```
/*EE417 Lesson 5 A1 -Manchester code to PAM4 code converter
Name: Ron Kalin, Date: 6/11/24
Group: Kalin, Jammeh
PAM4 combines two NRZ bits, which means 4 Manchester bits.
Manchester 0101 = 00 NRZ = 0 PAM4
Manchester 0110 = 01 NRZ = 1 PAM4
Manchester 1001 = 10 NRZ = 2 PAM4
Manchester 1010 = 11 NRZ = 3 PAM4
*/
/*module manchester_to_pam4 (  //mealy, top level module
    input wire clk, // Clock for sampling
    input wire rst, // Reset
    input wire manchester_in, // Manchester-encoded serial input
    output reg [2:0] pam4_out // 3-bit PAM4 output
);
    reg [1:0] state; // State machine for decoding
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            state <= 2'b00; // Initial state
            pam4_out <= 3'b000; // Reset output
        end else begin
            case (state)
                2'b00: begin // Waiting for rising edge
                    if (manchester_in) state <= 2'b01;
                end
                2'b01: begin // Rising edge detected
                    state <= 2'b10;
                end
                2'b10: begin // Falling edge detected
                    state <= 2'b00;
                    // Map Manchester data to PAM4 levels
                    case (manchester_in)
                        1'b0: pam4_out <= 3'b001; // -1
                        1'b1: pam4_out <= 3'b010; // 0
                    endcase
                end
            endcase
        end
    end
endmodule


module manchester_to_PAM_mealy_assign_glitchy(
       output PAM_out,
       input manchester_in, input clock, input reset);
//this module will show glitchy mealy FSM conversion from manchester to PAM

reg  [2:0] state; //number of state bits required = number of states in state diagram
wire [2:0] next_state;// ex: 5 to 8 states = 3 bits

//parameter Sx = 2'b00;  //waiting for new manchester input
//parameter S0 = 2'b01;  //manchester 0 is being converted to 01
//parameter S1 = 2'b10;  //manchester 1 is being converted to 10

parameter S0    = 4'b0000; //waiting for new manchester input
parameter S00   = 4'b0000;
parameter S01   = 4'b0001;
parameter S001  = 4'b0001;
parameter S010  = 4'b0010;
parameter S0010 = 4'b0010;
parameter S0011 = 4'b0011;
parameter S0100 = 4'b0100;
parameter S0101 = 4'b0101;

parameter S1    = 4'b0000; //waiting for new manchester input
parameter S10   = 4'b0000;
parameter S11   = 4'b0001;
parameter S101  = 4'b0010;
parameter S110  = 4'b0010;
parameter S1010 = 4'b0011;
```

```
  parameter S1011 = 4'b0100;
  parameter S1100 = 4'b1100;
  parameter S1101 = 4'b1101;

  parameter S2    = 4'b0010; //waiting for new manchester input
  parameter S20   = 4'b0010;
  parameter S21   = 4'b0011;
  parameter S201  = 4'b0101;
  parameter S210  = 4'b0110;
  parameter S2010 = 4'b1010;
  parameter S2011 = 4'b1011;
  parameter S2100 = 4'b1100;
  parameter S2101 = 4'b1101;

  parameter S3    = 4'b0011; //waiting for new manchester input
  parameter S30   = 4'b0011;
  parameter S31   = 4'b0011;
  parameter S301  = 4'b0101;
  parameter S310  = 4'b0110;
  parameter S3010 = 4'b1010;
  parameter S3011 = 4'b1011;
  parameter S3100 = 4'b1100;
  parameter S3101 = 4'b1101;

  // Sequential logic updating the state

  always @ (posedge clock or posedge reset)
    if (reset) state <= S0;
    else state <= next_state;

  // Combinational logice to find next_state and PAM_out

  assign next_state [0] = manchester_in | (~state[1] & ~ state[0]);
  assign next_state [1] = ~state[1] & manchester_in;
  assign PAM_out = ~state[0] | (manchester_in & state[1]);


  endmodule


  module manchester_to_PAM_Mealy_case_glitchy(PAM_out,
                                              manchester_in,
                                               clock, reset);
  output PAM_out;
  input manchester_in, clock, reset;

  reg  [2:0] state, next_state; //number of state bits required = number of states in state
  diagram
  reg PAM_out;                   // ex: 5 to 8 states = 3 bits, 9 to 16 states=4bits
    //to assign it values within always block

  parameter Sx = 2'b01;  //waiting for new manchester input
  parameter S0 = 2'b01;  //manchester 0 is being converter to 01
  parameter S1 = 2'b11;  //manchester 1 is being converter to 10

  // Sequential logic updating the state

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

  // Combinational logic to find next_state and PAM_out

  always @ *  //if state or manchester_in change
    case (state)
      Sx : if(manchester_in) begin
                             next_state = S1;
                              PAM_out = 3'b001; end
              else           begin
                             next_state = S0;
```

```
140                                PAM_out = 3'b000; end
141
142       S0 :    begin           next_state = Sx;  //machnester_in has to be 0
143                                PAM_out = 3'b001; end
144
145       S1 :    begin           next_state = Sx;  //machnester_in has to be 1
146                                PAM_out = 3'b000; end
147
148    default :   begin           next_state = Sx;  //default case
149                                PAM_out = 3'b000; end
150      endcase
151
152    endmodule
153
154
155    module manchester_to_PAM_Mealy_assign_nonglitchy (PAM_out,
156                                                      manchester_in,
157                                                       clock, reset);
158    output reg PAM_out;
159    input manchester_in, clock, reset;
160
161    reg [2:0] state; //the use of register assures no glitches
162    wire [2:0] next_state;
163    wire       next_out;
164
165    parameter Sx = 2'b01; //waiting for new manchester input
166    parameter S0 = 2'b00; // manchester 0 is being converted to 01
167    parameter S1 = 2'b00; // manchester 1 is being converted to 10
168
169    // sequential logic updating the state
170
171    always @ (posedge clock or posedge reset)  //asynchronous reset
172      if (reset) begin state <= Sx;
173                       PAM_out <= 3'b000;  end
174      else       begin state <= next_state;
175                       PAM_out <= next_out; end
176
177    // combinational logic to find next_state and PAM_out
178
179    assign next_state[0] = manchester_in | (~state[1] & ~state[0]);
180    assign next_state[1] = ~state[1] & manchester_in;
181    assign next_out = ~next_state[0] | (manchester_in & ~next_state[1]);
182
183    endmodule
184
185
186    module manchester_to_PAM_Mealy_case_nonglitchy (PAM_out,
187                                                    manchester_in,
188                                                     clock, reset);
189    output PAM_out;
190    input manchester_in, clock, reset;
191
192    reg [2:0] state, next_state;
193    reg       next_out, PAM_out; // assign values within always block
194
195    parameter Sx = 2'b01; //waiting for new manchester input
196    parameter S0 = 2'b00; // manchester 0 is being converted to 01
197    parameter S1 = 2'b00; // manchester 1 is being converted to 10
198
199    // Sequential logic updating the state
200    always @ (posedge clock or posedge reset) //asynchronous reset
201      if (reset) begin state <= Sx;
202                       PAM_out <= 1'b0; end
203      else begin state <= next_state;
204                 PAM_out <= next_out; end
205
206    // Combinational logice to find next_state and PAM_out
207    always @ *  //if state or manchester_in change
208      case (state)
209        Sx : if(manchester_in) begin
```

```
                                next_state = S1;
                                 PAM_out = 3'b001; end
                    else        begin
                                next_state = S0;
                                 PAM_out = 3'b000; end

          S0 :   begin          next_state = Sx;  //machnester_in has to be 0
                                next_out = 3'b001; end

          S1 :   begin          next_state = Sx;  //machnester_in has to be 1
                                next_out = 3'b000; end

    default :   begin           next_state = Sx;  //default case
                                next_out = 3'b000; end
      endcase

    endmodule*/


    //ee417 lesson 5 Assignment 1 L5A1
    // Name: Ron Kalin, Date: 06-13-24  Group: Kalin/Jammeh
    // Design: manchester to PAM4 converter using
    // manchester to NRZ converter then NRZ to PAM4 converter
    //mealy, top level module, output PAM_out, input clock, reset, manchester_in
    module manchester_to_pam4 (
        output[1:0] PAM_out, // 2-bit PAM4 output
        input clk, // Clock for sampling
        input rst, // Reset
        input manchester_in); // Manchester-encoded 1bit serial input

    //define internal wires
    wire NRZ_out;

    //instantiate submodules
     manchester_to_NRZ_Mealy_case_nonglitchy  M1 (NRZ_out, manchester_in, clk, rst);
     NRZ_to_PAM_Mealy_case_nonglitchy  M2 (PAM_out, NRZ_out, clk, rst);

    endmodule

    //convert manchester to ZRZ
    module manchester_to_NRZ_Mealy_case_nonglitchy  (NRZ_out,
                                                     manchester_in,
                                                     clock, reset);
    output NRZ_out;
    input manchester_in, clock, reset;

    reg [1:0] state, next_state; // 3 total states from state diagram = 2 bits
    reg       next_out, NRZ_out; // assign values within always block

    parameter Sx = 2'b00; // waiting for new manchester input
    parameter S0 = 2'b01; // manchester 01 is being converted to NRZ 0
    parameter S1 = 2'b10; // manchester 10 is being converted to NRZ 1

    // Sequential logic updating the state
    always @ (posedge clock or posedge reset) //asynchronous reset
      if (reset) begin state <= Sx;
                       NRZ_out <= 1'b0; end
      else begin state <= next_state;
                  NRZ_out <= next_out; end

    // Combinational logic to find next_state and NRZ_out
    always @ *  //if state or manchester_in change
      case (state)
        Sx : if(manchester_in) begin
                                next_state = S1;
                                next_out = 1'b1; end
                    else        begin
                                next_state = S0;
                                next_out = 1'b0; end

```

```
280         S0 :                    begin
281                                 next_state = Sx;  //manchester_in has to be 1
282                                 next_out = 1'b0; end
283
284         S1 :                    begin
285                                 next_state = Sx;  //manchester_in has to be 0
286                                 next_out = 1'b1; end
287
288     default :   begin           next_state = Sx;  //default case
289                                 next_out = 1'b0; end
290       endcase
291
292     endmodule
293
294
295     //convert NRZ to PAM4
296     module NRZ_to_PAM_Mealy_case_nonglitchy  (PAM_out,
297                                               NRZ_in,
298                                               clock, reset);
299     output [1:0] PAM_out;
300     input NRZ_in, clock, reset;
301                                 // 1 bit=2 states, 2bits=4 states, 3bits=8 states, nbits=2^n
        states
302     reg [2:0] state, next_state; // 6 total states from state diagram = 3 bits
303     reg [1:0] next_out, PAM_out; // assign values within always block
304                                 // using next_out as a register prevents glitches
305
306     parameter S00 = 3'b000;// waiting for new NRZ input
307     parameter S0  = 3'b101;
308     parameter S01 = 3'b001;
309     parameter S10 = 3'b010;
310     parameter S1  = 3'b111;
311     parameter S11 = 3'b011;
312
313
314     // Sequential logic updating the state
315     always @ (posedge clock or posedge reset) //asynchronous reset
316       if (reset) begin state <= S00;
317                        PAM_out <= 2'b00; end
318       else begin state <= next_state;
319                     PAM_out <= next_out; end
320
321     // Combinational logic to find next_state and NRZ_out
322     always @ *  //if state or NRZ_in change
323       case (state)
324         S00 : if(NRZ_in) begin
325                             next_state = S1;
326                             next_out = 2'b00; end
327               else       begin
328                             next_state = S0;
329                             next_out = 2'b00; end
330
331         S0 : if(NRZ_in) begin
332                             next_state = S01;
333                             next_out = 2'b01; end
334               else       begin
335                             next_state = S00;
336                             next_out = 2'b00; end
337
338         S01: if(NRZ_in) begin
339                             next_state = S1;
340                             next_out = 2'b01; end
341               else       begin
342                             next_state = S0;
343                             next_out = 2'b01; end
344
345         S1 : if(NRZ_in) begin
346                             next_state = S11;
347                             next_out = 2'b11; end
348               else       begin
```

```
                            next_state = S10;
                            next_out = 2'b10; end

        S10: if(NRZ_in) begin
                            next_state = S1;
                            next_out = 2'b10; end
             else       begin
                            next_state = S0;
                            next_out = 2'b10; end

        S11: if(NRZ_in) begin
                            next_state = S1;
                            next_out = 2'b11; end
             else       begin
                            next_state = S0;
                            next_out = 2'b11; end

    default :           begin
                            next_state = S00;  //default case
                            next_out = 2'b00; end
     endcase

   endmodule
```