CS223: Digital Design

Section 01

Lab 5

Efe Beydoğan

21901548

12.12.2020

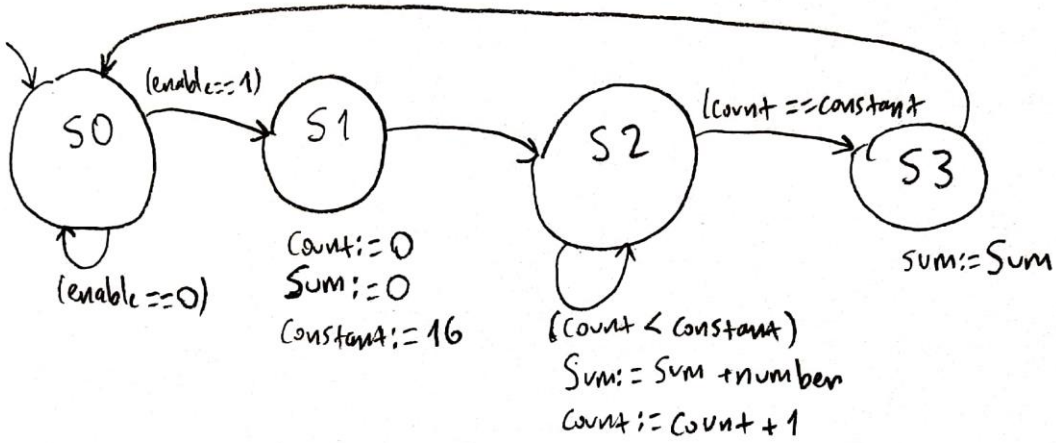
a) <u>Show the ReduceSum HLSM. Show also the controller/datapath diagram for it:</u>

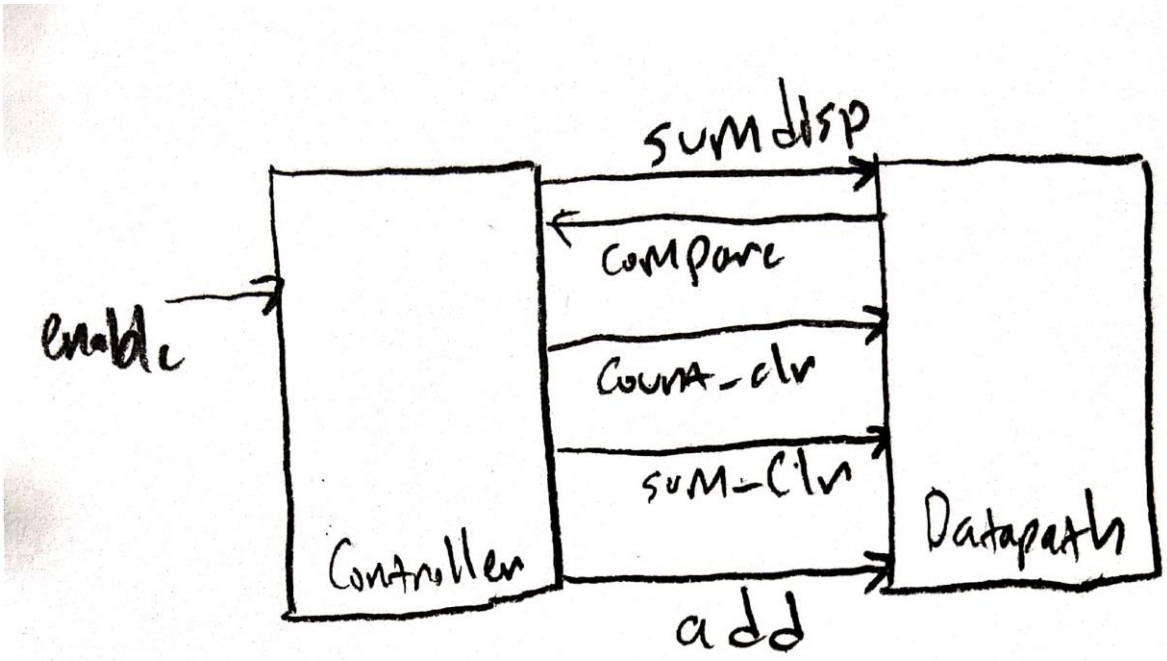
ReduceSum HLSM:

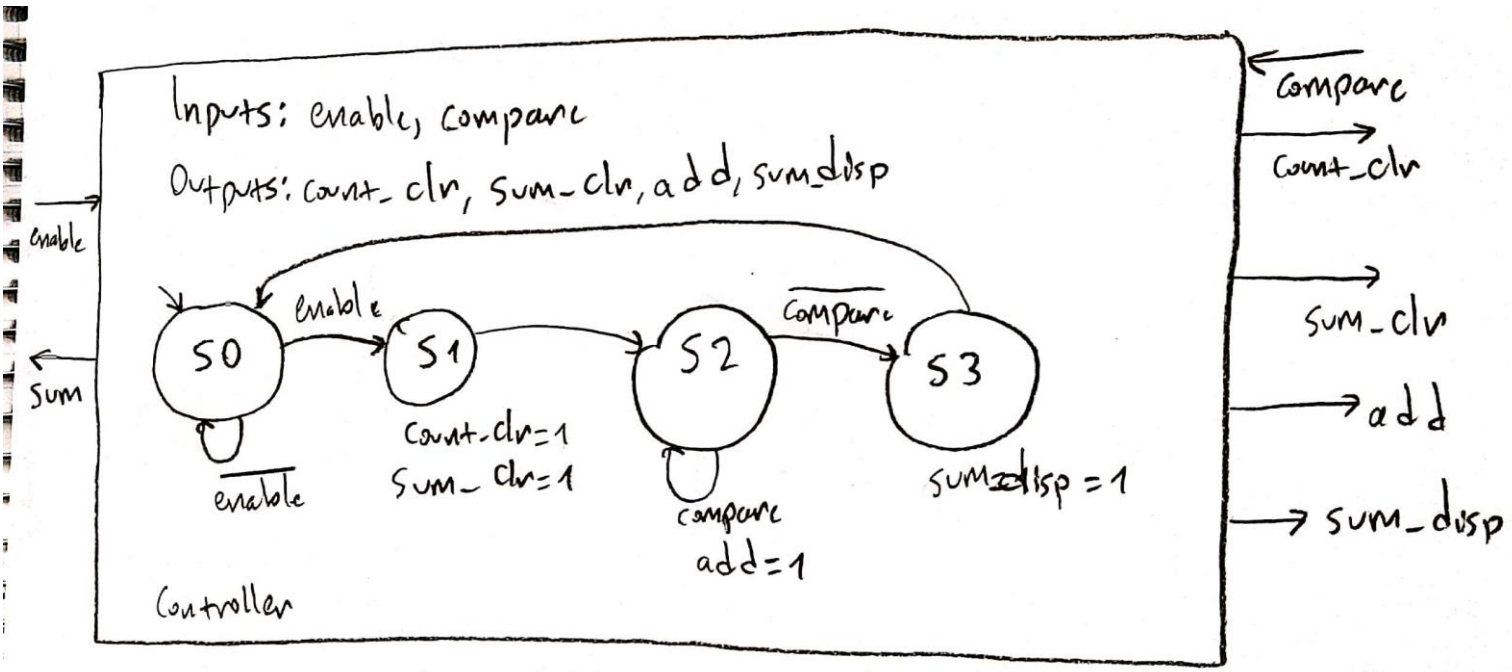
Inputs: enable (bit), number (8 bit)

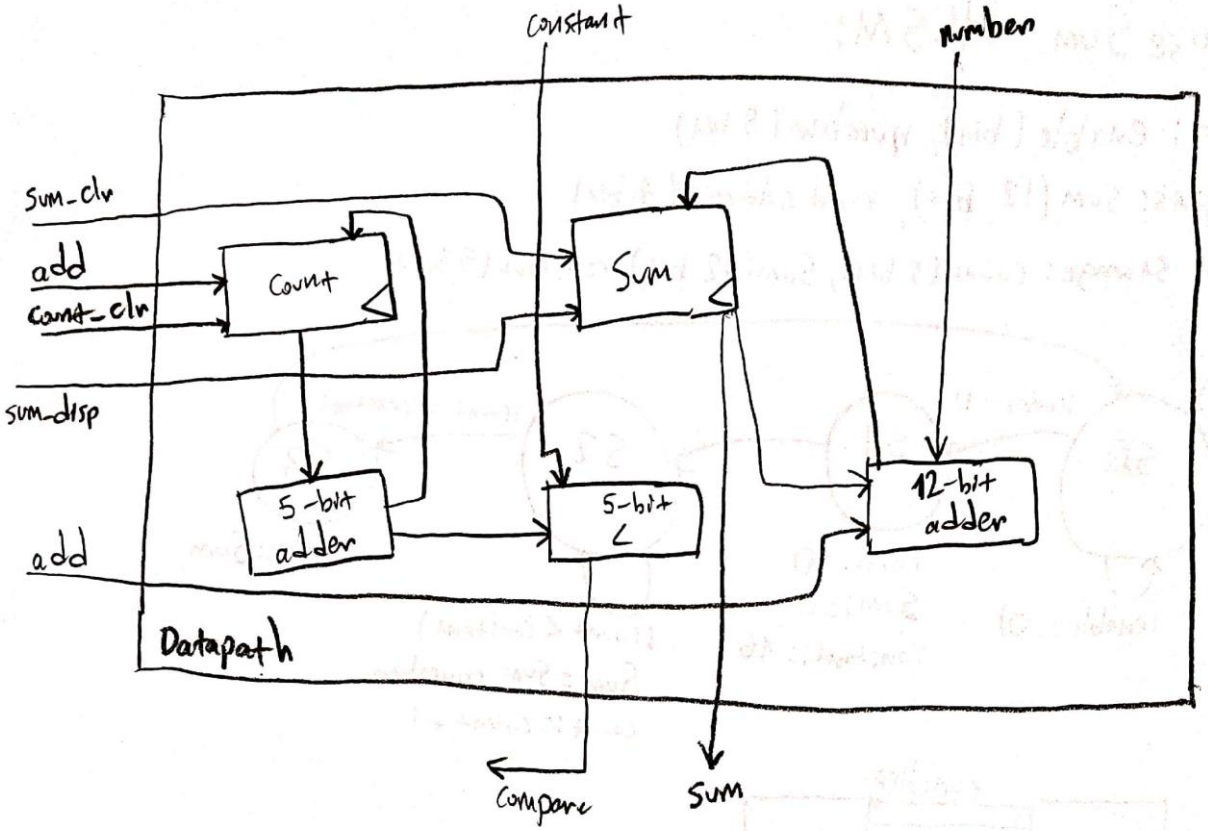
Outputs: sum (12 bit), read Address (4 bit)

Local storage: count (5 bit), Sum (12 bit), constant (5 bit)









b) You will be given a debouncer module for your pushbuttons. Research and explain briefly why there is a need for such a circuit:

A button normally outputs 1 when pressed, and outputs 0 when not pressed. However, since the button is a mechanical device, pressing it down may cause a small amount of bounce as the button settles in the down position. When it is first pressed, before outputting 1 constantly, the button first goes back and forth between 1 and 0 outputs. In order to prevent this, a debouncer is needed to preserve the output as 1 for an amount of time which is enough for the button to settle in its down position. This settling down usually takes about a few milliseconds, so the debouncer should hold the output of 1 for a small amount of time before the button settles.

c) Write the SystemVerilog Code for your memory module. Write a testbench for it:

Code for memory module:

```
`timescale 1ns / 1ps


module memory_module(input logic clk, writeEnable, input logic [3:0] writeAddress, input logic [7:0] writeData, input logic [3:0] readAddress1, input logic [3:0] readAddress2, output logic [7:0] readData1, output logic [7:0] readData2);


  logic [7:0] RAM[15:0]; // 16x8 memory module


  always_ff @(posedge clk)
  begin
    if ( writeEnable)
      RAM[writeAddress] <= writeData;
    end


  assign readData1 = RAM[readAddress1];
  assign readData2 = RAM[readAddress2];


endmodule
```

Testbench for memory module:

```
`timescale 1ns / 1ps


module memorymodule_testbench();
   logic clk, writeEnable;
   logic [3:0] writeAddress;
   logic [7:0] writeData;
   logic [3:0] readAddress1;
   logic [3:0] readAddress2;
   logic [7:0] readData1;
   logic [7:0] readData2;


   memory_module dut( clk, writeEnable, writeAddress, writeData, readAddress1, readAddress2, readData1, readData2);


   always
      begin
         clk = 1; #1;
         clk = 0; #1;
      end


   initial begin
      // writeEnable = 0; #5;
      writeEnable = 1;
      writeAddress = 4'b0;
      writeData = 8'b0;
      readAddress1 = 4'b0;
      readAddress2 = 4'b0;
      for ( int i = 4'b0; i < 5'b10000; i++) begin
```

```
        #10;
        writeAddress = i;
        if ( i != 4'b0)
            writeData = writeData + 1;

        readAddress1 = i;
        readAddress2 = i;
        #10;
    end
  end
endmodule
```

d) Write the SystemVerilog Code for your ReduceSum Module. Write a testbench for it:

Code for ReduceSum module:

```
`timescale 1ns / 1ps

module ReduceSum_module(input logic clk, enable, input logic [7:0] number, output logic [11:0] sum, output logic [3:0] readAddress );
    logic [4:0] count = 5'b0;
    localparam constant = 5'b10000;
    logic [11:0] Sum = 12'b0;
    typedef enum logic [1:0] {S0, S1, S2, S3} statetype;
    statetype [1:0] state, nextState;

    // register
    always_ff @(posedge clk)
    begin
```

```
    state <= nextState;
end

// state transition
always_ff @ (posedge clk)
case(state)
   S0:
   begin
      if (enable)
         nextState <= S1;
      else
         nextState <= S0;
   end

   S1:
   begin
      count <=  4'b0;
      Sum <= 12'b0;
      nextState <= S2;
   end

   S2:
   begin
      if (count != constant) begin
         Sum = Sum + number;
         count = count + 1;
         nextState <= S2;
```

```
            end
        else
            nextState <= S3;
    end

    S3:
    begin
        //Sum = Sum + number;
        sum = Sum;
        nextState <= S0;
    end
    default:
        nextState <= S0;
  endcase
  assign readAddress = count;
endmodule
```

Testbench for ReduceSum module:

```
module reducesum_testbench();
 logic clk, enable;
   logic [7:0] value;
   logic [11:0] sum;
   logic [3:0] address;

   ReduceSum_module dut( clk, enable, value, sum, address );

   always
```

```
    begin
        clk = 1; #20; clk = 0; #20;
    end

  initial begin
    enable = 0; #10; enable = 1; sum = 0; address = 0;
    value = 8'b00000001;
  end
endmodule
```

e) Write the SystemVerilog Code for the top design which will also include the Seven Segment Display and debouncer modules:

```
`timescale 1ns / 1ps

module TopModule( input logic clk, input logic [3:0] buttons, input logic [7:0] number,
  input logic [3:0] address, output logic [11:0] reduceSumSum, output logic [6:0] seg, output logic dp, output logic [3:0] an);

  logic [3:0] addressCounter;
  logic [3:0] reduceSumAddress;

  logic prev, next, enter, sum;

  logic [7:0] readData1, readData2;

  debounce previousButton(clk, buttons[3], prev);
  debounce nextButton(clk, buttons[2], next);
  debounce enterButton(clk, buttons[1], enter);
```

```
    debounce sumButton(clk, buttons[0], sum);


    always_ff @(posedge clk)
        begin
            if ( prev) begin
                if ( addressCounter == 4'b0000) begin
                    addressCounter <= 4'b1111;
                end
                else
                    addressCounter <= addressCounter - 1;
            end


            else if ( next) begin
                if ( addressCounter == 4'b1111)
                    addressCounter <= 4'b0000;
                else
                    addressCounter <= addressCounter + 1;
            end
        end


        memory_module mem( clk, enter, address, number, addressCounter, reduceSumAddress, readData1, readData2);
        SevSeg_4digit display( clk, addressCounter, 0, readData1[7:4], readData1[3:0], seg, dp, an);
        ReduceSum_module reduceSum(clk, sum, readData2, reduceSumSum, reduceSumAddress);
endmodule
```