Organization of Digital Computer

Lab EECS 112L

LAB 2 - Pipelined Processor

EECS Department
Henry Samueli School of Engineering

University of California, Irvine

Winter, 2018

NIKET JADHAV

23142260

1. Implementation

The given instruction memory was used to test the different instructions of the processor. We can focus at each new module that was added to see how exactly the processor has been implemented.

HAZARD DETECTOR

```
`timescale 1ns / 1ps

module HazardDetector(
        input logic [2:0] MemRead_IDEX,
        input logic [4:0] RD_IDEX, RS1_IFID, RS2_IFID,
        output logic PCWrite, IFIDWrite, HazardSelect
        );

    always_comb begin
        if ((MemRead_IDEX != 3'b101) && ((RD_IDEX == RS1_IFID) || (RD_IDEX == RS2_IFID))) begin
            PCWrite <= 1;
            IFIDWrite <= 1;
            HazardSelect <= 1;
        end
        else begin
            PCWrite <= 0;
            IFIDWrite <= 0;
            HazardSelect <= 0;
        end
    end
endmodule
```

```
1  `timescale 1ns / 1ps
2  module hazard_helper(
3          input logic HazardSel, ALUSrc, MemtoReg, RegWrite,
4          input logic [1:0] MemWrite, ALUOp,
5          input logic [2:0] MemRead,
6
7          output logic ALUSrc_Hazard, MemtoReg_Hazard, RegWrite_Hazard,
8          output logic [1:0] MemWrite_Hazard, ALUOp_Hazard,
9          output logic [2:0] MemRead_Hazard
10         );
11
12     assign ALUSrc_Hazard = HazardSel ? 0 : ALUSrc;
13     assign MemtoReg_Hazard = HazardSel ? 0 : MemtoReg;
14     assign RegWrite_Hazard = HazardSel ? 0 : RegWrite;
15     assign MemWrite_Hazard = HazardSel ? 2'b11 : MemWrite;
16     assign ALUOp_Hazard = HazardSel ? 0 : ALUOp;
17     assign MemRead_Hazard = HazardSel ? 3'b101 : MemRead;
18 endmodule
```

The hazard helper module was designed to assist the hazard detector. These two modules effectively worked together to detect data hazards within the processor.

Forwarding Unit

To tackle the problem of data dependency, we had to implement a forwarding unit which forwarded data between the various stages of the processor. This unit worked along with the forwardA and forwardB unit to provide the correct sources of data in the SrcA and SrcB ports of the ALU.

```
`timescale 1ns / 1ps

module ForwardingUnit(
        input logic [4:0] RS1_IDEX, RS2_IDEX, RD_EXMEM, RD_MEMWB,
        input logic RegWrite_EXMEM, RegWrite_MEMWB,
        output logic [1:0] ForwardA, ForwardB
        );

    always_comb begin
        // ForwardA
        if (RegWrite_EXMEM && (RD_EXMEM != 0) && (RD_EXMEM == RS1_IDEX))      // EX Hazard
            ForwardA <= 2'b10;
        else if (RegWrite_MEMWB && (RD_MEMWB != 0) && (RD_MEMWB == RS1_IDEX))     // MEM Hazard
            ForwardA <= 2'b01;
        else
            ForwardA <= 2'b00;

        // ForwardB
        if (RegWrite_EXMEM && (RD_EXMEM != 0) && (RD_EXMEM == RS2_IDEX))      // EX Hazard

            ForwardB <= 2'b10;
        else if (RegWrite_MEMWB && (RD_MEMWB != 0) && (RD_MEMWB == RS2_IDEX))     // MEM Hazard
            ForwardB <= 2'b01;
        else
            ForwardB <= 2'b00;
    end
endmodule
```

```
`timescale 1ns / 1ps
module forwardB(
        input logic [1:0] ForwardB,
        input logic [31:0] Reg2_IDEX, ALUResult_EXMEM, WB_Data,
        output logic [31:0] SrcB
        );

    always_comb
    begin
        case (ForwardB)
            2'b00:
                SrcB = Reg2_IDEX;
            2'b10:
                SrcB = ALUResult_EXMEM;
            2'b01:
                SrcB = WB_Data;
            default:
                SrcB = Reg2_IDEX;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
module forwardA(
        input logic [1:0] ForwardA,
        input logic [31:0] Reg1_IDEX, ALUResult_EXMEM, WB_Data,
        output logic [31:0] SrcA
        );

    always_comb
    begin
        case (ForwardA)
            2'b00:
                SrcA = Reg1_IDEX;
            2'b10:
                SrcA = ALUResult_EXMEM;
            2'b01:
                SrcA = WB_Data;
            default:
                SrcA = Reg1_IDEX;
        endcase
    end
endmodule
```

REGISTER IMPLEMENTATION

There were a total of 4 registers that were added to the the implimentation of the processor to enable pipelining. They facilitated as data holders between the different pipelined cycles of the processor.

1. IFID Register

```
`timescale 1ns / 1ps
//import my_112l_pkg::*;
module RegIFID #(
        parameter PC_W = 9, // Program Counter
        parameter INS_W = 32 // Instruction Width
        )(
        input logic hazard_detected, clk,
        input logic [PC_W-1:0] PC, PCPlus4,
        input logic [INS_W-1:0] inst_code,

        output logic [PC_W-1:0] NextPC, NextPCPlus4,
        output logic [INS_W-1:0] inst_next
        );

    always@(posedge clk)
    begin
        if (hazard_detected == 0) begin
            NextPC <= PC;
            NextPCPlus4 <= PCPlus4;
            inst_next<=inst_code;
        end
    end
endmodule
```

Used for instruction fetching as decoding. It basically kept a track of the PC.

2. IDEX Register

```
`timescale 1ns / 1ps
module RegIDEX #(
        parameter PC_W = 9, // Program Counter
        parameter INS_W = 32, // Instruction Width
        parameter RF_ADDRESS_W = 5, // Register File Address
        parameter DATA_W = 32 // Data WriteData
        )(
        input logic clk,
        input logic ALUSrc, MemtoReg, RegWrite,
        input logic [1:0] MemWrite, ALUOp,
        input logic [2:0] MemRead,
        input logic [PC_W-1:0] IFID_PC, IFID_PCPlus4,
        input logic [DATA_W-1:0] Reg1, Reg2, ExtImm,
        input logic [2:0] Funct3_IFID,
        input logic [RF_ADDRESS_W-1:0] RD_IFID, RS1_IFID, RS2_IFID,
        input logic [6:0] opcode, Funct7,

        output logic ALUSrc_Out, MemtoReg_Out, RegWrite_Out,
        output logic [1:0] MemWrite_Out, ALUOp_Out,
        output logic [2:0] MemRead_Out,
        output logic [PC_W-1:0] PC_IDEX_Out, PCPlus4_IDEX_Out,
        output logic [DATA_W-1:0] Reg1_Out, Reg2_Out, ExtImm_Out,
        output logic [2:0] Funct3_IDEX_Out,
        output logic [RF_ADDRESS_W-1:0] RD_IDEX_Out, RS1_IDEX_Out, RS2_IDEX_Out,
        output logic [6:0] opcode_IDEX_Out, Funct7_IDEX_Out
        );

    always @(posedge clk) begin
        ALUSrc_Out <= ALUSrc;
        MemtoReg_Out <= MemtoReg;
        RegWrite_Out <= RegWrite;
        MemWrite_Out <= MemWrite;
        ALUOp_Out <= ALUOp;
        MemRead_Out <= MemRead;
        PC_IDEX_Out <= IFID_PC;
        PCPlus4_IDEX_Out <= IFID_PCPlus4;
        Reg1_Out <= Reg1;
        Reg2_Out <= Reg2;
        ExtImm_Out <= ExtImm;
        Funct3_IDEX_Out <= Funct3_IFID;
        RD_IDEX_Out <= RD_IFID;
        RS1_IDEX_Out <= RS1_IFID;
        RS2_IDEX_Out <= RS2_IFID;
        opcode_IDEX_Out <= opcode;
        Funct7_IDEX_Out <= Funct7;
    end


endmodule
```

This register was primarily used to send data from the register files as well as some of the control signals from the controller to the EXE stage, this is, to the ALUController and ALU.

3. EXMEM Register

```
`timescale 1ns / 1ps

module RegEXMEM #(
        parameter PC_W = 9, // Program Counter
        parameter RF_ADDRESS_W = 5, // Register File Address
        parameter DATA_W = 32 // Data WriteData
        )(
        input logic clk,

        input logic MemtoReg,
        input logic RegWrite,
        input logic [2:0] MemRead,
        input logic [1:0] MemWrite,
        input logic Branch_Taken,
        input logic [DATA_W-1:0] ALUResult, ALUResult_j, Reg2,
        input logic [RF_ADDRESS_W-1:0] RD,
        input logic [PC_W-1:0] PC, PCPlus4,
        input logic [6:0] opcode,
        input logic [DATA_W-1:0] ExtImm_IDEX,


        output logic MemtoReg_EXMEM_Out,
        output logic RegWrite_EXMEM_Out,
        output logic [2:0] MemRead_EXMEM_Out,
        output logic [1:0] MemWrite_EXMEM_Out,
        output logic Branch_Taken_Out,
        output logic [DATA_W-1:0]  ALUResult_EXMEM_Out, ALUResult_j_EXMEM_Out, Reg2_EXMEM_Out,
        output logic [RF_ADDRESS_W-1:0] RD_EXMEM_Out,
        output logic [PC_W-1:0] PC_EXMEM_Out, PCPlus4_EXMEM_Out,
        output logic [6:0] opcode_EXMEM_Out,
        output logic [DATA_W-1:0] ExtImm_EXMEM_Out

        );

    always @(posedge clk) begin
        MemtoReg_EXMEM_Out <= MemtoReg;
        RegWrite_EXMEM_Out <= RegWrite;
        MemRead_EXMEM_Out <= MemRead;
        MemWrite_EXMEM_Out <= MemWrite;
        Branch_Taken_Out <= Branch_Taken;
        ALUResult_EXMEM_Out <= ALUResult;
        ALUResult_j_EXMEM_Out <= ALUResult_j;
        Reg2_EXMEM_Out <= Reg2;
        RD_EXMEM_Out <= RD;
        PC_EXMEM_Out <= PC;
        PCPlus4_EXMEM_Out <= PCPlus4;
        opcode_EXMEM_Out <= opcode;
        ExtImm_EXMEM_Out <= ExtImm_IDEX;

    end

endmodule
```

This register propagated the results from the ALU, along with some of the control signals received from the IDEX register to the MEM stage.

4. MEMWB Register

```
`timescale 1ns / 1ps


module RegMEMWB #(
        parameter PC_W = 9,
        parameter RF_ADDRESS_W = 5, // Register File Address
        parameter DATA_W = 32 // Data WriteData
        )(
        input logic clk,
        input logic  MemtoReg, RegWrite,
        input logic [DATA_W-1:0] ReadData, ALUResult_EXMEM, ALUResult_j_EXMEM,
        input logic [RF_ADDRESS_W-1:0] RD,
        input logic [PC_W-1:0] PC_EXMEM,
        input logic [6:0] opcode_EXMEM,
        input logic [DATA_W-1:0] ExtImm_EXMEM,

        output logic MemtoReg_MEMWB_Out, RegWrite_MEMWB_Out,
        output logic [DATA_W-1:0] ReadData_MEMWB_Out, ALUResult_MEMWB_Out, ALUResult_j_MEMWB_Out,
        output logic [RF_ADDRESS_W-1:0] RD_MEMWB_Out,
        output logic [PC_W-1:0] PC_MEMWB_Out,
        output logic [6:0] opcode_MEMWB_Out,
        output logic [DATA_W-1:0] ExtImm_MEMWB_Out
        );

    always @(posedge clk)
    begin
        MemtoReg_MEMWB_Out <= MemtoReg;
        RegWrite_MEMWB_Out <= RegWrite;
        ReadData_MEMWB_Out <= ReadData;
        ALUResult_MEMWB_Out <= ALUResult_EXMEM;
        ALUResult_j_MEMWB_Out <= ALUResult_j_EXMEM;
        RD_MEMWB_Out <= RD;
        PC_MEMWB_Out <= PC_EXMEM;
        opcode_MEMWB_Out <= opcode_EXMEM;
        ExtImm_MEMWB_Out <= ExtImm_EXMEM;
    end

endmodule
```

This register was used to send the result of the ALU or Read Data back to the Register File.

Datapath.sv

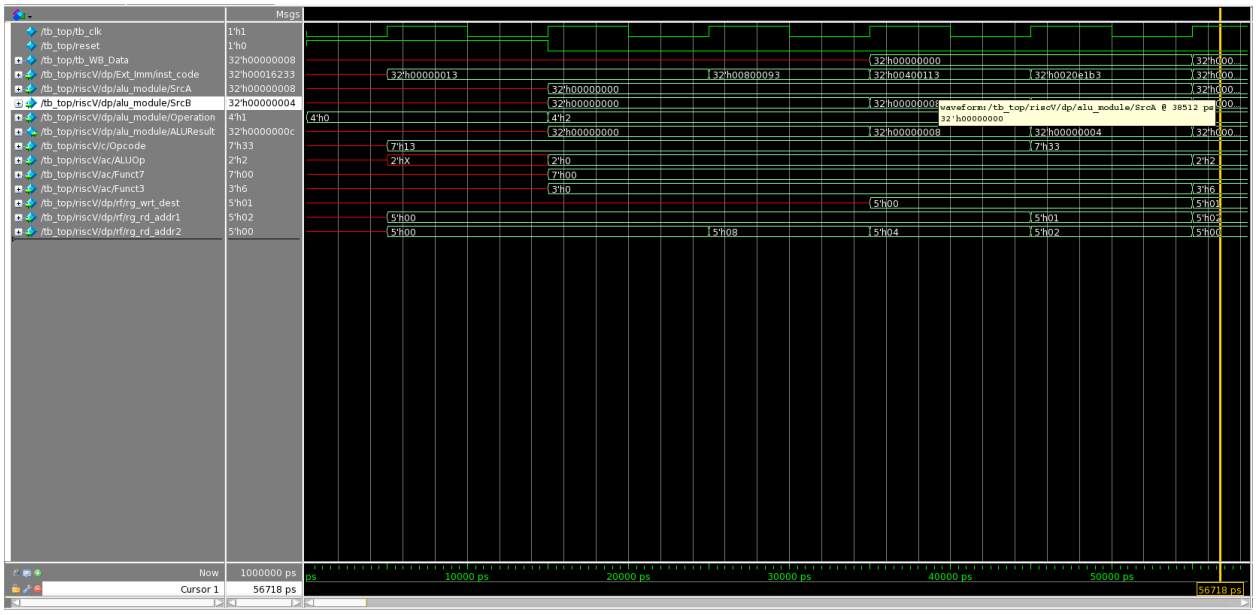
To accommodate all these changes, Datapath was heavily modified. A huge number of connecting wires were introduced to connect the registers, forwarding and hazard detection unit.

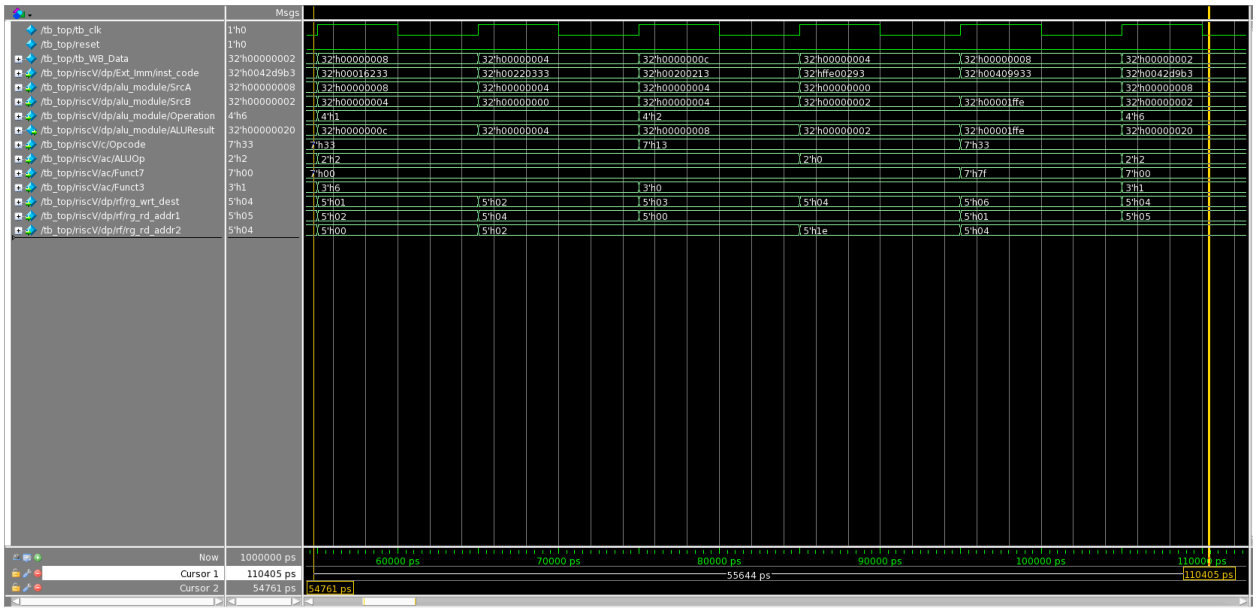
```
logic [PC_W-1:0] IFID_PCIn, IFID_PCPlus4In, IFID_PCOut, IFID_PCPlus4Out, PCPlus4, PC_JALR, selectedPC, PC_temp;
logic [INS_W-1:0] Instr, IFID_InstIn, IFID_InstOut;     // remove instr
//logic [DATA_W-1:0] Result;
logic [DATA_W-1:0] Reg1, Reg2;
logic [DATA_W-1:0] ReadData;
logic [DATA_W-1:0] SrcA;
logic [DATA_W-1:0] ExtImm, PC_extended, PCPlusImm;
logic Branch_Taken, Branch_Taken_EXMEM_Out;
logic [DATA_W-1:0] SW, SH, SB, SelectedWriteData, LW, LH, LHU, LB, LBU;
logic [DATA_W-1:0] out1, out2;
logic [2:0] Load_Inst;
logic [1:0] PCSel;
logic [1:0] ALUop;
logic [1:0] Store_Inst;
logic [DATA_W-1:0] AUIPC, JALR_returnPC;
logic [DATA_W-1:0] SelectedReadData, ReadData_MEMWB_Out;
logic [DATA_W-1:0] SrcB;
logic [RF_ADDRESS_W-1:0] RD, RS1, RS2;
logic [6:0] Funct7;
logic ALUSrc_IDEX_Out, RegWrite_IDEX_Out, RegWrite_EXMEM_Out, RegWrite_MEMWB_Out;
logic MemtoReg_IDEX_Out, MemtoReg_EXMEM_Out, MemtoReg_MEMWB_Out;
logic [2:0] MemRead_IDEX_Out, MemRead_EXMEM_Out;
logic [1:0] MemWrite_IDEX_Out, MemWrite_EXMEM_Out;
logic [PC_W-1:0] PC_IDEX_Out, PCPlus4_IDEX_Out, PC_EXMEM_Out, PCPlus4_EXMEM_Out;
logic [DATA_W-1:0] Reg1_IDEX_Out, Reg2_IDEX_Out, ExtImm_IDEX_Out, Reg2_EXMEM_Out, ALUResult_EXMEM_Out;
logic [DATA_W-1:0] ExtImm_EXMEM_Out;
logic [RF_ADDRESS_W-1:0] RD_IDEX_Out, RS1_IDEX_Out, RS2_IDEX_Out, RD_EXMEM_Out, RD_MEMWB_Out;
logic [6:0] opcode_EXMEM_Out;
logic [DATA_W-1:0] Result_temp, ALUResult_j, ALUResult, ALUResult_j_EXMEM_Out, ALUResult_MEMWB_Out, ALUResult_j_MEMWB_Out;
logic [PC_W-1:0] PC_MEMWB_Out;
logic [6:0] opcode_MEMWB_Out;
logic [DATA_W-1:0] ExtImm_MEMWB_Out;



//For Hazard detection
logic PCWrite, IFIDWrite, HazardSel;
logic ALUSrc_Haz, MemtoReg_Haz, RegWrite_Haz;
logic [1:0] MemWrite_Haz, ALUOp_Haz;
logic [2:0] MemRead_Haz;

//For Forwarding
logic [1:0] ForwardA, ForwardB;
logic [DATA_W-1:0] SrcA_Fwd, SrcB_Fwd;




// Hazard Detector
HazardDetector hzdetector(MemRead_IDEX_Out, RD_IDEX_Out, RS1, RS2,
        PCWrite, IFIDWrite, HazardSel);
hazard_helper hz_hhelp (
        HazardSel, ALUsrc, MemtoReg, RegWrite,
        MemWrite, ALUOp,
        MemRead,

        ALUSrc_Haz, MemtoReg_Haz, RegWrite_Haz,
        MemWrite_Haz, ALUOp_Haz,
        MemRead_Haz);

// Forwarding Unit
ForwardingUnit fwd (RS1_IDEX_Out, RS2_IDEX_Out, RD_EXMEM_Out, RD_MEMWB_Out, RegWrite_EXMEM_Out, RegWrite_MEMWB_Out,
        ForwardA, ForwardB);
forwardA fwdA(ForwardA,
        Reg1_IDEX_Out, ALUResult_EXMEM_Out, WB_Data,
        SrcA_Fwd);
forwardB fwdB (ForwardB,
        Reg2_IDEX_Out, ALUResult_EXMEM_Out, WB_Data,
        SrcB_Fwd);
```

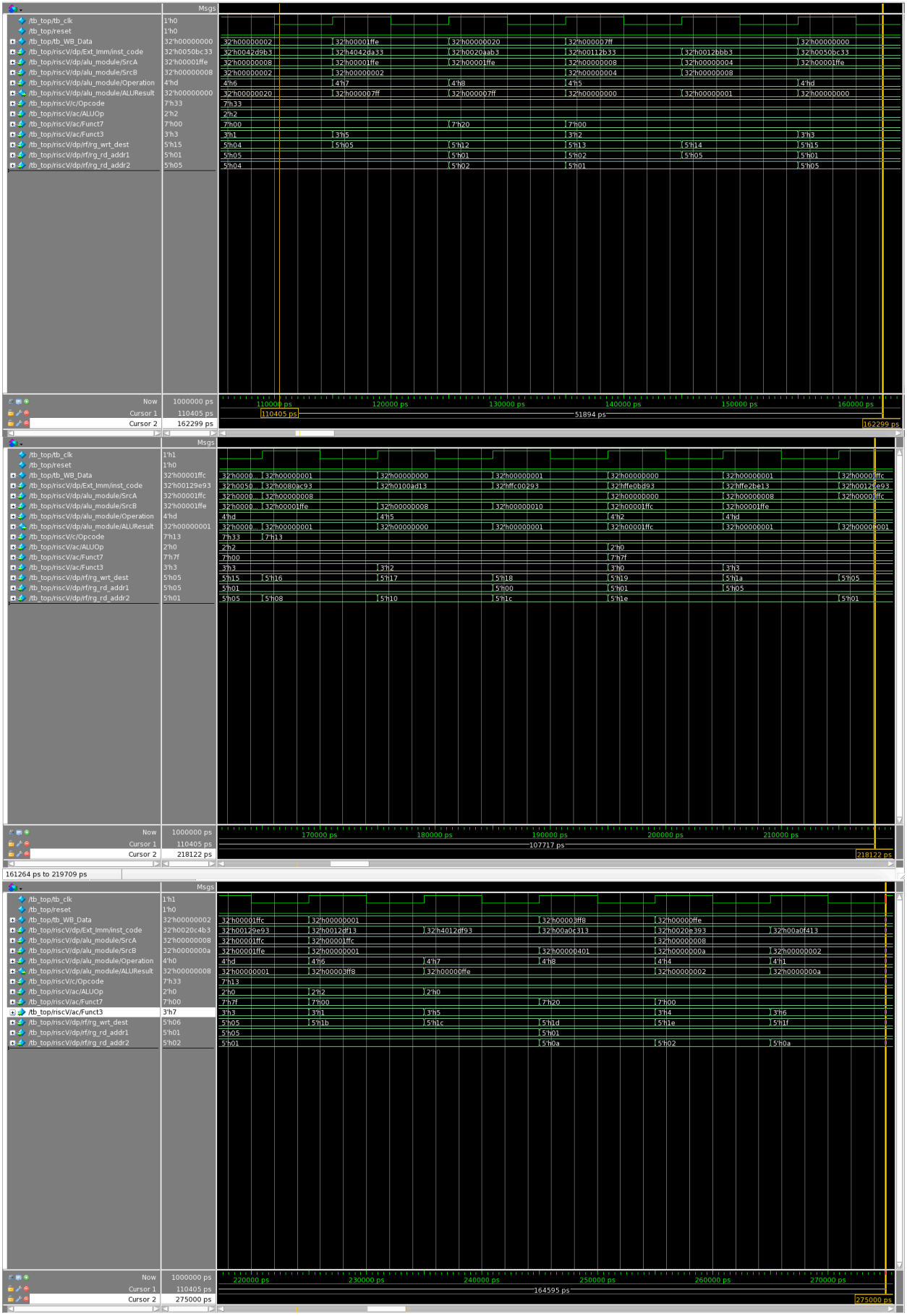
```
// IF ID Register
RegIFID ifid(IFIDWrite, clk, IFID_PCIn, IFID_PCPlus4In, IFID_InstIn, IFID_PCOut, IFID_PCPlus4Out, IFID_InstOut);
assign IFID_opcode_Out = IFID_InstOut[6:0];
assign Funct7 = IFID_InstOut[31:25];
assign IFID_Funct3_Out = IFID_InstOut[14:12];
assign RD  = IFID_InstOut[11:7];
assign RS1    = IFID_InstOut[19:15];
assign RS2    = IFID_InstOut[24:20];
assign ALU_CC_Out = ALU_CC;

// //Register File
RegFile rf(clk, reset, RegWrite_MEMWB_Out, RD_MEMWB_Out, RS1, RS2, WB_Data, Reg1, Reg2);


//// sign extender and immediate generator
imm_Gen Ext_Imm (IFID_InstOut,ExtImm);

//ID EX Register
RegIDEX idex(clk, ALUSrc_Haz, MemtoReg_Haz, RegWrite_Haz, MemWrite_Haz, ALUOp_Haz, MemRead_Haz, IFID_PCOut, IFID_PCPlus4Out,
            Reg1, Reg2, ExtImm, IFID_Funct3_Out, RD, RS1, RS2, IFID_opcode_Out, Funct7,

            ALUSrc_IDEX_Out, MemtoReg_IDEX_Out, RegWrite_IDEX_Out, MemWrite_IDEX_Out, ALUOp_Out, MemRead_IDEX_Out,
            PC_IDEX_Out, PCPlus4_IDEX_Out, Reg1_IDEX_Out, Reg2_IDEX_Out, ExtImm_IDEX_Out, Funct3_IDEX_Out,
            RD_IDEX_Out, RS1_IDEX_Out, RS2_IDEX_Out, opcode_IDEX_Out, Funct7_IDEX_Out);

//// ALU
//mux2 #(32) scramux(Reg1_IDEX_Out, 32'b0, LUI_signal, SrcA);
mux2 #(32) srcbmux(SrcB_Fwd, ExtImm_IDEX_Out, ALUSrc_IDEX_Out, SrcB);
alu alu_module(SrcA_Fwd, SrcB, ALU_CC, ALUResult, Branch_Taken);
mux2 #(32) aluresultmux(ALUResult, {23'b0,PCPlus4_IDEX_Out}, (opcode_IDEX_Out == 7'b1101111 || opcode_IDEX_Out == 7'b1100111), ALUResult_j);



// EX MEM Register
RegEXMEM exmem( clk, MemtoReg_IDEX_Out, RegWrite_IDEX_Out, MemRead_IDEX_Out, MemWrite_IDEX_Out, Branch_Taken, ALUResult, ALUResult_j,
                SrcB_Fwd, RD_IDEX_Out, PC_IDEX_Out, PCPlus4_IDEX_Out, opcode_IDEX_Out, ExtImm_IDEX_Out,

                MemtoReg_EXMEM_Out, RegWrite_EXMEM_Out, MemRead_EXMEM_Out, MemWrite_EXMEM_Out, Branch_Taken_EXMEM_Out,
                ALUResult_EXMEM_Out, ALUResult_j_EXMEM_Out, Reg2_EXMEM_Out, RD_EXMEM_Out, PC_EXMEM_Out, PCPlus4_EXMEM_Out, opcode_EXMEM_Out, ExtImm_EXMEM_Out);

// Data memory
datamemory data_mem (clk, MemRead_EXMEM_Out, MemWrite_EXMEM_Out, ALUResult_j_EXMEM_Out[DM_ADDRESS-1:0], Reg2_EXMEM_Out, ReadData);

// MEM WB Register
RegMEMWB memwb(clk, MemtoReg_EXMEM_Out, RegWrite_EXMEM_Out, ReadData, ALUResult_EXMEM_Out, ALUResult_j_EXMEM_Out, RD_EXMEM_Out,
                PC_EXMEM_Out, opcode_EXMEM_Out, ExtImm_EXMEM_Out,

                MemtoReg_MEMWB_Out, RegWrite_MEMWB_Out, ReadData_MEMWB_Out, ALUResult_MEMWB_Out, ALUResult_j_MEMWB_Out,
                RD_MEMWB_Out, PC_MEMWB_Out, opcode_MEMWB_Out, ExtImm_MEMWB_Out);


//Final Result Mux
mux2 #(32) resmux(ALUResult_j_MEMWB_Out, ReadData_MEMWB_Out, MemtoReg_MEMWB_Out, Result_temp);
mux2 #(32) resmux2(Result_temp, (ExtImm_MEMWB_Out + PC_MEMWB_Out), (opcode_MEMWB_Out == 7'b0010111), WB_Data);
```

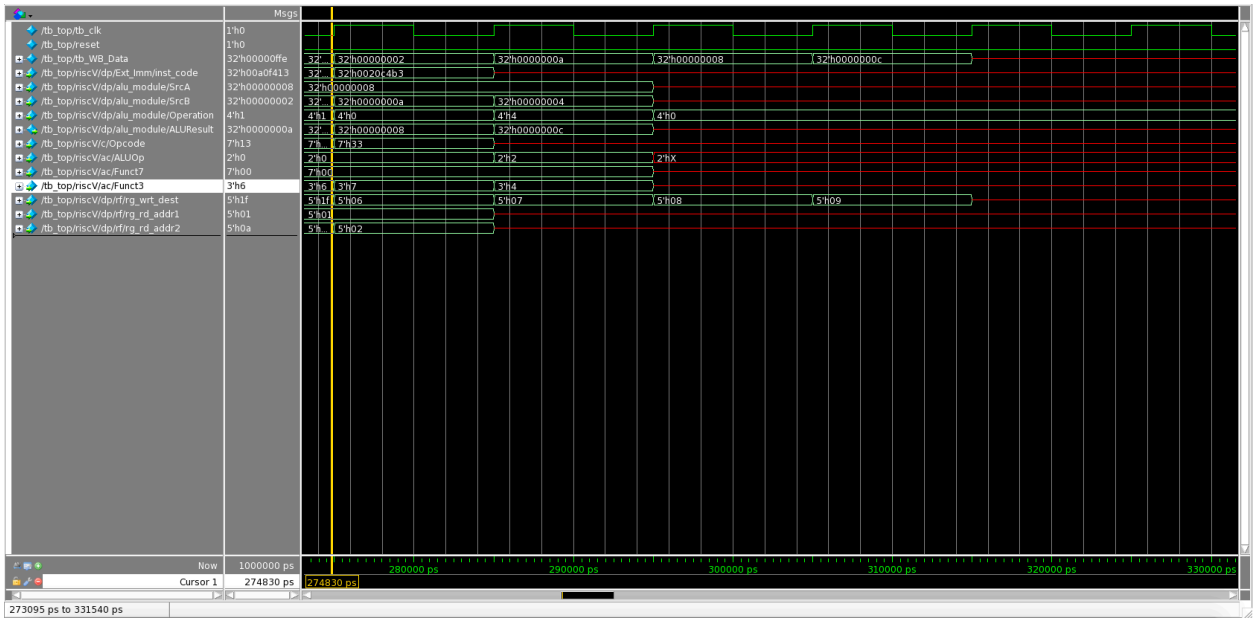
SIMULATION

This is the simulation for the given instruction memory file. All of the instructions were successfully executed, which confirmed that the hazard detection and forwarding unit worked well.

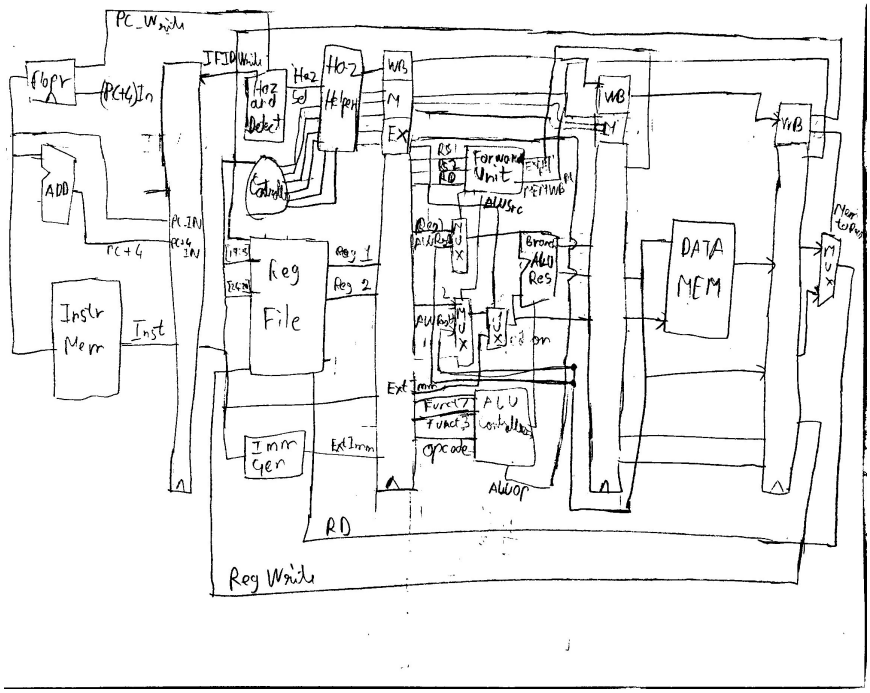





Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_Data
/tb_top/riscV/dp/Ext_Imm/inst_code
/tb_top/riscV/dp/alu_module/SrcA
/tb_top/riscV/dp/alu_module/SrcB
/tb_top/riscV/dp/alu_module/Operation
/tb_top/riscV/dp/alu_module/ALUResult
/tb_top/riscV/c/Opcode
/tb_top/riscV/ac/ALUOp
/tb_top/riscV/ac/Funct7
/tb_top/riscV/ac/Funct3
/tb_top/riscV/dp/rf/rg_wrt_dest
/tb_top/riscV/dp/rf/rg_rd_addr1
/tb_top/riscV/dp/rf/rg_rd_addr2
Now
Cursor 1
Cursor 2
110000 ps
120000 ps
130000 ps
140000 ps
150000 ps
160000 ps
110405 ps
51894 ps
162299 ps
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_Data
/tb_top/riscV/dp/Ext_Imm/inst_code
/tb_top/riscV/dp/alu_module/SrcA
/tb_top/riscV/dp/alu_module/SrcB
/tb_top/riscV/dp/alu_module/Operation
/tb_top/riscV/dp/alu_module/ALUResult
/tb_top/riscV/c/Opcode
/tb_top/riscV/ac/ALUOp
/tb_top/riscV/ac/Funct7
/tb_top/riscV/ac/Funct3
/tb_top/riscV/dp/rf/rg_wrt_dest
/tb_top/riscV/dp/rf/rg_rd_addr1
/tb_top/riscV/dp/rf/rg_rd_addr2
Now
Cursor 1
Cursor 2
170000 ps
180000 ps
190000 ps
200000 ps
210000 ps
107717 ps
218122 ps
161264 ps to 219709 ps
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_Data
/tb_top/riscV/dp/Ext_Imm/inst_code
/tb_top/riscV/dp/alu_module/SrcA
/tb_top/riscV/dp/alu_module/SrcB
/tb_top/riscV/dp/alu_module/Operation
/tb_top/riscV/dp/alu_module/ALUResult
/tb_top/riscV/c/Opcode
/tb_top/riscV/ac/ALUOp
/tb_top/riscV/ac/Funct7
/tb_top/riscV/ac/Funct3
/tb_top/riscV/dp/rf/rg_wrt_dest
/tb_top/riscV/dp/rf/rg_rd_addr1
/tb_top/riscV/dp/rf/rg_rd_addr2
Now
Cursor 1
Cursor 2
220000 ps
230000 ps
240000 ps
250000 ps
260000 ps
270000 ps
164595 ps
275000 ps


Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_Data
/tb_top/riscV/dp/Ext_Imm/inst_code
/tb_top/riscV/dp/alu_module/SrcA
/tb_top/riscV/dp/alu_module/SrcB
/tb_top/riscV/dp/alu_module/Operation
/tb_top/riscV/dp/alu_module/ALUResult
/tb_top/riscV/c/Opcode
/tb_top/riscV/ac/ALUOp
/tb_top/riscV/ac/Funct7
/tb_top/riscV/ac/Funct3
/tb_top/riscV/dp/rf/rg_wrt_dest
/tb_top/riscV/dp/rf/rg_rd_addr1
/tb_top/riscV/dp/rf/rg_rd_addr2
1'h0
1'h0
32'h00000ffe
32'h00a0f413
32'h00000008
32'h00000002
4'h1
32'h0000000a
7'h13
2'h0
7'h00
3'h6
5'h1f
5'h01
5'h0a
32'h00000002
32'h0000000a
32'h00000008
32'h0000000c
32'h00020c4b3
32'h00000004
4'h0
4'h4
4'h0
7'h33
2'h2
2'hX
3'h7
3'h4
5'h06
5'h07
5'h08
5'h09
5'h02
Now
1000000 ps
Cursor 1
274830 ps
274830 ps
280000 ps
290000 ps
300000 ps
310000 ps
320000 ps
330000 ps
273095 ps to 331540 ps

BLOCK DIAGRAM



Attached below is the risk.qor.rpt for critical path length, critical path slack and area.

From this report we can see that the overall compile time decreased from 2023.08 in the single cycle processor to 1345.18 in the pipelined processor. Similarly, the Overall Compile Wall Clock Time decreased from 2039.64 to 1355.95.

COMPARISON of AREAS

1. SINGLE CYCLE PROCESSOR

Combinational Area: 88903.383668

Noncombinational Area: 115259.898943

Buf/Inv Area: 5475.786740

Total Buffer Area: 3166.38

Total Inverter Area: 2309.41

Macro/Black Box Area: 0.000000

Net Area: 172200.492779 Cell Area: 214879.773248

Design Area: 338351.764397

2. PIPELINED PROCESSOR.

Area

Combinational Area: 98120.67879

Noncombinational Area:

116759.094455

Buf/Inv Area: 11613.872847

Total Buffer Area: 9064.81

Total Inverter Area: 2549.06

Macro/Black Box Area: 0.000000

Net Area: 123471.991149 Cell Area: 214879.773248

Design Area: 338351.764397

1. PIPELINED PROCESSOR REPORT

Information: Updating design information... (UID-85)

Warning: Design 'riscv' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)

****

Report : qor

Design : riscv

Version: J-2014.09-SP4

Date : Sat Mar 24 06:26:42 2018

****

Timing Path Group 'clk'

Levels of Logic: 21.00

Critical Path Length: 4.23

Critical Path Slack: -0.24

Critical Path Clk Period: 4.00

Total Negative Slack: -2436.18

No. of Violating Paths: 15574.00

Worst Hold Violation: 0.00

Total Hold Violation: 0.00

No. of Hold Violations: 0.00

Cell Count

Hierarchical Cell Count: 4

Hierarchical Port Count: 311

Leaf Cell Count: 57788

Buf/Inv Cell Count: 6128

Buf Cell Count: 4190

Inv Cell Count: 1938

CT Buf/Inv Cell Count: 0

Combinational Cell Count: 40119

Sequential Cell Count: 17669

Macro Count: 0

Area

Combinational Area: 98120.678792

Noncombinational Area:

116759.094455

Buf/Inv Area: 11613.872847

Total Buffer Area: 9064.81

Total Inverter Area: 2549.06

Macro/Black Box Area: 0.000000

Net Area: 123471.991149

Cell Area: 214879.773248

Design Area: 338351.764397

Design Rules

Total Number of Nets: 57842

Nets With Violations: 47

Max Trans Violations: 0

Max Cap Violations: 47

Hostname: zuma.eecs.uci.edu

Compile CPU Statistics

Resource Sharing: 24.28

Logic Optimization: 50.12

Mapping Optimization: 941.87

Overall Compile Time: 1345.18

Overall Compile Wall Clock Time: 1355.95

Design WNS: 0.24 TNS: 2436.18 Number of Violating Paths: 15574

Design (Hold) WNS: 0.00 TNS: 0.00 Number of Violating Paths: 0

---

1. SINGLE CYCLE PROCESSOR

Information: Updating design information... (UID-85)

Warning: Design 'riscv' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)

****

Report : qor

Design : riscv

Version: J-2014.09-SP4

Date : Mon Mar 19 06:30:54 2018

****

Timing Path Group 'clk'

Levels of Logic: 37.00

Critical Path Length: 5.46

Critical Path Slack: -3.49

Critical Path Clk Period: 4.00

Total Negative Slack: -56957.62

No. of Violating Paths: 17416.00

Worst Hold Violation: 0.00

Total Hold Violation: 0.00

No. of Hold Violations: 0.00

Cell Count

Hierarchical Cell Count: 5

Hierarchical Port Count: 333

Leaf Cell Count: 53454

Buf/Inv Cell Count: 3206

Buf Cell Count: 1393

Inv Cell Count: 1813

CT Buf/Inv Cell Count: 0

Combinational Cell Count: 36006

Sequential Cell Count: 17448

Macro Count: 0

Area

Combinational Area: 88903.383668

Noncombinational Area:

115259.898943

Buf/Inv Area: 5475.786740

Total Buffer Area: 3166.38

Total Inverter Area: 2309.41

Macro/Black Box Area: 0.000000

Net Area: 172200.492779

Cell Area: 204163.282611

Design Area: 376363.775391

Design Rules

Total Number of Nets: 53473

Nets With Violations: 47

Max Trans Violations: 0

Max Cap Violations: 47

Hostname: zuma.eecs.uci.edu

Compile CPU Statistics

Resource Sharing: 12.71

Logic Optimization: 93.31

Mapping Optimization: 1183.53

Overall Compile Time: 2023.08

Overall Compile Wall Clock Time: 2039.64

Design WNS: 3.49 TNS: 56957.62 Number of Violating Paths: 17416

Design (Hold) WNS: 0.00 TNS: 0.00 Number of Violating Paths: 0
