BILKENT UNIVERSITY

COMPUTER ENGINEERING DEPT.

CS223 LAB 5 SECTION 4

LAB REPORT

ZÜBEYİR BODUR

21702382

1. SystemVerilog Modules

```
/**
 * Title: Simple Processor Datapath
 * Author: Zubeyir Bodur
 * ID: 21702382
 * Section: 4
 * Lab : 5
 * Description: Design sources
 */
`timescale 1ns / 1ps

module datapathSim(
    output logic [3:0] ALUResult,
    input logic clk, wr_en,
    input logic [1:0] ALUSel,
    input logic [2:0] AddrSrc1, AddrSrc2, AddrDest,
    input logic isExternal,
    input logic [3:0] EXTDATA
    );
    logic [3:0] rdd1, rdd2, wrd;

    logic [1:0] [3:0] inputsMux;
    assign inputsMux = {EXTDATA, ALUResult};
    mux#(4, 1) ext_mux(wrd, inputsMux, isExternal);
    registerFile RF(rdd1, rdd2, AddrSrc1, AddrSrc2, AddrDest, wrd,
wr_en, clk);
    alu ALU(ALUResult, rdd1, rdd2, ALUSel);
endmodule
```

```
module datapath(
   output logic a, b, c, d, e, f, g, dp,
   output logic [3:0] an,
   input logic clk, pushButn,
   input logic [1:0] ALUSel,
   input logic [2:0] AddrSrc1, AddrSrc2, AddrDest,
   input logic isExternal,
   input logic [3:0] EXTDATA
   );
   logic [3:0] ALUResult;
   logic wr_en;
   datapathSim dpath(ALUResult, clk, wr_en, ALUSel,
            AddrSrc1, AddrSrc2, AddrDest,
            isExternal, EXTDATA);
   debouncer debounce(wr_en, clk, pushButn);
   sevenSegmentDisplay ssd(a, b, c, d, e, f, g, dp, an, clk,
                    {3'b0,ALUResult[3]},
                    {3'b0,ALUResult[2]},
                    {3'b0,ALUResult[1]},
                    {3'b0,ALUResult[0]});
endmodule
```

```
// LED positions inside 7-segment :
//   A
// F   B
//   G
// E   C
//   D
// DP
// digit positions on Basys3 :
// in3(msb), in2, in1, in0(lsb)
module sevenSegmentDisplay(
   output logic a, b, c, d, e, f, g, dp,   //individual LED output
for the 7-segment along with the digital point
   output logic [3:0] an,                 // anode: 4-bit enable
signal (active low)
   input logic clk,
   input logic [4:0] in3, in2, in1, in0   // 4 values for 4 digits
(decimal value)
   );
   // divide system clock (100Mhz for Basys3) by 2^N using a counter
   localparam N = 18;
   logic [N-1:0] count = {N{1'b0}}; //initial value

   always_ff @(posedge clk) begin
      count <= count + 1;
   end

   logic [4:0] digit_val; // 7-bit register to hold the current
data on output
   logic [3:0] digit_en; //register for enable vector
```

```
always_comb begin
   digit_en = 4'b1111; //default
   digit_val = in0; //default
   case(count[N-1:N-2]) //using only the 2 MSB's of the counter

      2'b00 : begin //select first 7Seg.
         digit_val = in0;
         digit_en = 4'b1110;
      end

      2'b01: begin //select second 7Seg.
         digit_val = in1;
         digit_en = 4'b1101;
      end

      2'b10: begin //select third 7Seg.
         digit_val = in2;
         digit_en = 4'b1011;
      end

      2'b11: begin //select forth 7Seg.
         digit_val = in3;
         digit_en = 4'b0111;
      end
   endcase
end
```

```
    //Convert digit number to LED vector. LEDs are active low.
    logic [6:0] sseg_LEDs;
    always_comb begin
        sseg_LEDs = 7'b1111111; //default
        case(digit_val)
            5'd0 : sseg_LEDs = 7'b1000000; //to display 0
            5'd1 : sseg_LEDs = 7'b1111001; //to display 1
            5'd2 : sseg_LEDs = 7'b0100100; //to display 2
            5'd3 : sseg_LEDs = 7'b0110000; //to display 3
            5'd4 : sseg_LEDs = 7'b0011001; //to display 4
            5'd5 : sseg_LEDs = 7'b0010010; //to display 5
            5'd6 : sseg_LEDs = 7'b0000010; //to display 6
            5'd7 : sseg_LEDs = 7'b1111000; //to display 7
            5'd8 : sseg_LEDs = 7'b0000000; //to display 8
            5'd9 : sseg_LEDs = 7'b0010000; //to display 9
            5'd10 : sseg_LEDs = 7'b0001000; //to display A
            5'd11 : sseg_LEDs = 7'b0000011; //to display B
            5'd12 : sseg_LEDs = 7'b1000110; //to display C
            5'd13 : sseg_LEDs = 7'b0100001; //to display D
            5'd14 : sseg_LEDs = 7'b0000110; //to display E
            5'd15 : sseg_LEDs = 7'b0001110; //to display F
            5'd16 : sseg_LEDs = 7'b0111111; // to display -
            5'd17 : sseg_LEDs = 7'b0110111; // to display =
            5'd18 : sseg_LEDs = 7'b0100111; // to display c
            default :  sseg_LEDs = 7'b0111111; // display - otherwise
        endcase
    end
    assign an = digit_en;
    assign {g, f, e, d, c, b, a} = sseg_LEDs;
    assign dp = 1'b1; //turn dp off
endmodule
```

```
// 8x4 register file
module registerFile(
   output logic [3:0] rdd1, rdd2,
   input logic [2:0] rda1, rda2,
   input logic [2:0] wra,
   input logic [3:0] wrd,
   input logic wr_en, clk
   );
   logic [3:0] RAM [7:0];
   typedef enum logic {init, read_write} statetype;
   statetype state, nextstate;

   always_ff @(posedge clk)
      state <= nextstate;

   always_comb
      case (state)
         init :     nextstate = read_write;
         read_write: nextstate = read_write;
         default :   nextstate = init;
      endcase

   always_ff @(posedge clk)
      case (state)
         read_write: begin
            rdd1 <= RAM[rda1];
            rdd2 <= RAM[rda2];
            if (wr_en) begin
               RAM[wra] <= wrd;
            end
         end
         init : RAM <= {4'h0, 4'h0, 4'h0, 4'h0,
                        4'h0, 4'h0, 4'h0, 4'h0};
      endcase
endmodule
```

```
module debouncer(
   output logic btn_out,
   input logic clk, btn_in
   );
   logic [24:0] timer;
   typedef enum logic [1:0]{S0,S1,S2,S3} states;
   states state, nextState;
   logic gotInput;

   always_ff@(posedge clk)
      begin
         state <= nextState;
         if(gotInput)
            timer <= 25000000;
         else
            timer <= timer - 1;
      end
   always_comb
      case(state)
         S0: if(btn_in)
            begin //startTimer
               nextState = S1;
               gotInput = 1;
            end
            else begin nextState = S0; gotInput = 0; end
         S1: begin nextState = S2; gotInput = 0; end
         S2: begin nextState = S3; gotInput = 0; end
         S3: begin if(timer == 0) nextState = S0; else nextState
= S3; gotInput = 0; end
         default: begin nextState = S0; gotInput = 0; end
         endcase

   assign btn_out = ( state == S1 );
endmodule
```

```
module alu#(parameter N = 4)(
   output logic [N-1:0] res,
   input logic [N-1:0] A, B,
   input logic [1:0]  sel
   );
   always_comb
      case(sel)
         2'b00: res = A + 1;
         2'b01: res = A + B;
         2'b10: res = A - B;
         2'b11: res = A | B;
         default: res = {N{1'bZ}};
      endcase
endmodule

// register with SYNCHRONOUS reset
module register#(parameter N = 1)(
   output logic [N - 1: 0] Q,
   input logic clk, rst,
   input logic [N - 1: 0] D
   );
   logic [N - 1 : 0] d_;
   // mux2_1#(N) enmux(d_, D, Q, en); // multiplexer for enable
   logic [1:0] [N - 1 : 0] inputs;
   assign inputs = {{N{1'b0}}, D};
   mux#(N, 1) rstmux(d_, inputs, rst); // multiplexer for reset
   always_ff @(posedge clk) begin
      Q <= d_;
   end
endmodule

// M bit 2**N : 1 mux
module mux#(parameter M = 1, N = 1)(
   output logic [M - 1 : 0] Y,
   input logic [2**N - 1:0] [M - 1 : 0] D,
    input logic [N - 1:0] S
    );
    always_comb begin
        Y = D[S];
    end
endmodule

// N : 2**N, one hot decoder
module dec#(parameter N = 2)(
    output logic [2**N - 1:0] Y,
    input logic [N - 1:0] A
    );
    always_comb begin
        Y = 0;
        Y[A] = 1;
    end
endmodule
```

2. Testbenches

```
/**
 * Title: Simple Processor Datapath
 * Author: Zubeyir Bodur
 * ID: 21702382
 * Section: 4
 * Lab : 5
 * Description: Testbenches
 */
`timescale 1ns / 1ps

module datapathSimTest(
    );
    logic [3:0] ALUResult;
    logic clk, wr_en;
    logic [1:0] ALUSel;
    logic [2:0] AddrSrc1, AddrSrc2, AddrDest;
    logic isExternal;
    logic [3:0] EXTDATA;
    integer i, j;

    datapathSim uut(.ALUResult(ALUResult),
                    .clk(clk),
                    .wr_en(wr_en),
                    .ALUSel(ALUSel),
                    .AddrSrc1(AddrSrc1),
                    .AddrSrc2(AddrSrc2),
                    .AddrDest(AddrDest),
                    .isExternal(isExternal),
                    .EXTDATA(EXTDATA));
    initial begin
        wr_en = 0; clk = 0; AddrSrc1 = 0; AddrSrc2 = 3'b111;
        AddrDest = 0; EXTDATA = 0; ALUSel = 2'b0; isExternal = 0;
#10;
        for (i = 0; i < 32; i = i + 1) begin
            #1 clk = ~clk;
        end
        for (i = 0; i < 4; i = i + 1) begin
            for (j = 0; j < 8; j = j + 1) begin
                #1 clk = ~clk; #1 clk = ~clk;
                ALUSel = ALUSel + 1;
            end
            AddrSrc1 = AddrSrc1 + 1;
            AddrSrc2 = AddrSrc2 - 1;
        end
        #5 EXTDATA = 4'b1111; AddrDest = 3'b101; isExternal = 1;
        for (i = 0; i < 6; i = i + 1) begin
            #1 clk = ~clk;
        end
        #5 wr_en = 1;
        for (i = 0; i < 6; i = i + 1) begin
            #1 clk = ~clk;
        end
        #5 wr_en = 0;
        for (i = 0; i < 4; i = i + 1) begin
            for (j = 0; j < 8; j = j + 1) begin
                #1 clk = ~clk; #1 clk = ~clk;
                ALUSel = ALUSel + 1;
            end
            AddrSrc1 = AddrSrc1 + 1;
            AddrSrc2 = AddrSrc2 - 1;
        end
        $stop;
    end
endmodule
```

```
// simple testbench for a register file
module registerFileTest(
    );
    logic [3:0] rdd1, rdd2;
    logic [2:0] rda1, rda2;
    logic [2:0] wra;
    logic [3:0] wrd;
    logic wr_en, clk;
    integer i;

    registerFile RF(.rdd1(rdd1),
                    .rdd2(rdd2),
                    .rda1(rda1),
                    .rda2(rda2),
                    .wra(wra),
                    .wrd(wrd),
                    .wr_en(wr_en),
                    .clk(clk));

    initial begin
        wr_en = 0; clk = 0; rda1 = 0; rda2 = 3'b111; wra = 0; wrd
= 0; #10
        for (i = 0; i < 32; i = i + 1) begin
            #1 clk = ~clk;
        end
        for (i = 0; i < 32; i = i + 1) begin
            #1 clk = ~clk; #1 clk = ~clk;
            rda1 = rda1 + 1;
            rda2 = rda2 - 1;
        end
        #5 wrd = 4'b1111; wra = 3'b101; #5 wr_en = 1;
        for (i = 0; i < 4; i = i + 1) begin
            #1 clk = ~clk;
        end
        #5 wr_en = 0;
        for (i = 0; i < 32; i = i + 1) begin
```

```
            #1 clk = ~clk; #1 clk = ~clk;
            rda1 = rda1 + 1;
            rda2 = rda2 - 1;
        end
        $stop;
    end
endmodule

// working ALU
module aluTest(
    );
    parameter N = 4;
    logic [N - 1:0] A, B, ALUResult;
    logic [1:0] ALUSel;
    integer i, j, k;
    alu#(N) uut(.res(ALUResult),
                .A(A),
                .B(B),
                .sel(ALUSel));
    initial begin
        A = 0; B = 0; ALUSel = 0; #10
        for (i = 0; i < 4 ; i = i + 1) begin
            for (j = 0; j < 2**N ; j = j + 1) begin
                for (k = 0; k < 2**N ; k = k + 1) begin
                    #10 B = B + 1;
                end
                #10 A = A + 1;
            end
            #10 ALUSel = ALUSel + 1;
        end
        $stop;
    end
endmodule
```