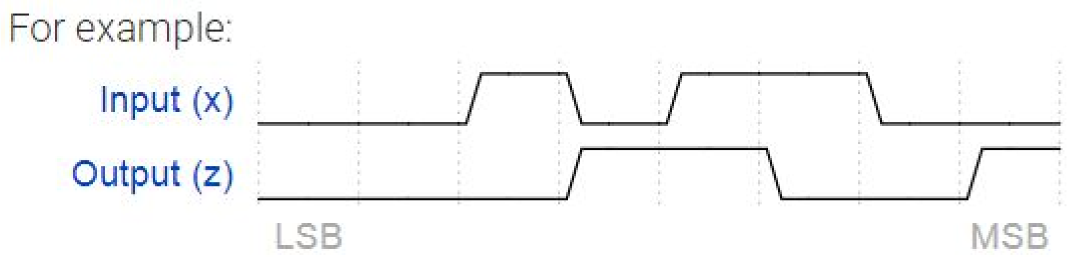
Exams/ece241 2014 q5a

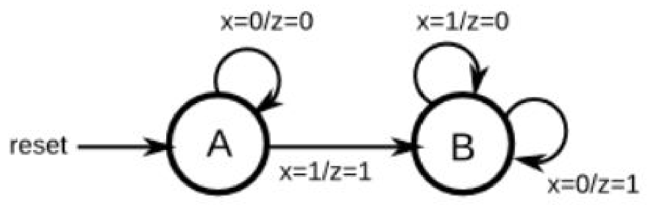
You are to design a one-input one-output serial 2's complementer **Moore** state machine. The input (x) is a series of bits (one per clock cycle) beginning with the least-significant bit of the number, and the output (Z) is the 2's complement of the input. The machine will accept input numbers of arbitrary length. The circuit requires an asynchronous reset. The conversion begins when *Reset* is released and stops when *Reset* is asserted.

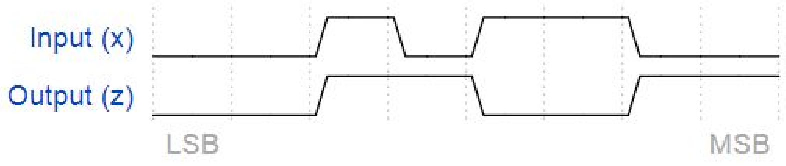


```
module top_module (
    input clk,
    input areset,
    input x,
    output z
);
//2's complement:Moore = invert all bits then add 1
    //              = LSB should be treated differently
    //                not until you found the first non-zero bit , the output remains 0  StateLSB
    //                first input 1 => ouput 1
    //                then everything you  have to do is invert
    //              example:  0000_1101_00
    //          its 2's comp: 1111_ 0011_00

    parameter [1:0] WAIT_1 = 2'b00;
    parameter [1:0] FIRST_1= 2'b01;
    parameter [1:0] RECV_0 = 2'b10;
    parameter [1:0] RECV_1 = 2'b11;


    reg [1:0] curr_state;
    reg [1:0] next_state;

    always @ (posedge clk or posedge areset)begin
        if(areset)begin
            curr_state <= WAIT_1;
        end
        else begin
            curr_state <= next_state;
        end
    end

    always @ (*)begin
        case(curr_state)
            WAIT_1 : next_state = x? FIRST_1 : WAIT_1;
            FIRST_1: next_state = x? RECV_1  : RECV_0; //first 1     output 1
            RECV_1 : next_state = x? RECV_1  : RECV_0; //recieving 1 output 0
            RECV_0 : next_state = x? RECV_1  : RECV_0; //recieving 0 output 1
            default: next_state = curr_state;
        endcase
    end
    assign z = (curr_state == FIRST_1)|(curr_state==RECV_0);

endmodule
```

Exams/ece241 2014 q5b

The following diagram is a **Mealy** machine implementation of the 2's complementer. Implement using one-hot encoding.





```
module top_module (
   input clk,
   input areset,
   input x,
   output z
);
   parameter A = 1'b0;
   parameter B = 1'b1;

   reg curr_state;
   reg next_state;

   always @ (posedge clk or posedge areset)begin
      if(areset)begin
         curr_state<= A;
      end
      else begin
         curr_state <= next_state;
      end
   end

   always @ (*)begin
      case(curr_state)
         A: if (x==0) begin
            next_state = A;
                   z = 1'b0;
         end
         else begin
            next_state = B;
                   z = 1'b1;
         end
         B: if (x==1) begin
            next_state = B;
                   z = 1'b0;
         end
         else begin
            next_state = B;
                   z = 1'b1;
         end
```

```
        endcase
    end

endmodule
```