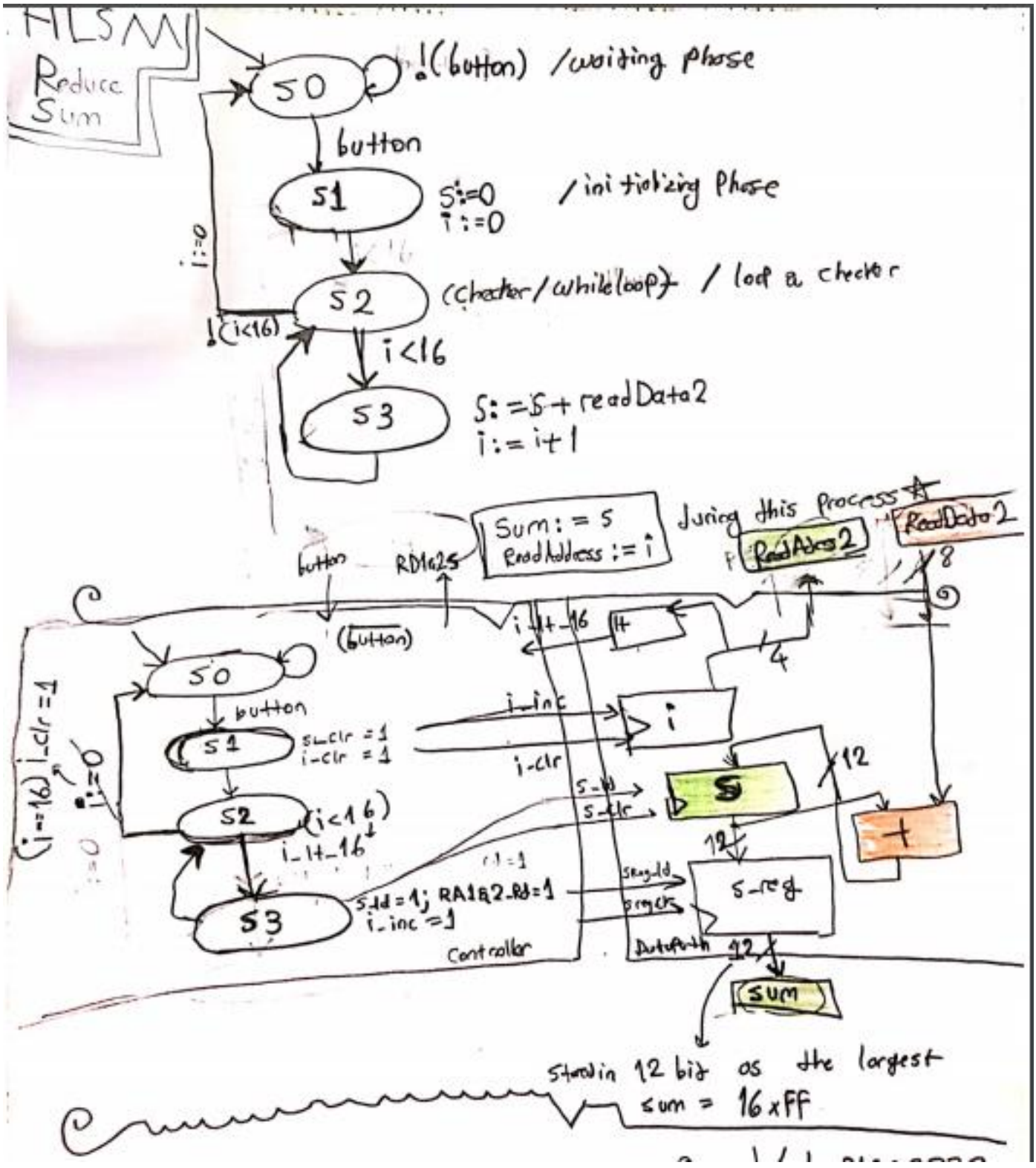
CS223 DIGITAL DESIGN
SECTION 3
Laboratory Assignment 5:
Reduce Sum on Array

Nurettin Onur Vural 21902330

13.12.2020

PRELIMINARY REPORT

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



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

When using switches, sending input signals may not cause big problems, howerver when using mechanical switches & buttons, probably there will be problems. Such mechanical systems tend to vibrate and produce severeal transitions; giving a sense of the motion of a bouncing ball which is why the module to correct this is named debouncer. When using FPGA, this debouncing of the buttons will cause undesired transitions on the seven segment display in our case. To prevent this, a module named "debouncer" is helpful as it waits for the transition stage that causes trouble (bouncing) and uses a counter to achieve this. In this respect, it removes the rebound signals and produces a stable pulse from them which can be integrated as the correct input signal coming from the button.

Source Links:

https://www.fpga4fun.com/Debouncer.html

https://www.eecs.umich.edu/courses/eecs270/270lab/270_docs/debounce.html

https://www.maximintegrated.com/en/glossary/definitions.mvp/term/debounce/gpk/82

https://www.beyond-circuits.com/wordpress/tutorial/tutorial11/

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

```
module memoryModule(
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] mem[15:0];

integer i;
initial
begin
for(i=0; i<16; i=i+1)
mem[i] <= 8'b0;
end

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

always_ff @ (posedge clk) begin
    if ((readAddress1==writeAddress)&& writeEnable) begin
        readData1 <= writeData; //since if we have read from mem[3], it
//would be the old data
        readData2 <= mem[readAddress2];
end
else if ((readAddress2==writeAddress)&& writeEnable) begin
        readData2 <= writeData; //since if we have read from mem[3], it
//would be the old data
```

```
        readData1 <= mem[readAddress1];
end
else begin
       readData1 <= mem[readAddress1] ;
       readData2 <= mem[readAddress2] ;
end
end

endmodule
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

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

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

always begin
  clk <= 1; #5;
  clk <= 0; #5;
 end

 initial begin
 writeEnable <= 0; writeAddress <= 4'b0000; readAddress1 <= 4'b0001;
readAddress2 <= 4'b0010; writeData <= 8'b0001_0110;  #10;
 writeEnable <= 1; #10;
 writeAddress <= 4'b0001; writeData <= 8'b0000_1100; #10;
 writeAddress <= 4'b0010; writeData <= 8'b0100_0000; #10;
 writeData <= 8'b0000_0001; #10;
 readAddress1 <= 4'b0000; #10;
 //readAddress2 <= 4'b0000; #10;
 writeData <= 8'b1111_1111; #10;
 writeAddress <= 4'b0000; #10;
 end

endmodule
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

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

```
module reduceSum(input logic clk, button, input logic [7:0] readData2,
output logic [3:0] readAddress2, output logic [11:0] sum);

typedef enum logic [2:0] {S0,S1,S2,S3} statetype;
statetype [2:0] state, nextstate;

integer s;
integer i;

// state register
always_ff @ (posedge clk)
begin
    if(button)
     state <= S0;
    else
```

```
     state <= nextstate;
end

// next state logic
always_ff @( posedge clk)
  case (state)
   S0: begin //WAITING FOR BUTTON
     if (button)
           nextstate = S1;
     else
          nextstate = S0;
     end
   S1: begin //INITIALIZING
         s = 0;
         i = 0;
         nextstate = S2;
         end
   S2: begin //LOOP CONTROL
     if ( i <= 15 )
        nextstate = S3;
     else
        begin
        i = 0;
        nextstate = S0;
        end
     end
   S3: begin //ADDITION
       s = s + readData2;
       i = i + 1;
       nextstate = S2;
       end
    default: nextstate = S0;
   endcase

 assign sum = s;
 assign readAddress2 = i;

endmodule
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

module reduceSumTestBench();
logic clk,  button, writeEnable;
logic [3:0] writeAddress, readAddress1 ,readAddress2;
logic [7:0] writeData, readData1, readData2;
logic [11:0] sum;

memoryModule m( clk, writeEnable, writeAddress, readAddress1,
readAddress2, writeData, readData1, readData2);
reduceSum dut(clk, button, readData2, readAddress2, sum);



always begin
  clk <= 1; #5;
  clk <= 0; #5;
 end
```

```
 initial begin


 writeEnable <= 1;

 //readAddress2 <= 4'b0000;
 readAddress1 <= 4'b0001; #10;
 writeAddress <= 4'b0000; writeData <= 8'b0000_0001; #10;
 writeAddress <= 4'b0001; writeData <= 8'b0000_0010; #10;
 writeAddress <= 4'b0011; writeData <= 8'b0000_0011; #10;
 writeAddress <= 4'b0100; writeData <= 8'b1111_1111; #10;


 button <= 1; #10;
 button <= 0; #10;

 end
endmodule
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

e) Write the SystemVerilog Code for the top design which will also include the Seven

Segment Display and debouncer modules.

```
module debounce(input logic clk, input logic button, output logic pulse
);

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(button)
            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 pulse = ( state == S1 );
endmodule
```

```
module SevSeg_4digit(
 input clk,
 input [3:0] in3, in2, in1, in0, //user inputs for each digit
(hexadecimal value)
 output [6:0]seg, logic dp, // just connect them to FPGA pins
(individual LEDs).
 output [3:0] an   // just connect them to FPGA pins (enable vector for
4 digits active low)
 );

// divide system clock (100Mhz for Basys3) by 2^N using a counter,
which allows us to multiplex at lower speed
localparam N = 18;
logic [N-1:0] count = {N{1'b0}}; //initial value
always@ (posedge clk)
       count <= count + 1;


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

always@ (*)
 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 :  //select first 7Seg.
    begin
     digit_val = {1'b0, in0};
     digit_en = 4'b1110;
    end

   2'b01:  //select second 7Seg.
    begin
     digit_val = {1'b0, in1};
     digit_en = 4'b1101;
    end

   2'b10:  //select third 7Seg.
    begin
     digit_val = {1'b1, in2};
     digit_en = 4'b1011;
    end

   2'b11:  //select forth 7Seg.
    begin
     digit_val = {1'b0, in3};
     digit_en = 4'b0111;
    end
  endcase
 end


//Convert digit number to LED vector. LEDs are active low.
logic [6:0] sseg_LEDs;
```

```
always @(*)
 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'b0110111; //to display "="
   default : sseg_LEDs = 7'b0111111; //dash
  endcase
 end

assign an = digit_en;
assign seg = sseg_LEDs;
assign dp = 1'b1; //turn dp off
endmodule

module topDesign( input logic clk, writeEnableButtonO, displayPrevO,
displayNextO, giveSumO,
input logic [3:0] writeAddress, input logic [7:0] writeData, output
logic [11:0] sum,  output [6:0]seg, logic dp, // just connect them to
FPGA pins (individual LEDs).
output [3:0] an  );

logic [3:0] readAddress1, readAddress2;
logic [7:0] readData1, readData2;
logic writeEnableButton, displayPrev, displayNext, giveSum; //input
buttons to be updated
//logic [11:0] s, tempS;

debounce db1( clk, writeEnableButtonO, writeEnableButton );
debounce db2( clk, displayPrevO, displayPrev );
debounce db3( clk, displayNextO, displayNext );
debounce db4( clk, giveSumO, giveSum );

always_ff @ (posedge clk)
begin
 if (displayPrev)
 begin
     if (readAddress1 == 4'b0000)
        readAddress1 <= 4'b1111;
     else
        readAddress1 <= readAddress1 - 1;
 end
else if(displayNext)
```

```
begin
     if (readAddress1 ==  4'b1111)
        readAddress1 <= 4'b0000;
     else
        readAddress1 <= readAddress1 + 1; //readAddress1 <=
readAddress1 + 1;
 end;
end

memoryModule mM(clk, writeEnableButton, writeAddress, readAddress1,
readAddress2, writeData, readData1, readData2);
SevSeg_4digit sS(clk, readAddress1, 4'b0000, readData1[7:4],
readData1[3:0], seg, dp, an );
reduceSum rS(clk, giveSum, readData2, readAddress2, sum);

endmodule
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```