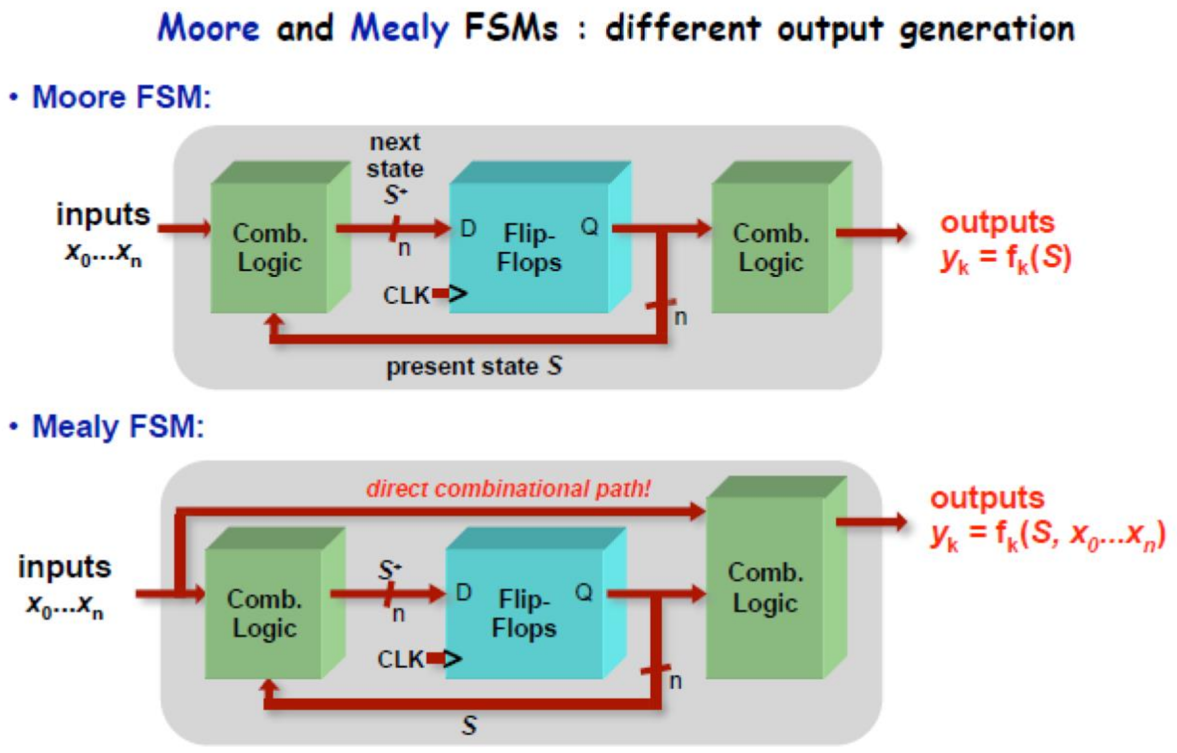
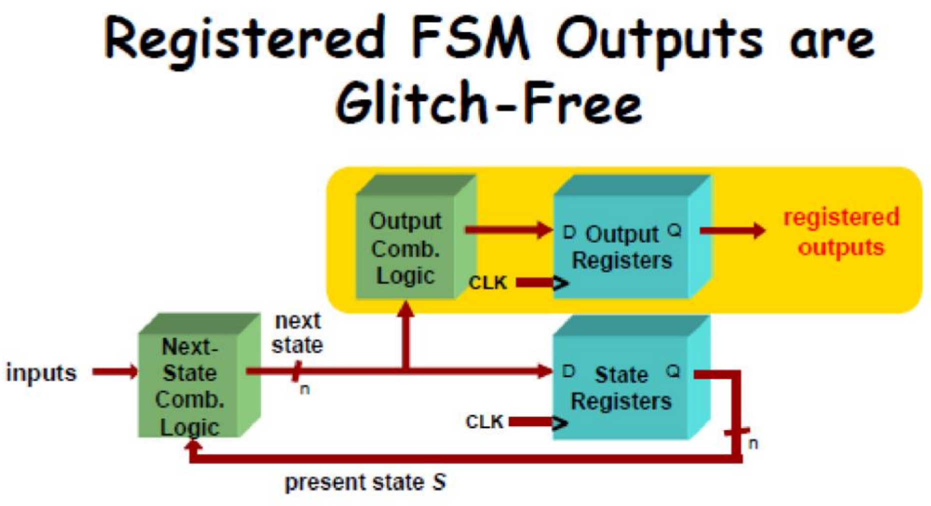
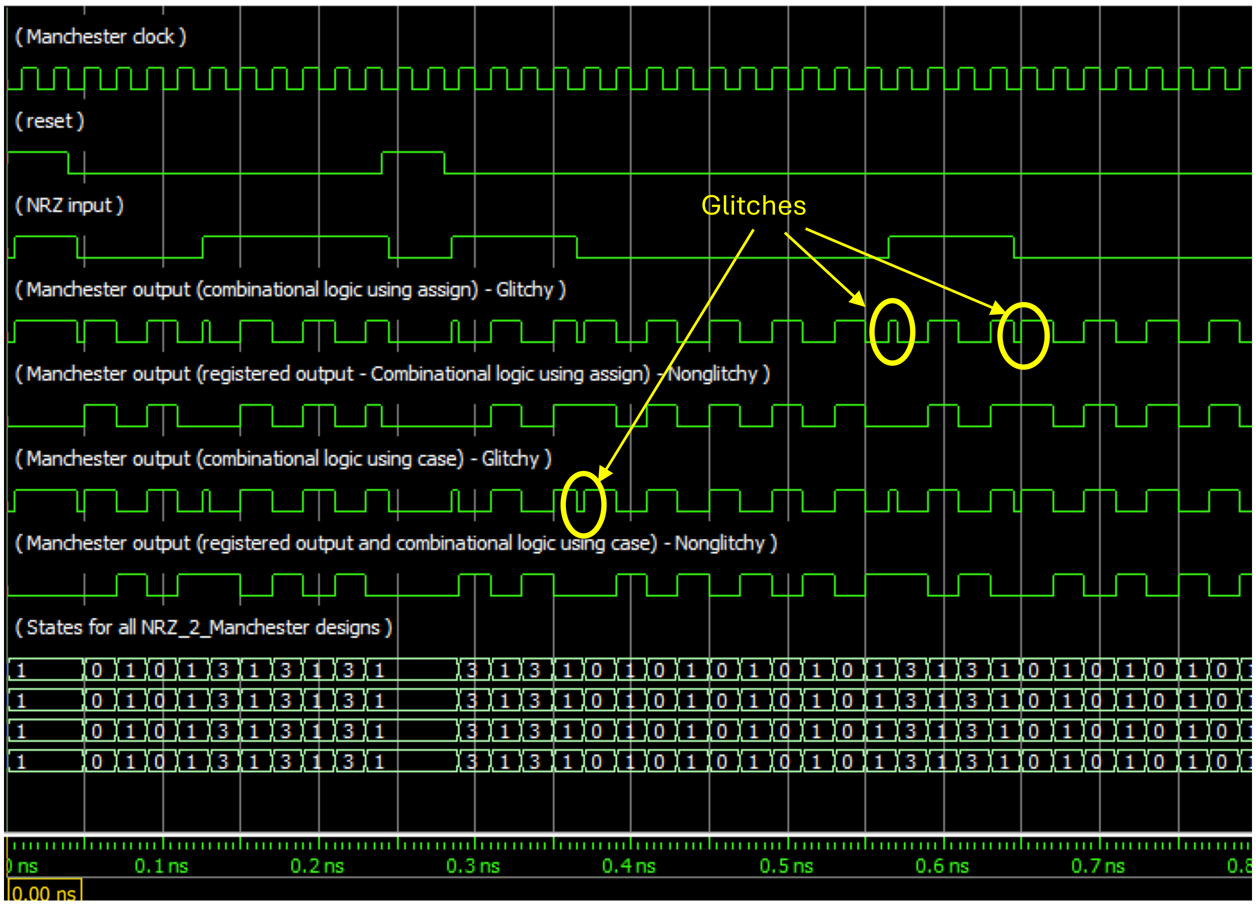
NRZ-Manchester Code Conversions

This module compares different designs for an NRZ to Manchester code converter. Two designs might cause glitches at the output if the input does not align changes with the state levels. This occurs when the output is designed using combinational logic depending on the input and the current state. The combinational logic was designed using the assign operator and the switching function was found using the K-map, and the other design uses a case structure. In the second design it a case structure was used to define the combinational logic. To obtain a glitch free output, a register was added at the output. The combinational logic finds next_out and the value is passed to the output only at the positive clock edge.





```
module NRZ_2_Manchester_Mealy (NRZ_in,
                              clock,
                              reset,
                              Manchester_assign,
                              Manchester_case,
                              Manchester_reg_assign,
                              Manchester_reg_case);

 input   NRZ_in, reset, clock;
 output  Manchester_assign;
 output  Manchester_case;
 output  Manchester_reg_case;
 output  Manchester_reg_assign;

 NRZ_2_Manchester_Mealy_glitchy_assign        M1  (Manchester_assign,
                                                   NRZ_in,
                                                   clock, reset);

 NRZ_2_Manchester_Mealy_nonglitchy_assign     M2  (Manchester_reg_assign,
                                                   NRZ_in,
                                                   clock, reset);

 NRZ_2_Manchester_Mealy_case                  M3  (Manchester_case,
                                                   NRZ_in,
                                                   clock, reset);

 NRZ_2_Manchester_Mealy_case_nonglitchy       M4  (Manchester_reg_case,
                                                   NRZ_in,
                                                   clock, reset);

 endmodule
```



```verilog
module NRZ_2_Manchester_Mealy_glitchy_assign (Manchester_out,
                                              NRZ_in,
                                              clock, reset);

output Manchester_out;
input  NRZ_in, clock, reset;

reg   [1:0] state;
wire  [1:0] next_state;

parameter Sx = 2'b01;     // waiting for a new NRZ input
parameter S0 = 2'b00;     // An NRZ 0 is being converted to 01
parameter S1 = 2'b11;     // An NRZ 1 is being converted to 10

// Sequential logic updating the state

always @ (posedge clock or posedge reset)     // asyncronous reset
   if (reset) state <= Sx;
   else     state <= next_state;

// Combinational logic to find the next_state and the Manchester_out

assign next_state [0] = NRZ_in | (~state[1] & ~state[0]);
assign next_state [1] = ~state[1] & NRZ_in;
assign Manchester_out = ~state[0] | (NRZ_in & ~state[1]);

endmodule
```

```
module NRZ_2_Manchester_Mealy_case         (Manchester_out,
                                            NRZ_in,
                                            clock, reset);

output Manchester_out;
input  NRZ_in, clock, reset;

reg [1:0] state, next_state;
reg       Manchester_out;       // to assign it values within always block

parameter Sx = 2'b01;    // waiting for a new NRZ input
parameter S0 = 2'b00;    // An NRZ 0 is being converted to 01
parameter S1 = 2'b11;    // An NRZ 1 is being converted to 10

// Sequential logic updating the state

always @ (posedge clock or posedge reset)    // asyncronous reset
   if (reset) state <= Sx;
      else    state <= next_state;


// Combinational logic to find the next_state and the Manchester_out

always @ *    // if the state or the NRZ_in change

   case (state)

       Sx : if(NRZ_in) begin
                       next_state = S1;
                       Manchester_out = 1'b1; end
               else    begin
                       next_state = S0;
                       Manchester_out = 1'b0; end

       S0 :    begin   next_state = Sx;              // NRZ_in has to be 0
                       Manchester_out = 1'b1; end

       S1 :    begin   next_state = Sx;              // NRZ_in has to be 1
                       Manchester_out = 1'b0; end

   default:    begin   next_state = Sx;
                       Manchester_out = 1'b0; end

   endcase

endmodule
```

```
module NRZ_2_Manchester_Mealy_nonglitchy_assign (Manchester_out,
                                                  NRZ_in,
                                                  clock, reset);
output reg Manchester_out;
input      NRZ_in, clock, reset;

reg   [1:0] state;
wire  [1:0] next_state;
wire        next_out;

parameter Sx = 2'b01;    // waiting for a new NRZ input
parameter S0 = 2'b00;    // An NRZ 0 is being converted to 01
parameter S1 = 2'b11;    // An NRZ 1 is being converted to 10

// Sequential logic updating the state

always @ (posedge clock or posedge reset)    // asyncronous reset
   if (reset) begin state <= Sx;
                    Manchester_out <= 1'b0;      end
   else       begin state <= next_state;
                    Manchester_out <= next_out; end

// Combinational logic to find the next_state and the Manchester_out

assign next_state [0] = NRZ_in | (~state[1] & ~state[0]);
assign next_state [1] = ~state[1] & NRZ_in;
assign next_out = ~next_state[0] | (NRZ_in & ~next_state[1]);

endmodule
```

```
module NRZ_2_Manchester_Mealy_case_nonglitchy     (Manchester_out,
                                                   NRZ_in,
                                                   clock, reset);
output Manchester_out;
input  NRZ_in, clock, reset;

reg [1:0] state, next_state;
reg       next_out, Manchester_out; // to assign values within always block

parameter Sx = 2'b01;    // waiting for a new NRZ input
parameter S0 = 2'b00;    // An NRZ 0 is being converted to 01
parameter S1 = 2'b11;    // An NRZ 1 is being converted to 10

// Sequential logic updating the state

always @ (posedge clock or posedge reset)    // asyncronous reset
   if (reset) begin state <= Sx;
                    Manchester_out <= 1'b0; end
      else    begin state <= next_state;
                    Manchester_out <= next_out; end


 // Combinational logic to find the next_state and the Manchester_out

 always @ *    // if the state or the NRZ_in change

    case (state)

        Sx : if(NRZ_in) begin
                        next_state = S1;
                        next_out = 1'b1; end
                else    begin
                        next_state = S0;
                        next_out = 1'b0; end

        S0 :    begin   next_state = Sx;                // NRZ_in has to be 0
                        next_out = 1'b1; end

        S1 :    begin   next_state = Sx;                // NRZ_in has to be 1
                        next_out = 1'b0; end

    default:    begin   next_state = Sx;
                        next_out = 1'b0; end

    endcase

 endmodule
```