CS 223
Digital Design
Section 01
Laboratory Assignment 5
Reduce Sum on Array

Arda Önal
21903350

12.11.2020

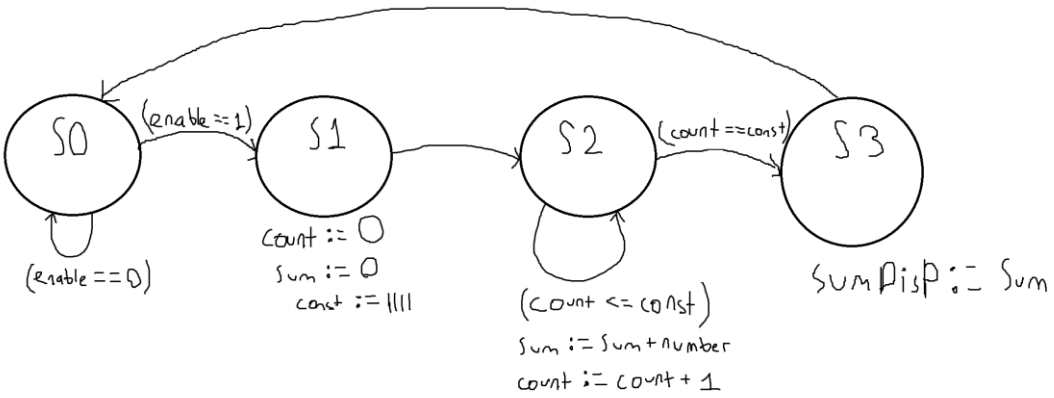
a) Show the ReduceSum HLSM. Show also the controller/datapath diagram for it.

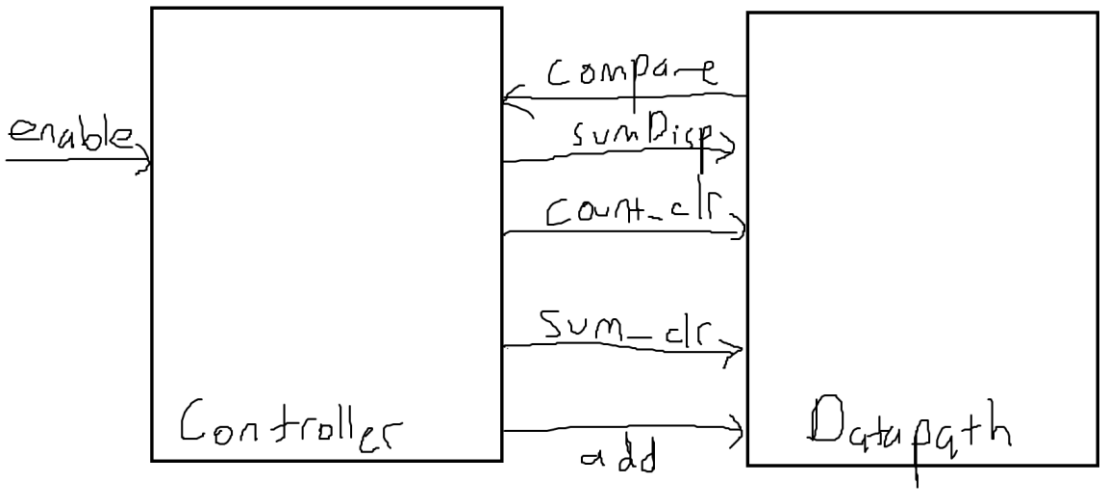
ReduceSum HLSM

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

Outputs: sumDisp (12bit), readAddress (4 bits)

Local Storage: count (4bits), Sum (12bits), const (5bits)

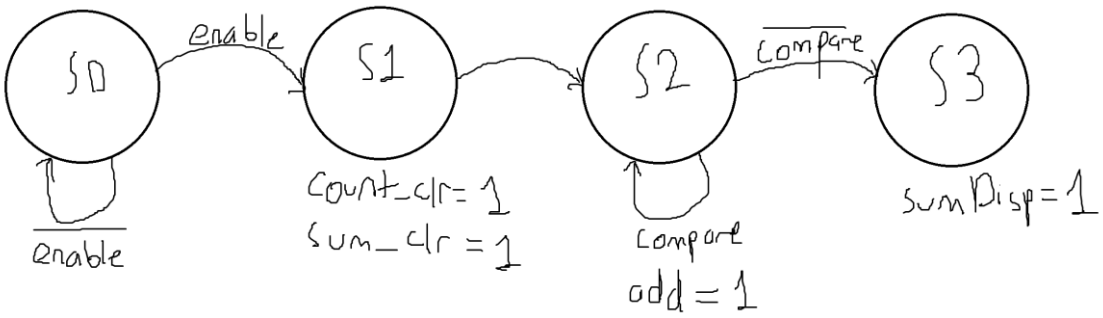




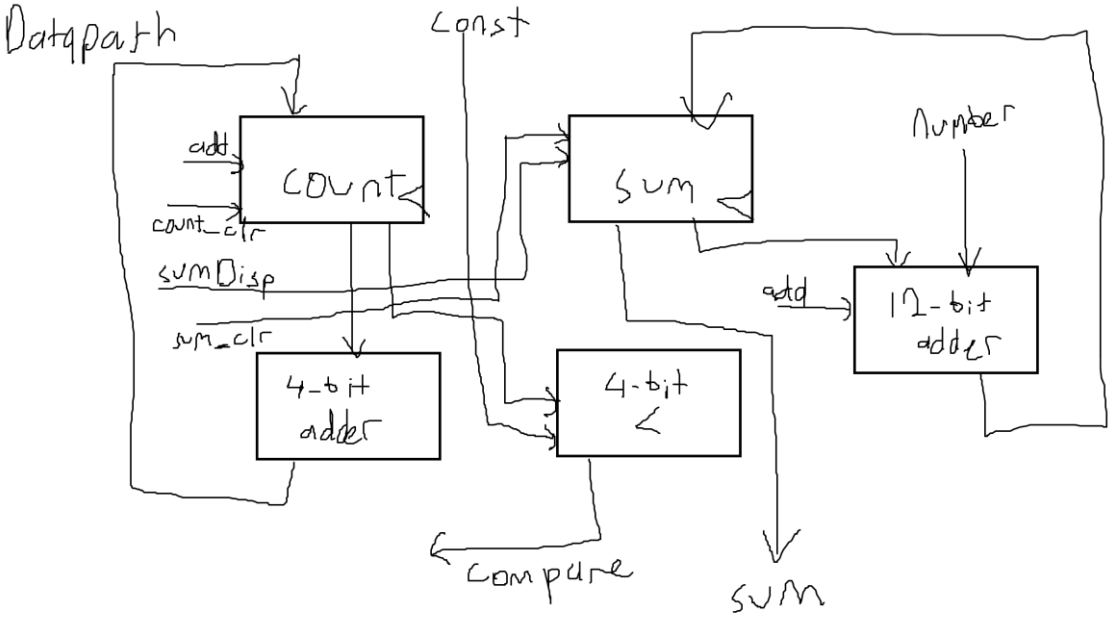
Controller

Inputs: enable, compare

Outputs: count_clr, sum_clr, add, sumDisp



Datapath



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

In BASYS3, if a button is being used as an input, the button bounces several rising edges will be generated instead of just one. Because of this, our code would not work properly and it is necessary to debounce the button. The debouncer module fixes this problem.

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

```
module memory_module(input logic clk,writeEnable,input logic [7:0] writeData,input logic [3:0] writeAdress, input logic [3:0] readAdress1, input logic [3:0] readAdress2,  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[writeAdress] <= writeData;
   end


   assign readData1 = RAM[readAdress1]; //Assigning the values in the speciced addresses to outputs
   assign readData2 = RAM[readAdress2];
endmodule
```

Testbench:

```
module testbench_memory();
   logic clk, writeEnable;
   logic [3:0] writeAdress;
   logic [7:0] writeData;
   logic [3:0] readAdress1;
   logic [3:0] readAdress2;
   logic [7:0] readData1;
   logic [7:0] readData2;
```

```
    memory_module dut(
clk,writeEnable,writeData,writeAdress,readAdress1,readAdress2,readData1,readData2);


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


    initial begin
        writeEnable = 1;
        writeAdress = 4'b0;
        writeData = 8'b0;
        readAdress1 = 4'b0;
        readAdress2 = 4'b0;
        for ( int i  = 4'b0; i <= 4'b1111; i++) begin
            #10;
            writeAdress = i;
            writeData = writeData + i;


            readAdress1 = i;
            readAdress2 = i;
            #10;
        end
     end
endmodule
```

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

```
module ReduceSum_module(input logic clock, enable, input logic [7:0] value, output logic [11:0] outputSum, output logic [3:0] address );

    logic [4:0] counter = 5'b00000; // Initializing

    logic [11:0] Sum = 12'b0; // Initializing, sum is 12 bits because if every 16 8 bit value is maximum we get a 12 bit value.


    typedef enum logic [1:0] {S0, S1, S2, S3} statetype;

    statetype [1:0] state, nextState;


    // register

    always_ff @(posedge clock)

    begin

     state <= nextState;

    end


    // state transition conditions

    always_ff @ (posedge clock)

    case(state)

      S0: // If enable is 1 it transitions to state1, otherwise remains in state0

      begin

        if (enable)

          nextState <= S1;

        else

          nextState <= S0;

      end

      S1: // Initialization state

      begin

        counter <=  5'b0;

        Sum <= 12'b0;

        nextState <= S2;
```

```
        end
        S2: // Counts until it is 15 and adds the value to the sum 15 times
        begin
            counter = counter + 1;
            Sum = Sum + value;
            if (counter != 5'b01111)// value of maximum count, remains its state until it gets to
15
                nextState <= S2;
            else
                nextState <= S3; // changes state when count is 15
        end
        S3:
        begin
            Sum = Sum + value; // We added our value 15 times but we actually want to add it
16 times
            outputSum = Sum; // Producing the output sum
            nextState <= S0; // Going back to state0
        end
        default:
            nextState<= S0;
    endcase


    assign address = counter; // Assigning output address
endmodule
```

Testbench:

```
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 = 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.

```
module combined_module( input logic clk, input logic [3:0] buttons, input logic [7:0] data,
input logic [3:0] address,
    output logic [11:0] sum, output logic [6:0] seg, output logic dp, output logic [3:0] an);


    logic [3:0] count1; // Counter for address 1
    logic [3:0] count2; // Counter for address 2


    logic Prev, Next, Enter, Sum; // Logics for four buttons


    logic [11:0] tmpSum;
    logic [7:0] readData1;
    logic [7:0] readData2;
    logic [11:0] summation;


    // Calling the debouncer methods for the buttons
    debounce prev(clk, buttons[3], Previous);
    debounce next(clk, buttons[2], Next);
```

```
    debounce enter(clk, buttons[1], Enter);
    debounce summ(clk, buttons[0], Sum);


    always_ff@ (posedge clk)
    begin
        if (Previous)
        begin
            if ( count1 != 0)
                count1 <= count1 - 1; // Decrementing the index of address by one.
            else
                count1 <= 4'b1111; // This is case for address = 0, It goes back to address = 15.
        end
        else if (Next)
        begin
            if ( count1 == 4'b1111)
                count1 <= 4'b0000; // This is case for address = 15, It goes to address = 0.
            else
                count1 <= count1 + 1; // Incrementing the index of address by one.
        end
        else if (Sum)
        begin
            tmpSum <= summation; // If the sum button is clicked, it assigns the new summation
to our inner variable.
        end
    end
    // Calling the methods.
    memory_module memory(clk, Enter,data,address,count1,count2,readData1,readData2);
    SevSeg_4digit displayer(clk,count1,0,readData1[7:4],readData1[3:0],seg,dp,an);
    ReduceSum_module sumModule(clk, Enter, readData2, summation, count2);
    assign sum = tmpSum; // Assigning the inner summation to the output
endmodule
```