:0] wn,
    input [31:0] d,
    input we,
    output reg [31:0] qa, qb
    );
    reg [31:0] regFile [0:31];
    integer index;
    initial begin
        for (index=0; index<32; index=index+1)
            regFile[index]<=1'b0;
    end
    always@ (posedge clk, rna, rnb)
        begin
            qa<=regFile[rna];
            qb<=regFile[rnb];
        end
    always @(negedge clk, rna, rnb)
        begin
             case(we)
               1'b1: regFile[wn]<=d;
             endcase
        end
endmodule
```

7. programCounter

```
`timescale 1ns / 1ps
module programCounter(
    input clk,
    input [31:0] nextPC,
    output reg [31:0] PC
    );

    always@(posedge clk)
        begin
            PC<=nextPC;
        end
endmodule
```

8. instructionFetch

```
`timescale 1ns / 1ps
module instructionFetch(
    input clk,
    input [31:0] instruction,
    output reg [31:0] nextInstruction
    );

    always@(posedge clk)
            begin
                nextInstruction<=instruction;
            end
endmodule
```

9. instructionDecode

```
`timescale 1ns / 1ps

module instructionDecode(
    input clk,
    input wreg,
    input m2reg,
    input wmem,
    input [3:0] aluc,
    input aluimm,
    input [4:0] muxRegister,
    input [31:0] qa,
    input [31:0] qb,
    input [31:0] extendedImm,
    output reg ewreg,
    output reg em2reg,
    output reg ewmem,
    output reg [3:0] ealuc,
    output reg ealuimm,
    output reg [4:0] eMuxRegister,
    output reg [31:0] eqa,
    output reg [31:0] eqb,
    output reg [31:0] eExtendedImm
```

```
    );


   always @(posedge clk)
       begin
          ewreg<=wreg;
          em2reg<=m2reg;
          ewmem<=wmem;
          ealuc<=aluc;
          ealuimm<=aluimm;
          eMuxRegister<=muxRegister;
          eqa<=qa;
          eqb<=qb;
          eExtendedImm<=extendedImm;
       end
endmodule
```

10. Mux2

```
`timescale 1ns / 1ps
module mux2(
    input [31:0]eqb,
    input [31:0]eExtendedImm,
    input ealuimm,
    output reg [31:0] muxRegister2
    );

    always@ (*)
        begin
            case(ealuimm)
                1'b0: muxRegister2<=eqb;
                1'b1: muxRegister2<=eExtendedImm;
            endcase
        end
endmodule
```

11. ALU

```
`timescale 1ns / 1ps
module ALU(
    input [31:00]a, //eqa
    input [31:00]b, //output of mux2 - muxRegister2
    input [3:0]aluc, //ealuc
    output reg [31:00]r
    );
    always @ (*)
        begin
            case (aluc)
                4'b0010 : r = a + b;
            endcase
        end
endmodule
```

12. dataMemory

```
`timescale 1ns / 1ps
module dataMemory(
    input [31:0] a, //r after instructionExecute -rexe
    input [31:0] di, //eqb after instructionExecute - eeqb
    input we, //mwmem - ewmem after instructionExecute
    output reg [31:0]do
    );

    reg [31:0] dataMemory [0:511];
    initial begin
        dataMemory[0]<=32'hA00000AA;
        dataMemory[4]<=32'h10000011;
        dataMemory[8]<=32'h20000022;
        dataMemory[12]<=32'h30000033;
        dataMemory[16]<=32'h40000044;
        dataMemory[20]<=32'h50000055;
        dataMemory[24]<=32'h60000066;
        dataMemory[28]<=32'h70000077;
        dataMemory[32]<=32'h80000088;
        dataMemory[36]<=32'h90000099;
```

```
    end


    always@(a, di)
        begin
            do<=dataMemory[a];
        end
endmodule
```

13. Memory

```
module Memory(
    input clk,
    input mwreg,
    input mm2reg,
    input [4:0]emMuxRegister,
    input [31:0]mr,
    input [31:0]do,
    output reg wwreg,
    output reg wm2reg,
    output reg [4:0]emwMuxRegister,
    output reg [31:0] wmr,
    output reg [31:0] wdo
    );
    always@ (posedge clk)
       begin
         wwreg<=mwreg;
         wm2reg<=mm2reg;
         emwMuxRegister<=emMuxRegister;
         wmr<=mr;
         wdo<=do;
       end
  endmodule
```

14. instructionExecute

```
module instructionExecute(
    input clk,
    input ewreg,
    input em2reg,
    input ewmem,
    input [4:0] eMuxRegister,
    input [31:0]r,
    input [31:0]eqb,
    output reg mwreg,
    output reg mm2reg,
    output reg mwmem,
    output reg [4:0]emMuxRegister,
    output reg [31:0]mr, //r after EXE/MEM,
    output reg [31:0]emqb
);
    always@(posedge clk)
        begin
            mwreg<=ewreg;
            mm2reg<=em2reg;
            mwmem<=ewmem;
            emMuxRegister<=eMuxRegister;
            mr<=r;
            emqb<=eqb;
        end
endmodule
```

15. mux3

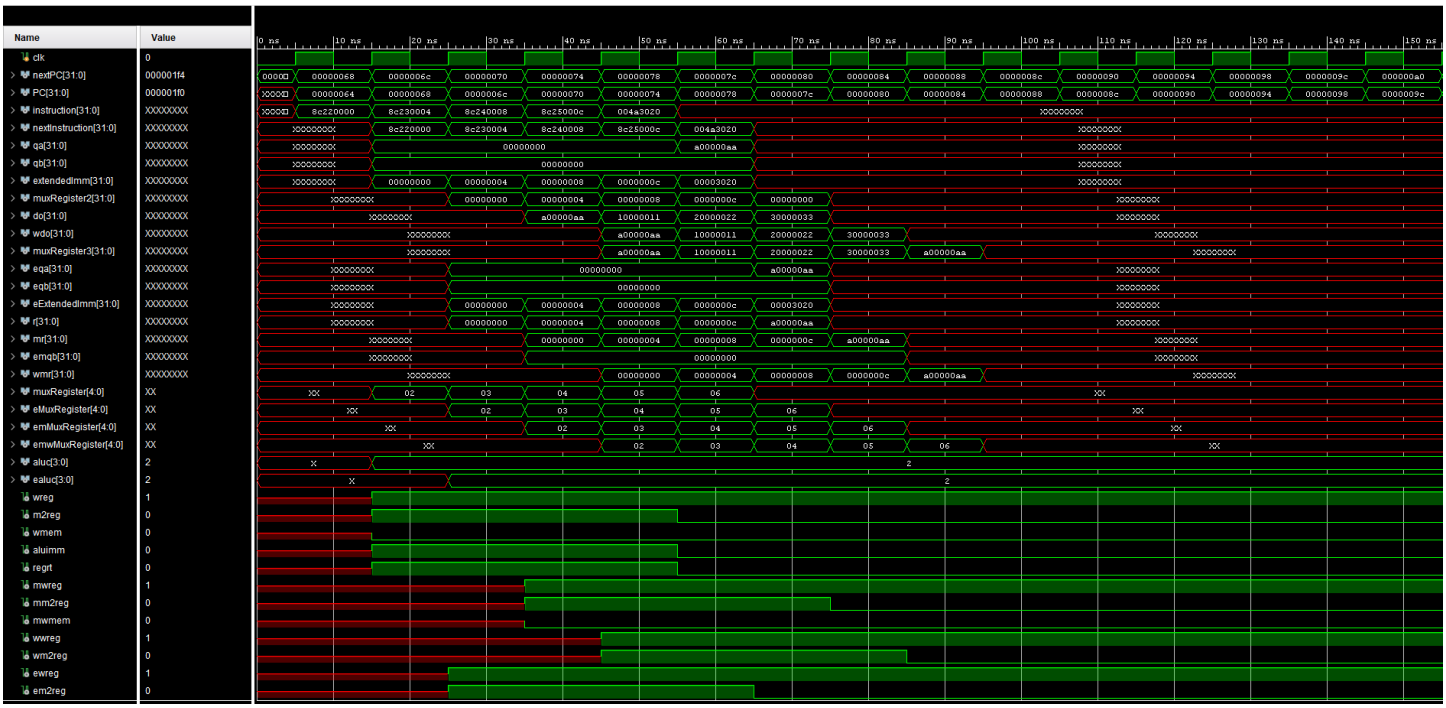
```
module mux3(
    input [31:0]wmr,
    input [31:0]wdo,
    input wm2reg,
    output reg [31:0] muxRegister3
    );

    always@ (*)
        begin
            case(wm2reg)
                1'b0: muxRegister3<=wmr;
                1'b1: muxRegister3<=wdo;
            endcase
        end
endmodule
```

16. testbench

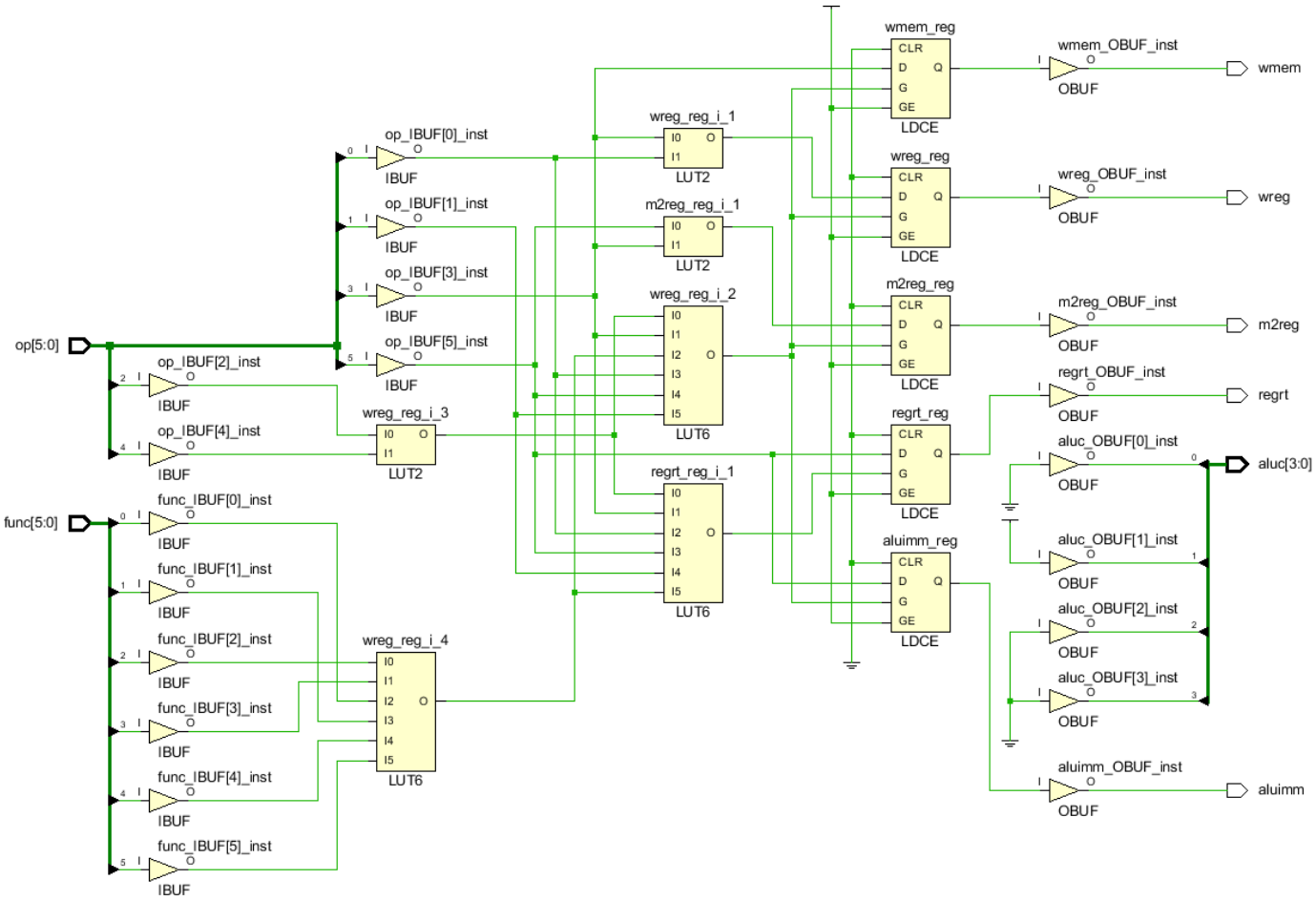
```
`timescale 1ns / 1ps

module testbench();

    reg clk;

    wire [31:0] nextPC, PC;

    wire [31:0] instruction, nextInstruction, qa, qb, extendedImm, muxRegister2, do, wdo, muxRegister3;

    wire [31:0] eqa, eqb, eExtendedImm, r, mr, emqb, wmr;

    wire [4:0] muxRegister, eMuxRegister, emMuxRegister, emwMuxRegister;

    wire [3:0] aluc, ealuc;

    wire wreg, m2reg, wmem, aluimm, regrt, mwreg, mm2reg, mwmem, wwreg, wm2reg, ewreg, em2reg, ewmem, ealuimm;


    initial begin

        clk=0;

    end

    programCounter programCounter_tb(clk, nextPC, PC);

    programCounterAdder programCounterAdder_tb(PC, nextPC);

    instructionMemory instructionMemory_tb(PC, instruction);

    instructionFetch instructionFetch_tb(clk, instruction, nextInstruction);

    controlUnit controlUnit_tb(nextInstruction[31:26], nextInstruction[5:0], wreg, m2reg, wmem, aluc, aluimm, regrt);

    mux1 mux1_tb(nextInstruction[15:11], nextInstruction[20:16], regrt, muxRegister);
```

```
    registerFile registerFile_tb(clk, nextInstruction[25:21], nextInstruction[20:16], emwMuxRegister, muxRegister3, wwreg, qa, qb);

    signExtension signExtension_tb(nextInstruction, extendedImm);

    instructionDecode instructionDecode_tb(clk, wreg, m2reg, wmem, aluc, aluimm, muxRegister, qa, qb, extendedImm, ewreg, em2reg, ewmem, ealuc, ealuimm, eMuxRegister, eqa, eqb, eExtendedImm);

    mux2 mux2_tb(eqb, eExtendedImm, ealuimm, muxRegister2);

    ALU ALU_tb(eqa,muxRegister2,ealuc,r);

    instructionExecute instructionExecute_tb( clk, ewreg, em2reg, ewmem, eMuxRegister, r, eqb, mwreg, mm2reg, mwmem, emMuxRegister, mr, emqb);

    dataMemory dataMemory_tb(mr, emqb, mwmem, do);

    Memory Memory_tb(clk, mwreg, mm2reg, emMuxRegister, mr, do, wwreg, wm2reg, emwMuxRegister, wmr, wdo);

    mux3 mux3_tb(wmr, wdo, wm2reg, muxRegister3);


    always
        begin
            #5
            clk = ~clk;
        end
endmodule
```

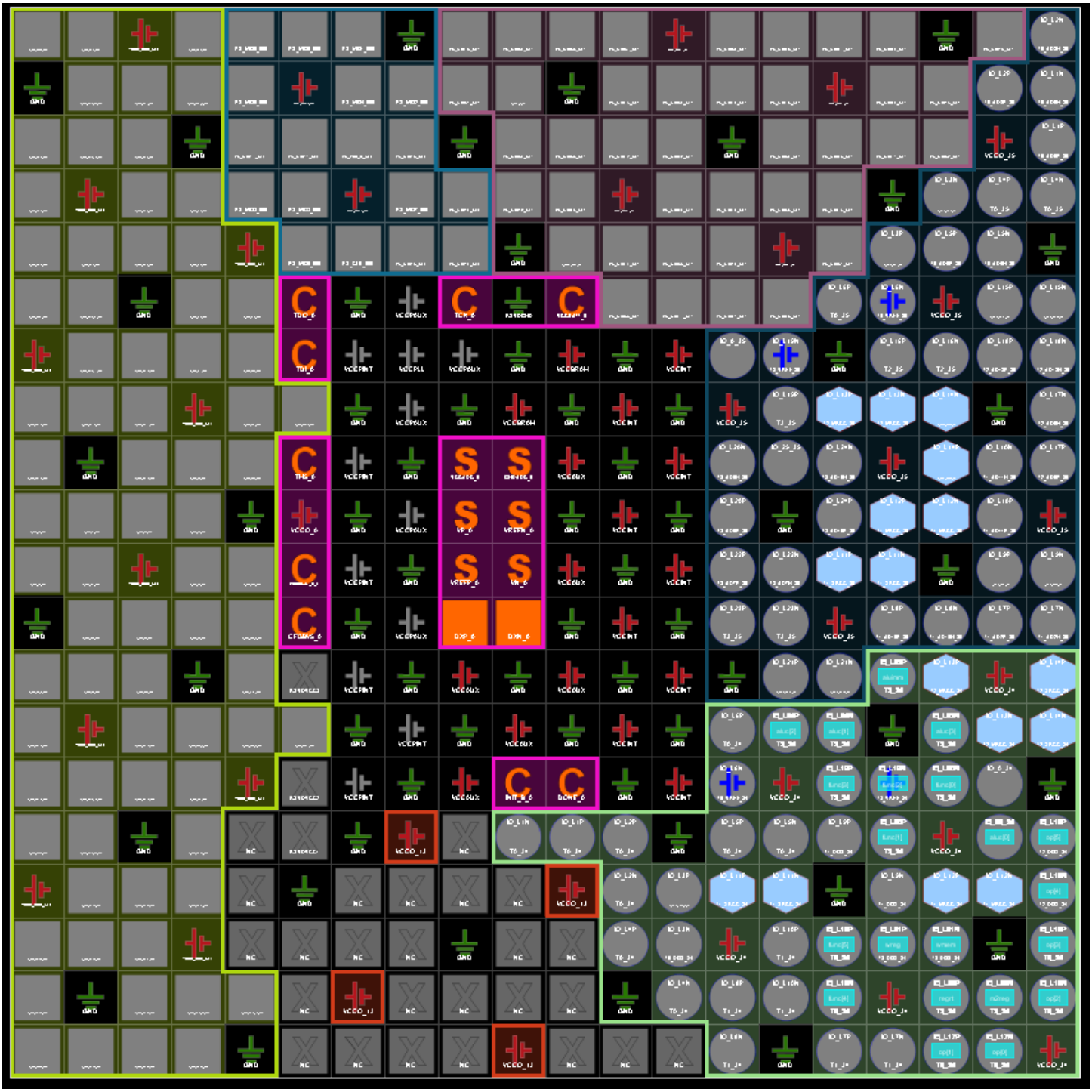
17. Waveform


Name
Value
clk
nextPC[31:0]
PC[31:0]
instruction[31:0]
nextInstruction[31:0]
qa[31:0]
qb[31:0]
extendedImm[31:0]
muxRegister2[31:0]
do[31:0]
wdo[31:0]
muxRegister3[31:0]
eqa[31:0]
eqb[31:0]
eExtendedImm[31:0]
r[31:0]
mr[31:0]
emqb[31:0]
wmr[31:0]
muxRegister[4:0]
eMuxRegister[4:0]
emMuxRegister[4:0]
emwMuxRegister[4:0]
aluc[3:0]
ealuc[3:0]
wreg
m2reg
wmem
aluimm
regrt
mwreg
mm2reg
mwmem
wwreg
wm2reg
ewreg
em2reg

18. Design Schematic



19. I/O Planning



20. Floor Planning