CS 223 Digital Design

Section 5

Lab 5

Name: Esad Ismail Tok

ID: 21801679

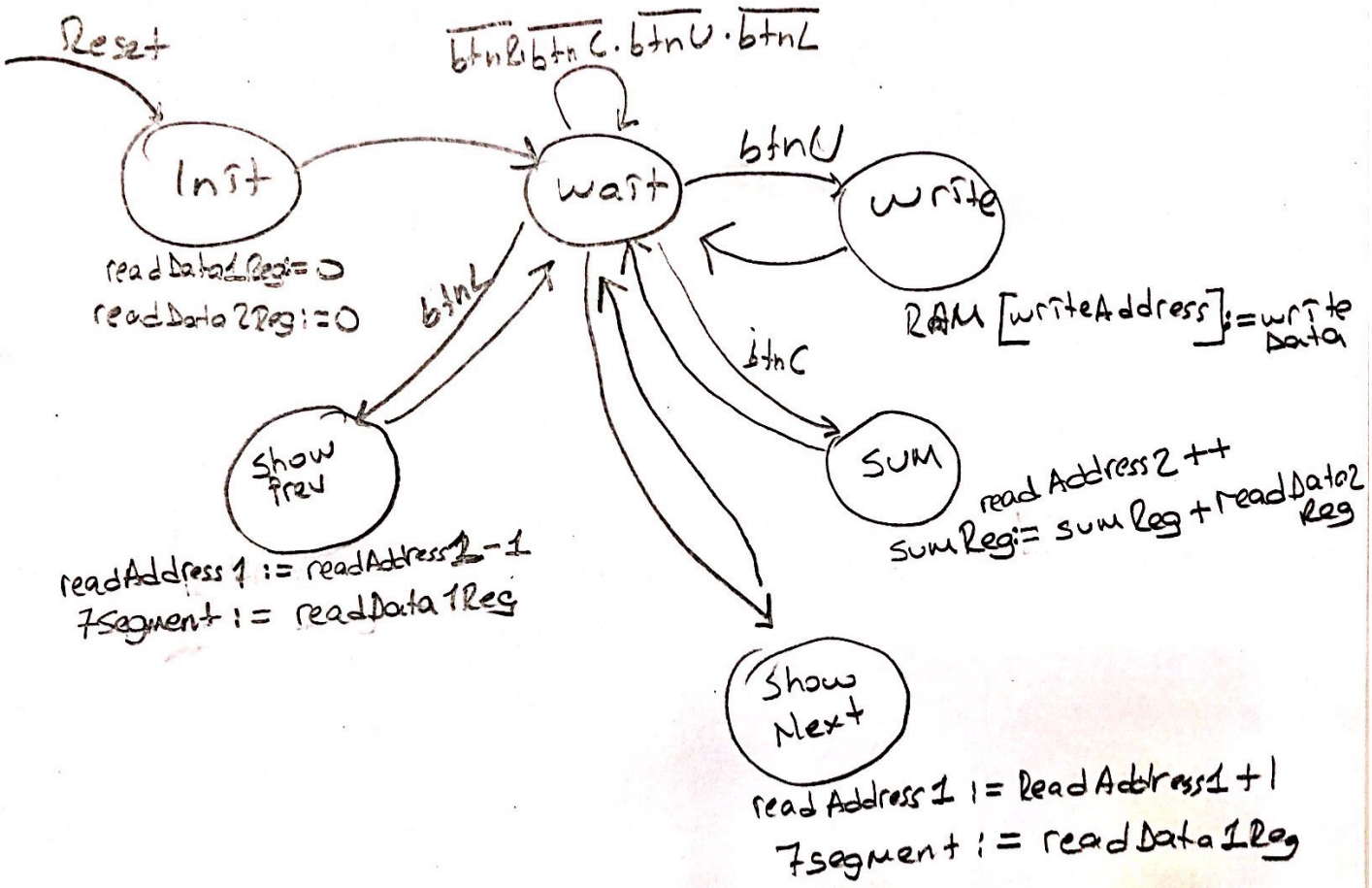
Date: 13.12.2020

~HLSM Diagrams ~

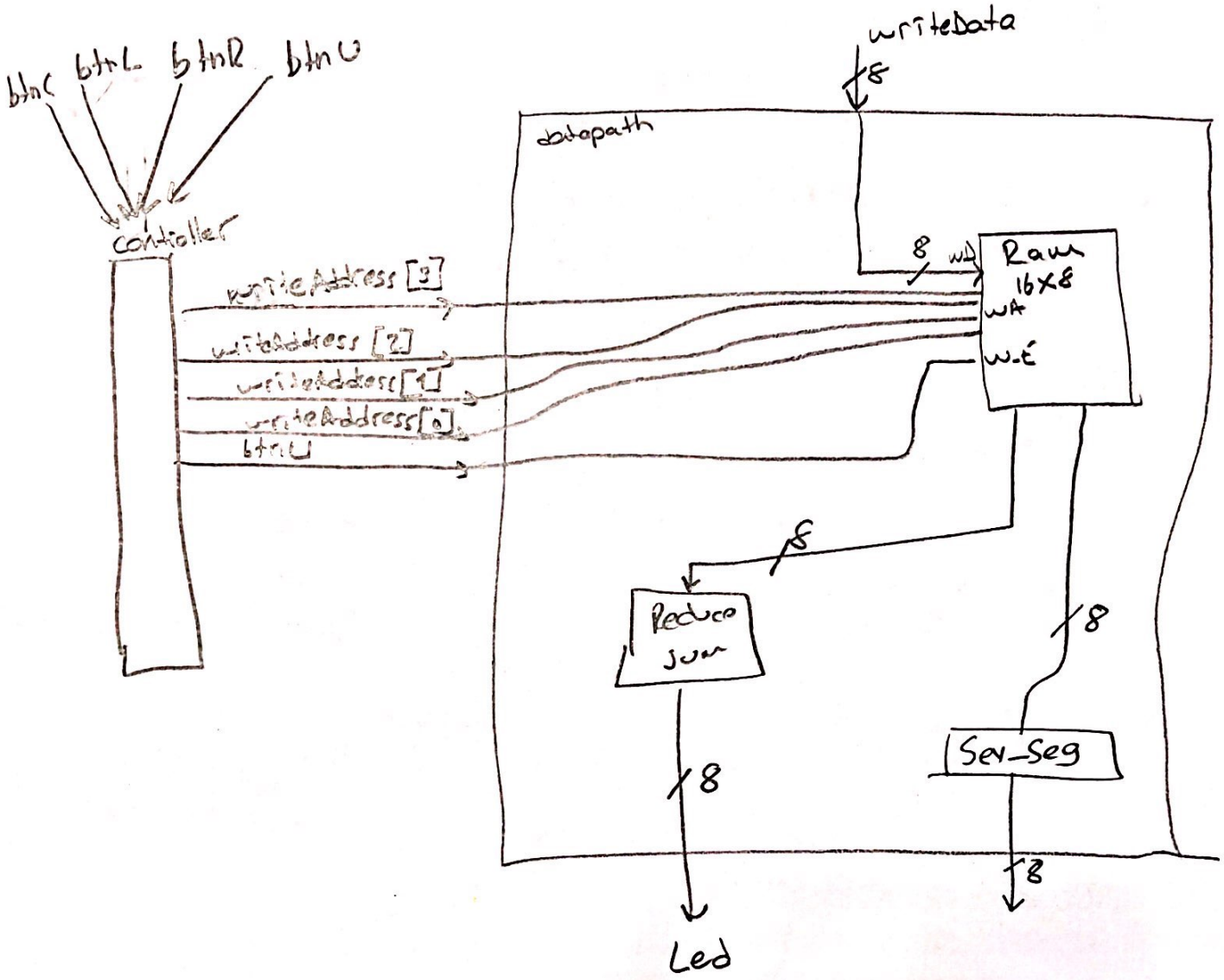
Inputs: btnC, btnU, btnL, btnR (bit)
writeData (8 bits)
writeAddress, readAddress1, readAddress2 (4 bits)

Outputs: ReadData1, readData2 (8 bits), sum (16 bits)

LocalStorage readData1Reg, readData2Reg (8bits)
RAM (16x8 bits), sumReg (16 bits)



~Controller Datapath Diagram~



Debouncer

In the circuits, controllers need clean signal transitions from the pushbutton switches. However in realistic circuits, the signal is not clean all the time. When the pushbutton is pressed the contact is bounces and the signals becomes intermittent until it settles down.
This signal from the pushbutton needs to be cleaned, in other words all glitches must be cleaned for the consistency of the circuitry.

The debouncer module allows the pushbutton signal to be cleaned from the glitches in the signals and by using debouncer controller receives clean signal transitions from the pushbutton switch.

Memory Module and TestBench

```
`timescale 1ns / 1ps

// Low level memory module that represents a 16x8 RAM
module memory(input logic clk, writeEnable,
              input logic[3:0] writeAddress, readAddress1, readAddress2,
              input logic [7:0] writeData,
              output logic [7:0] readData1, readData2);

    logic [7:0] RAM[15:0]; // Instantiating the memory array as a RAM format

    // First assign all memory array values to 0 to avoud X values
    initial begin
        for (int i = 0; i <= 15; i++)
            RAM[i] = 0;
    end

    // Reading the data from both read ports
    assign readData1 = RAM[readAddress1];
    assign readData2 = RAM[readAddress2];

    // Writing the data when enabled at the clock edges
    always @(posedge clk) begin
        if (writeEnable)
            RAM[writeAddress] <= writeData;
    end

endmodule


// TestBench for the memory module
module memoryTB();

    // Local Signals
    logic clk, writeEnable;
    logic[3:0] writeAddress, readAddress1, readAddress2;
    logic [7:0] writeData, readData1, readData2;

    // Instantiating the device under test
    memory dut(clk, writeEnable, writeAddress, readAddress1, readAddress2,
writeData, readData1, readData2);

    // Managing the clock
    always begin
```

```
    clk = 1; #5;
    clk = 0; #5;
  end

  // Testing the module according to the inputs
  initial begin
    writeEnable = 0; readAddress1 = 3; readAddress2 = 15; #10;
    writeEnable = 1; writeAddress = 5; writeData = 127; #10;
    writeEnable = 0; readAddress1 = 5; readAddress2 = 15; #10;
    writeEnable = 1; writeAddress = 15; writeData = 254; #10;
    writeEnable = 0; readAddress1 = 5; readAddress2 = 15; #10;
    writeEnable = 1; writeAddress = 14; writeData = 112; #10;
    writeEnable = 1; writeAddress = 13; writeData = 5; #10;
    writeEnable = 1; writeAddress = 6; writeData = 20; #10;
    writeEnable = 1; writeAddress = 5; writeData = 200; #10;
    writeEnable = 0; readAddress1 = 5; readAddress2 = 14; #10;
  end


endmodule
```

ReduceSum Module and TestBench

```
`timescale 1ns / 1ps

// Low level module that outputs the sum of the entered array elements
module ReduceSum(input logic [7:0] data, output logic [15:0] led);
    logic [15:0] sum = 0;

    // Upgrade the sum according to memory array items
    always@* begin
        sum = sum + data;
    end

    assign led = sum;

endmodule


// TestBench for the ReduceSum module
module ReduceSumTB();

    // Local Signals
    logic [7:0] data;
    logic [15:0] led;


    // Instantiating device under test
    ReduceSum dut(data, led);

    // First assign all memory array values to 0 to avoud X values


    // Testing the output according to the inputs
    initial begin
        data = 5; #10;
        data = 20; #10;
        data = 10; #10;
        data = 15;  #10;
        data = 120;  #10;
        data = 1;  #10;
        data = 9; #10;
    end

endmodule
```

Top Level HLSM Module

```
`timescale 1ns / 1ps

module ReduceSumHLSM(input logic clk, btnC, btnU, btnL, btnR,
                     input logic [7:0] writeData,
                     input logic [3:0] address,
                     output logic [15:0] led,
                     output [6:0]seg, logic dp,
                     output [3:0] an);

    // Defining debounced button signals
    logic buttonCenter, buttonUp, buttonLeft, buttonRight;
    debounce(clk, btnC, buttonCenter);
    debounce(clk, btnU, buttonUp);
    debounce(clk, btnL, buttonLeft);
    debounce(clk, btnR, buttonRight);

    // Signal that keeps track of the memory address that is initially 0
    logic [3:0] adr = 0;

    // Defining the read addresses
    logic [3:0] readAddress1 = adr;
    logic [3:0] readAddress2;

    // Definig the read datas
    logic [7:0] readData1, readData2;

    // Instantiating the RAM
    memory RAM(clk, buttonUp, address, readAddress1, readAddress2,
writeData, readData1, readData2);

    // Instantiating 7 segment display
    SevSeg_4digit SevSeg(clk, readAddress1, x, readData1[7:4],
readData1[3:0], seg, dp, an);


    // Instantiating the ReduceSum module
    ReduceSum Sum(readData2, led);



    always_ff@ (posedge clk) begin
        if (buttonRight) begin
```

```
            if (adr == 15)
                adr = 0;

            else
                adr = adr + 1;
        end

        else if (buttonLeft) begin
            if (adr == 0)
                adr = 15;

            else
                adr = adr - 1;
        end
    end



endmodule
```