Course Name: DSD

Course Number and Section: **14:332:434:02**

Experiment: [Experiment # 6 – Spooky registers]

Lab Instructor:

Date Performed: Today

Date Submitted: Tommorrow

Submitted by: Kai Gardner – kzg5 : 188002688

Course Name: DSD

Course Number and Section: **14:332:434:02**

! Important: Please include this page in your report if the submission is a paper submission. For electronic submission (email or Sakai) please omit this page.

----------For Lab Instructor Use ONLY----------

GRADE: _____

COMMENTS:

Electrical and Computer Engineering Department
School of Engineering
Rutgers University, Piscataway, NJ 08854
ECE Lab Report Structure

1. Purpose / Introduction / Overview – describe the problem and provide background information

2. Approach / Method – the approach took, how problems were solved

3. Results – present your data and analysis, experimental results, etc.

4. Conclusion / Summary – what was done and how it was done

This page is unintentionally left blank.

Top.sv

```
module top(
    input  logic        clk,
    input  logic        rst,
    input  logic [1:0]  sw,  // selects instruction: 1=LW example, 2=SW example
    output logic [31:0] ALUResult,              // observable for pre-lab
    output logic [31:0] RD1, RD2,               // RF read ports (visible)
    output logic [31:0] prode_register_file,    // RF[1] probe
    output logic [31:0] prode_data_memory,      // DM[2] probe
    output logic [6:0]  display_led             // in-lab (seven-seg segments)
);

    // I-type instruction examples from the lab handout
    // LW: op=010101, rs=00000, rt=00001, imm=...0101 (5)
    localparam logic [31:0] INST_LW = 32'b010101_00000_00001_0000_0000_0000_0101;
    // SW: op=010100, rs=00000, rt=00110, imm=...0010 (2)
    localparam logic [31:0] INST_SW = 32'b010100_00000_00110_0000_0000_0000_0010;


    logic [31:0] instr;
    logic [7:0]  memProdeInd; // DM probe index (0..255)
    logic [4:0]  regProdeInd; // RF probe index (0..31)
    always_comb begin
        // defaults to avoid latches
        instr = 32'b0;
        memProdeInd = 8'd0;
        regProdeInd = 5'd0;
        unique case (sw)
            2'b01: begin
                instr = INST_LW;
                memProdeInd = 8'd5; // observe DM[5]
```

```
logic [31:0] instr;
logic [7:0]  memProdeInd; // DM probe index (0..255)
logic [4:0]  regProdeInd; // RF probe index (0..31)
always_comb begin
    // defaults to avoid latches
    instr = 32'b0;
    memProdeInd = 8'd0;
    regProdeInd = 5'd0;
    unique case (sw)
        2'b01: begin
            instr = INST_LW;
            memProdeInd = 8'd5; // observe DM[5]
            regProdeInd = 5'd1; // observe RF[1]
        end
        2'b10: begin
            instr = INST_SW;
            memProdeInd = 8'd2; // observe DM[2]
            regProdeInd = 5'd6; // observe RF[6]
        end
        default: /* NOP */ ;
    endcase
end

// Instruction fields
logic [5:0]  op;
logic [4:0]  rs, rt, rd;
```

```
// Instruction fields
logic [5:0]  op;
logic [4:0]  rs, rt, rd;
logic [15:0] imm;
assign op  = instr[31:26];
assign rs  = instr[25:21];
assign rt  = instr[20:16];
assign rd  = instr[15:11];
assign imm = instr[15:0];

// Control signals mapped from op: {RegDst, ALUSrc, ALUControl[2:0], MemtoReg}
logic RegDst, ALUSrc;
logic [2:0] ALUControl;
logic MemtoReg;
assign RegDst    = op[5];
assign ALUSrc    = op[4];
assign ALUControl= op[3:1];
assign MemtoReg  = op[0];

// For this lab's two ops: LW (MemtoReg=1) writes reg; SW (MemtoReg=0) writes memory
logic RegWrite, MemWrite;
assign RegWrite = MemtoReg;     // LW:1, SW:0
assign MemWrite = ~MemtoReg;    // LW:0, SW:1
```

```
// Modules
sign_extend u_se(
    .imm16(imm),
    .signimm(SignImm)
);

mux_regdst u_mux_regdst(
    .RegDst(RegDst), .rt(rt), .rd(rd), .RegDst_out(writeReg)
);

register_file u_rf(
    .clk(clk), .rst(rst), .WE3(RegWrite), .prodeInd(regProdeInd),
    .A1(rs), .A2(rt), .A3(writeReg),
    .WD3(WD3),
    .RD1(RD1), .RD2(RD2),
    .prode(prode_register_file)
);

mux_alusrc u_mux_alusrc(
    .ALUSrc(ALUSrc), .RD2(RD2), .SignImm(SignImm), .ALUSrc_out(srcB)
);

ALU u_alu(
    .SrcA(RD1), .SrcB(srcB), .ALUControl(ALUControl), .ALUResult(ALUResult)
);
```

Data_memory.sv

```
module data_memory(
    input logic clk, rst,
    input logic [31:0] A,
    input logic [31:0] WD,
    input logic WE,
    input logic [7:0] prodeInd,
    output logic [31:0] RD,
    output logic [31:0] prode
);
    logic [31:0] mem [0:255];

    integer i;
    initial begin
        for (i = 0; i < 256; i = i + 1) begin
            mem[i] = i[31:0];
        end
    end

    assign RD = mem[A[7:0]];

    assign prode = mem[prodeInd];

        always @(posedge clk) begin
        if (WE) begin
            mem[A[7:0]] <= WD;
        end
    end
endmodule
```

Register_file.sv

```
module register_file (
    input  logic        clk, rst, WE3,
    input  logic [4:0]  A1, A2, A3,
    input  logic [31:0] WD3,
    input  logic [4:0]  prodeInd,
    output logic [31:0] RD1, RD2,
    output logic [31:0] prode
);
    logic [31:0] rf [31:0];

    initial begin
        for (int i = 0; i < 32; i++) rf[i] = i;
    end

    assign RD1   = rf[A1];
    assign RD2   = rf[A2];


    assign prode = rf[prodeInd];

    always @(posedge clk) begin
        if (WE3) rf[A3] <= WD3;
    end
endmodule
```

Muxs

```
module mux_regdst(
    input  logic       RegDst,   // 0: rt, 1: rd
    input  logic [4:0] rt,
    input  logic [4:0] rd,
    output logic [4:0] RegDst_out
);
    always_comb begin
        RegDst_out = RegDst ? rd : rt;
    end
endmodule
```

```
module mux_memtoreg(
    input  logic       MemtoReg,     // 0: ALUResult, 1: RD
    input  logic [31:0] ALUResult,
    input  logic [31:0] RD,          // from data memory
    output logic [31:0] MemtoReg_out
);
    always_comb begin
        MemtoReg_out = MemtoReg ? RD : ALUResult;
    end
endmodule
```

```
module mux_alusrc(
    input  logic       ALUSrc,     // 0: RD2, 1: SignImm
    input  logic [31:0] RD2,
    input  logic [31:0] SignImm,
    output logic [31:0] ALUSrc_out
);
    always_comb begin
        ALUSrc_out = ALUSrc ? SignImm : RD2;
    end
endmodule
```

Sign extender

```
module sign_extend(
    input  logic [15:0] imm16,
    output logic [31:0] signimm
);
    always_comb begin
        signimm = {{16{imm16[15]}}, imm16};
    end
endmodule
```

Waveform