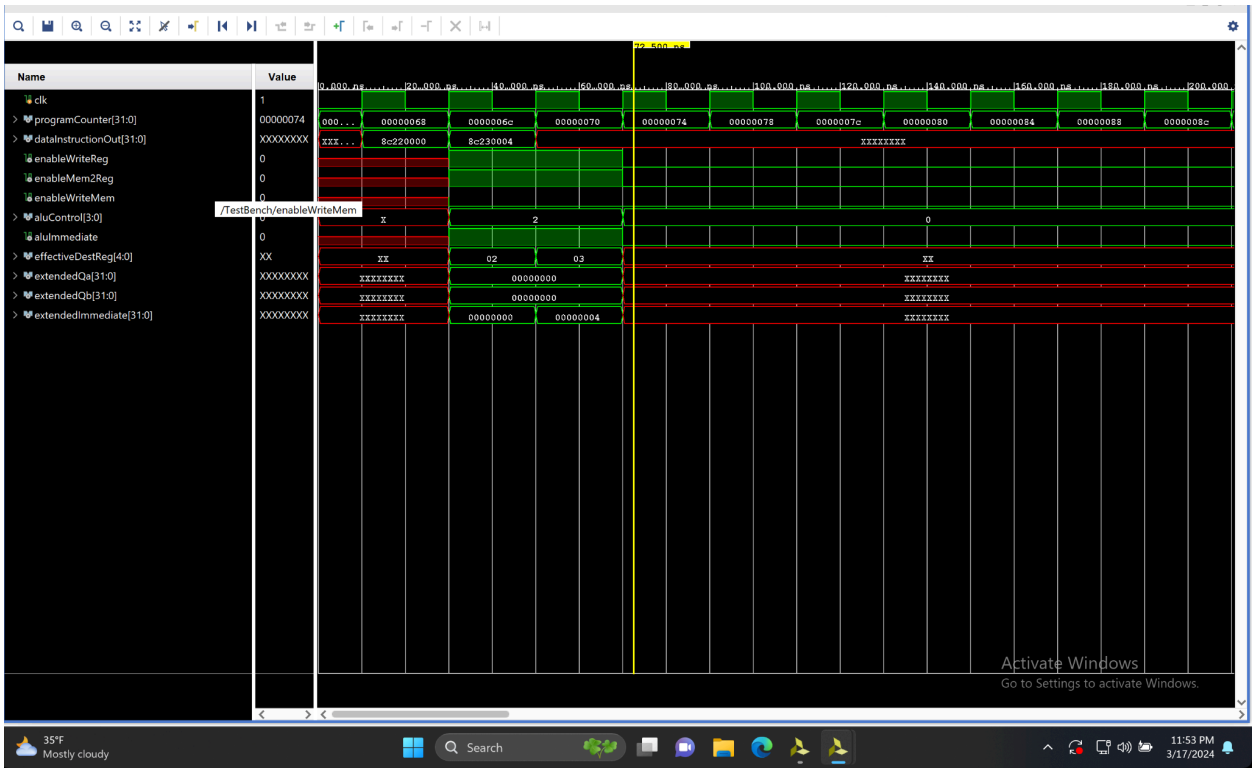
Sethu Senthil

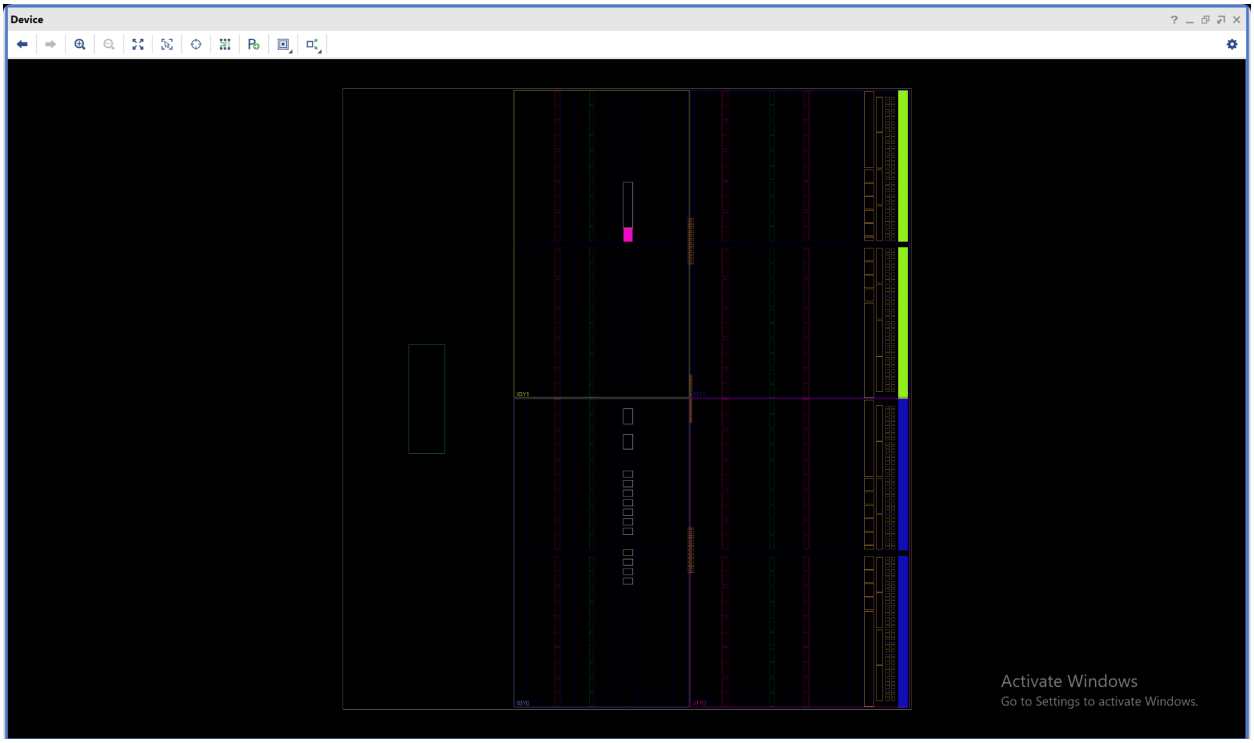
Lab 3

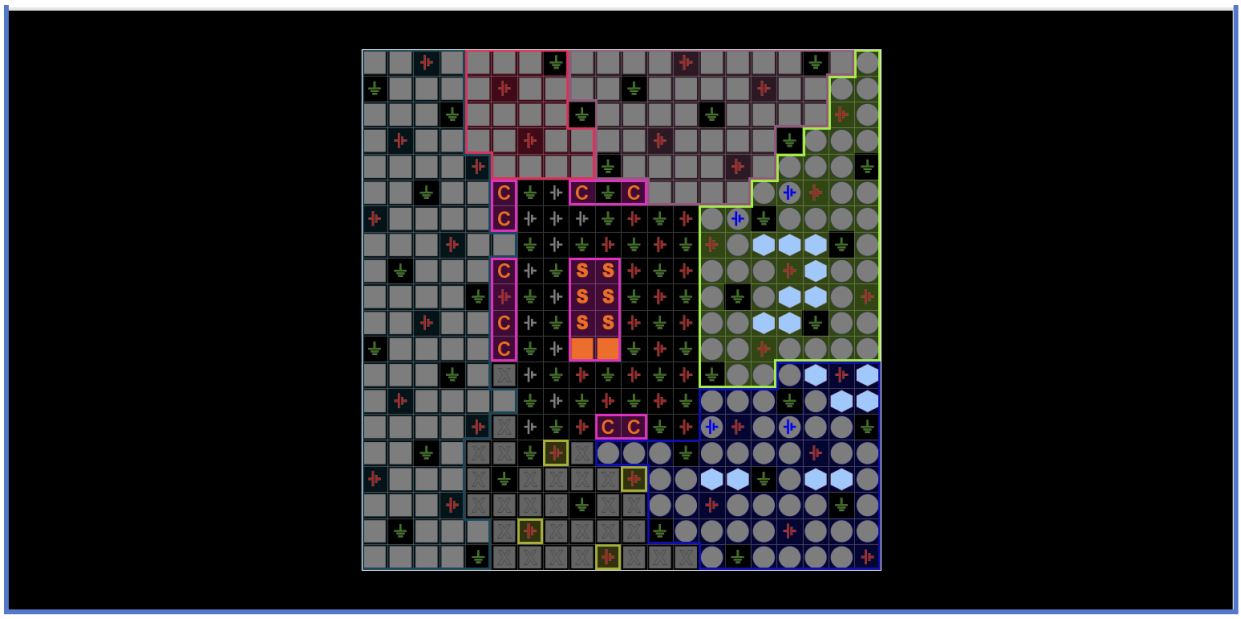
Section 1, CMPEN 331

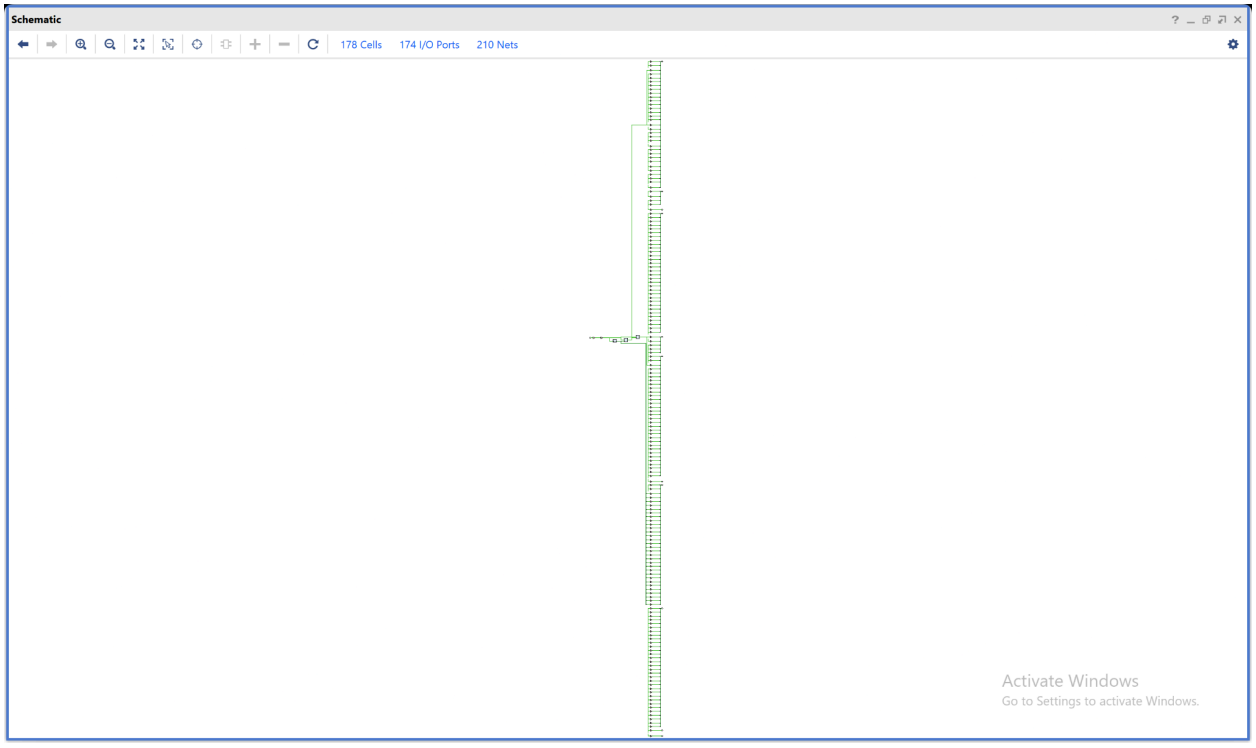
sns5787

Images:


Name
Value
clk
programCounter[31:0]
dataInstructionOut[31:0]
enableWriteReg
enableMem2Reg
enableWriteMem
aluControl[3:0]
aluImmediate
effectiveDestReg[4:0]
extendedQa[31:0]
extendedQb[31:0]
extendedImmediate[31:0]
/TestBench/enableWriteMem
Activate Windows
Go to Settings to activate Windows.
35°F
Mostly cloudy
Search
11:53 PM
3/17/2024


Device
Activate Windows
Go to Settings to activate Windows.




Schematic
178 Cells
174 I/O Ports
210 Nets
Activate Windows
Go to Settings to activate Windows.

Datapath Code:

```
module Datapath(
    input clk,
    output wire [31:0] programCounter,
    output wire [31:0] dataInstructionOut,
    output wire enableWriteReg,
    output wire enableMem2Reg,
    output wire enableWriteMem,
    output wire [3:0] aluControl,
    output wire aluImmediate,
    output wire [4:0] effectiveDestReg,
    output wire [31:0] extendedQa,
    output wire [31:0] extendedQb,
    output wire [31:0] extendedImmediate
);
    wire writeRegEnable;
    wire mem2RegEnable;
    wire writeMemEnable;
    wire aluImmediateCalc;
    wire regRt;
    wire [4:0] destinationReg;
    wire [31:0] extendedA;
```

```
    wire [31:0] extendedB;
    wire [31:0] immediate32;
    wire [31:0] nextProgramCounter;
    wire [31:0] instructionOut;
    wire [3:0] aluCtrl;
    wire [15:0] immediate;
    wire [4:0] sourceRs;
    wire [4:0] targetRt;
    wire [4:0] destinationRd;
    wire [5:0] opcode;
    wire [5:0] functionCode;


    ProgramCounter ProgramCounter(.clk(clk), .nextPc(nextProgramCounter),
.pc(programCounter));
    pcAdder pcAdder_db(.pc(programCounter), .nextPc(nextProgramCounter));
    InstructionMemory InstructionMemory_db(.pc(programCounter),
.instOut(instructionOut));
    IFIDpipelineReg IFIDpipelineReg_db(.clk(clk), .instOut(instructionOut),
.dinstOut(dataInstructionOut));
    ControlUnit controlUnit_db(.op(opcode), .func(functionCode), .wreg(writeRegEnable),
.m2reg(mem2RegEnable), .wmem(writeMemEnable), .aluc(aluCtrl),
.aluimm(aluImmediateCalc), .regrt(regRt));
```

```
    RegrtMultiplexer RegrtMultiplexer_db(.rt(targetRt), .rd(destinationRd), .regrt(regRt),
.destReg(destinationReg));
    RegisterFile RegisterFile_db(.rs(sourceRs), .rt(targetRt), .qa(extendedA),
.qb(extendedB));
    ImmediateExtender ImmediateExtender_db(.imm(immediate),
.imm32(immediate32));
    IDEXEpipeline IDEXEpipeline_db(
        .wreg(writeRegEnable),
        .m2reg(mem2RegEnable),
        .wmem(writeMemEnable),
        .aluc(aluCtrl),
        .aluimm(aluImmediateCalc),
        .destReg(destinationReg),
        .qa(extendedA),
        .qb(extendedB),
        .imm32(immediate32),
        .clk(clk),
        .ewreg(enableWriteReg),
        .em2reg(enableMem2Reg),
        .ewmem(enableWriteMem),
        .ealuc(aluControl),
        .ealuimm(aluImmediate),
        .edestReg(effectiveDestReg),
```

```
        .eqa(extendedQa),
        .eqb(extendedQb),
        .eimm32(extendedImmediate)
    );

    assign opcode = dataInstructionOut[31:26];
    assign functionCode = dataInstructionOut[5:0];
    assign sourceRs = dataInstructionOut[25:21];
    assign targetRt = dataInstructionOut[20:16];
    assign destinationRd = dataInstructionOut[15:11];
    assign immediate = dataInstructionOut[15:0];



module ProgramCounter( //PC pipeline
    input clk,
    input [31:0] nextPc,
    output reg [31:0] pc
);

initial
    begin
        pc = 32'd100; //set
```

```
    end

always @(posedge clk) //update
    begin
        pc = nextPc;
    end

endmodule




module pcAdder(
    input [31:0] pc,
    output reg [31:0] nextPc
);

    always @(*) begin
        nextPc = pc + 32'b00000000000000000000000000000100;
    end

endmodule
```

```
module InstructionMemory(
    input [31:0] pc,
    output reg [31:0] instOut
    );
       reg [31:0] memory [0:63]; //32x64

       initial begin
        memory[25] = {6'b100011, 5'b00001, 5'b00010, 5'b00000, 5'b00000, 6'b000000};
//lw $v0, 00($at)

        memory[26] = {6'b100011, 5'b00001, 5'b00011, 5'b00000, 5'b00000, 6'b000100};
//lw $v1, 04($at)
       end

    always @ (*)
        begin
            instOut = memory[pc[7:2]];
        end
endmodule


module IFIDpipelineReg(
    input clk,
```

```
    input [31:0] instOut,
    output reg [31:0] dinstOut
    );

    always @ (posedge clk)
        begin
            dinstOut = instOut;
        end

Endmodule

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
    );
```

```
// Continuously update control signals based on decoded instruction
always @ (*) begin

  // Decode instruction opcode
  case (op)

    // R-type instructions
    6'b000000: begin
      // Decode R-type function code
      case (func)

        // ADD instruction
        6'b100000: begin
          // Set control signals for ADD:
          // - Write result to register file
          // - No memory access
          // - Use ALU for addition
          // - Source operands from registers
          // - Destination register address is second source (rs2)
          wreg = 1'b1;
          m2reg = 1'b0;
          wmem = 1'b0;
          aluc = 4'b0010; // ALU operation for addition
```

```
      aluimm = 1'b0;
      regrt = 1'b0;
    end

    // Default behavior for unspecified R-type functions
    default: begin
      // Set default control signals (e.g., all signals to 0)
      // for unspecified R-type instructions
      wreg = 1'b0;
      m2reg = 1'b0;
      wmem = 1'b0;
      aluc = 4'b0000;
      aluimm = 1'b0;
      regrt = 1'b0;
    end
  endcase
end

// LW instruction (load word from memory)
6'b100011: begin
      // Set control signals for LW:
      // - Write result to register file
      // - Read data from memory
```

```
      // - No memory write
      // - Use ALU for address calculation (addition)
      // - ALU source from registers and immediate
      // - Destination register address is third source (rt)
      wreg = 1'b1;
      m2reg = 1'b1;
      wmem = 1'b0;
      aluc = 4'b0010; // ALU operation for addition
      aluimm = 1'b1;
      regrt = 1'b1;
end

// Default behavior for unspecified opcodes
default: begin
  // Set default control signals (e.g., all signals to 0)
  // for unspecified instructions
  wreg = 1'b0;
  m2reg = 1'b0;
  wmem = 1'b0;
  aluc = 4'b0000;
  aluimm = 1'b0;
  regrt = 1'b0;
end
```

```
  endcase
end

module RegrtMultiplexer(
    input [4:0] rt,
    input [4:0] rd,
    input regrt,
    output reg [4:0] destReg
    );

    always @(*)
    begin
        if (regrt == 0)
            destReg = rd;
        else
            destReg = rt;
    end
endmodule

module RegisterFile(
    //inputs
    input [4:0] rs,
    input [4:0] rt,
```

```
    //outputs
    output reg [31:0] qa,
    output reg [31:0] qb
    );

    reg [31:0] register [0:31]; //32x32

    integer r;
    initial begin
        for (r = 0; r <= 31; r = r + 1) begin
            register[r] = 0; //init all to 0
        end
    end

    always @ (*)
        begin //block updates
            qa = register[rs];
            qb = register[rt];
        end

endmodule
```

```
module ImmediateExtender(
    input [15:0] imm,
    output reg [31:0] imm32
    );

    always @ (*)
        begin
            imm32 = {{16{imm[15]}}, imm};
        end
endmodule



module IDEXEpipeline(
    input wreg,
    input m2reg,
    input wmem,
    input [3:0] aluc,
    input aluimm,
    input [4:0] destReg,
    input [31:0] qa,
    input [31:0] qb,
    input [31:0] imm32,
    input clk,
```

```
output reg ewreg,
output reg em2reg,
output reg ewmem,
output reg [3:0] ealuc,
output reg ealuimm,
output reg [4:0] edestReg,
output reg [31:0] eqa,
output reg [31:0] eqb,
output reg [31:0] eimm32
);

always @ (posedge clk)
    begin
        ewreg = wreg;
        em2reg = m2reg;
        ewmem = wmem;
        ealuc = aluc;
        ealuimm = aluimm;
        edestReg = destReg;
        eqa = qa;
        eqb = qb;
        eimm32 = imm32;
```

```
        end
Endmodule
```

Workbench Code:

```
module TestBench;

    reg clk;

    wire [31:0] programCounter;
    wire [31:0] dataInstructionOut;
    wire enableWriteReg;
    wire enableMem2Reg;
    wire enableWriteMem;
    wire [3:0] aluControl;
    wire aluImmediate;
    wire [4:0] effectiveDestReg;
    wire [31:0] extendedQa;
    wire [31:0] extendedQb;
    wire [31:0] extendedImmediate;
```

```
initial begin
    clk <= 1'b0;
end

Datapath datapath(
    .clk(clk),
    .programCounter(programCounter),
    .dataInstructionOut(dataInstructionOut),
    .enableWriteReg(enableWriteReg),
    .enableMem2Reg(enableMem2Reg),
    .enableWriteMem(enableWriteMem),
    .aluControl(aluControl),
    .aluImmediate(aluImmediate),
    .effectiveDestReg(effectiveDestReg),
    .extendedQa(extendedQa),
    .extendedQb(extendedQb),
    .extendedImmediate(extendedImmediate)
);


always begin
    #10;
    clk = ~clk;
```

```
    end


endmodule
```