CS223 Digital Design

Lab05

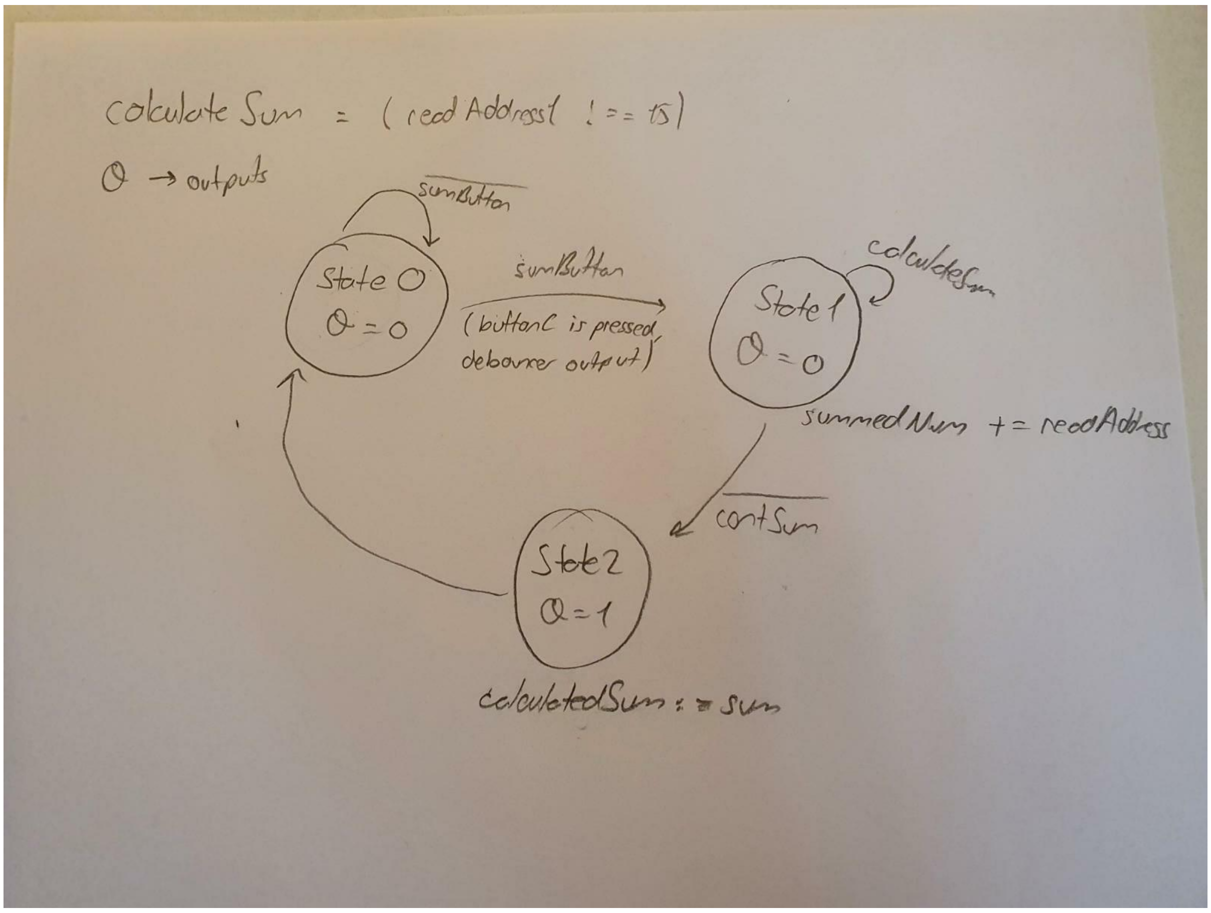
Preliminary Report

Arda İçöz

21901443

12/12/2020

Task A:


calculate Sum = (readAddress1 !== 15)
Q → outputs
$\overline{\text{sumButton}}$
State 0
Q = 0
sumButton
(buttonC is pressed, debouncer output)
calculateSum
State 1
Q = 0
summedNum += readAddress
$\overline{\text{contSum}}$
State2
Q = 1
calculatedSum := sum

Task B:

Mechanical inputs usually have unreliable inputs, that is, they may not create one clean pulse when pressed. Debouncer turns these glitched outputs onto a clean outputs.

Task C:

```
module memory(
   input logic clock,
   input logic 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

   );


   //defining a memory capable of storing 8-bit elements with size 16

   logic [7:0] mem[15:0];


   //writing the data to the memory location if writing is enabled

   always_ff @(posedge clock)

   begin

     if (writeEnable)

         mem[writeAddress] <= writeData;

   end


   //assigning outputs

   assign readData1 = mem[readAddress1];

   assign readData2 = mem[readAddress2];

endmodule


module memory_TB();


logic clock, writeEnable;

logic[3:0] writeAddress, readAddress1, readAddress2;
```

```
logic[7:0] writeData;

logic[7:0] readData1, readData2;

memory dut(clock, writeEnable, writeAddress, readAddress1, readAddress2,
writeData, readData1, readData2);

always begin
   clock = 0;
      forever #10 clock = ~clock;
end

initial begin
   writeEnable = 0; writeData = 8'b00000111; writeAddress = 4'b0000; #20;
readAddress1 = 4'b0101; readAddress2 = 4'b0010; #20;
   writeEnable = 0; writeData = 8'b00000111; writeAddress = 4'b0101; #20;
   writeEnable = 1; writeData = 8'b00000111; writeAddress = 4'b0101; #20;
   writeEnable = 0; writeData = 8'b00001010; writeAddress = 4'b0010; #20;
   writeEnable = 1; writeData = 8'b00001010; writeAddress = 4'b0010; #20;
end
endmodule
```

Task D:

```
module reduceSum(
   input logic clock,
   input logic sumButton,
   input logic calculateSum,
   input logic [7:0] readData,
   output logic [11:0] calculatedSum
   );

   typedef enum logic [1:0] {State0, State1, State2} statetype;
   statetype currentState, nextState;

   logic [11:0] summedNum;

   always_ff @(posedge clock)
   begin
      currentState <= nextState;
   end

   always_comb
      case( currentState )
         State0: begin
            summedNum = 12'b0;
            if( sumButton ) nextState = State1;
            else            nextState = State0;
         end
         State1: begin
            if ( sumButton ) summedNum <= summedNum + readData;
            else             nextState = State2;
         end
```

```
          State2: begin
             calculatedSum <= summedNum;
             nextState = State0;
          end
          default:
             nextState = State0;
       endcase
endmodule

module reduceSum_TB();
         logic clock;
         logic add;
         logic reset;
         logic display;
         logic [7:0] data;
         logic [11:0] sum;

         reducesum dut( clock, data, add, reset, display, sum );

         always begin
                  clock = 0;
                  forever #10 clk = ~clk;
         end

         initial begin
                  reset = 1; #20; reset = 0; display = 0; add = 1; data = 8'b00000100;
#20;
```

```
                for (int i = 0; i < 11; i++)
        #20;
                add = 0; display = 1; #100;
         end
endmodule
```

Task E:

```
module topDesign(
   input logic clock,
   input logic reset,
   input logic [11:0] switches,
   input logic buttonU,
   input logic buttonR,
   input logic buttonL,
   input logic buttonC,
   output logic [6:0] sevSegDisplay,
   output logic dp,
   output logic [3:0] an,
   output logic [11:0] leds
   );

   logic [3:0] readAddress1;
   logic [3:0] readAddress2;
   logic writeEnable;
   logic sumButton;
   logic calculateSum;
```

```
  logic [3:0] writeAddress;
  logic [7:0] writeData;
  logic [7:0] readData1;
  logic [7:0] readData2;

  typedef enum logic [1:0] {State0, State1, State2, State3, State4} statetype;
  statetype currentState, nextState;

  Controller topDesignController( clock, reset, buttonC, buttonR, buttonL,
buttonU, outputButtonC, outputButtonR, outputButtonL, outputButtonU );

  always_comb
    case( currentState )
      State0: begin
        if ( outputButtonU )      nextState = State1;
        else if ( outputButtonR ) nextState = State2;
        else if ( outputButtonL ) nextState = State3;
        else if ( outputButtonC ) begin
          calculateSum = 1;
          readAddress1 = 0;
          nextState = State4;
        end
        writeEnable = 0;
        sumButton = 0;
        if ( calculateSum ) nextState = State4;
      end
      State1: begin
```

```
            writeEnable = 1;
            nextState = State0;
        end
        State2: begin
            if ( readAddress2 == 15 ) readAddress2 = 0;
            else readAddress2++;
            nextState = State0;
        end
        State3: begin
            if ( readAddress2 == 0 ) readAddress2 = 15;
            else readAddress2--;
            nextState = State0;
        end
        State4: begin
            sumButton = 1;
            readAddress1++;
            calculateSum = ( readAddress1 == 15 );
        end
    endcase


    memory memoryRam( clock, writeEnable, readAddress1, writeData,
readAddress1, readAddress2, readData1, readData2 );
    SevSeg_4digit sevSeg( clock, readAddress1, 1'dx, readData1[7:4],
readData1[3:0], sevSegDisplay, dp, an );
    reduceSum reduce_sum( clock, sumButton, calculateSum, readData1,
calculatedSum );
```

```
endmodule
```