

```

1  /*EE417 Lesson 5 A1 -Manchester code to PAM4 code converter
2  Name: Ron Kalin, Date: 6/11/24
3  Group: Kalin, Jammeh
4  PAM4 combines two NRZ bits, which means 4 Manchester bits.
5  Manchester 0101 = 00 NRZ = 0 PAM4
6  Manchester 0110 = 01 NRZ = 1 PAM4
7  Manchester 1001 = 10 NRZ = 2 PAM4
8  Manchester 1010 = 11 NRZ = 3 PAM4
9  */
10 /*module manchester_to_pam4 ( //mealy, top level module
11     input wire clk, // Clock for sampling
12     input wire rst, // Reset
13     input wire manchester_in, // Manchester-encoded serial input
14     output reg [2:0] pam4_out // 3-bit PAM4 output
15 );
16     reg [1:0] state; // State machine for decoding
17     always @(posedge clk or posedge rst) begin
18         if (rst) begin
19             state <= 2'b00; // Initial state
20             pam4_out <= 3'b000; // Reset output
21         end else begin
22             case (state)
23                 2'b00: begin // waiting for rising edge
24                     if (manchester_in) state <= 2'b01;
25                 end
26                 2'b01: begin // Rising edge detected
27                     state <= 2'b10;
28                 end
29                 2'b10: begin // Falling edge detected
30                     state <= 2'b00;
31                     // Map Manchester data to PAM4 levels
32                     case (manchester_in)
33                         1'b0: pam4_out <= 3'b001; // -1
34                         1'b1: pam4_out <= 3'b010; // 0
35                     endcase
36                 end
37             endcase
38         end
39     end
40 endmodule
41
42 module manchester_to_PAM_mealy_assign_glitchy(
43     output PAM_out,
44     input manchester_in, input clock, input reset);
45 //this module will show glitchy mealy FSM conversion from manchester to PAM
46
47 reg [2:0] state; //number of state bits required = number of states in state diagram
48 wire [2:0] next_state; // ex: 5 to 8 states = 3 bits
49
50 //parameter sx = 2'b00; //waiting for new manchester input
51 //parameter s0 = 2'b01; //manchester 0 is being converted to 01
52 //parameter s1 = 2'b10; //manchester 1 is being converted to 10
53
54 parameter S0 = 4'b0000; //waiting for new manchester input
55 parameter S00 = 4'b0000;
56 parameter S01 = 4'b0001;
57 parameter S001 = 4'b0001;
58 parameter S010 = 4'b0010;
59 parameter S0010 = 4'b0010;
60 parameter S0011 = 4'b0011;
61 parameter S0100 = 4'b0100;
62 parameter S0101 = 4'b0101;
63
64 parameter S1 = 4'b0000; //waiting for new manchester input
65 parameter S10 = 4'b0000;
66 parameter S11 = 4'b0001;
67 parameter S101 = 4'b0010;
68 parameter S110 = 4'b0010;

```

```

70 parameter S1010 = 4'b0011;
71 parameter S1011 = 4'b0100;
72 parameter S1100 = 4'b1100;
73 parameter S1101 = 4'b1101;
74
75 parameter S2      = 4'b0010; //waiting for new manchester input
76 parameter S20     = 4'b0010;
77 parameter S21     = 4'b0011;
78 parameter S201    = 4'b0101;
79 parameter S210    = 4'b0110;
80 parameter S2010   = 4'b1010;
81 parameter S2011   = 4'b1011;
82 parameter S2100   = 4'b1100;
83 parameter S2101   = 4'b1101;
84
85 parameter S3      = 4'b0011; //waiting for new manchester input
86 parameter S30     = 4'b0011;
87 parameter S31     = 4'b0011;
88 parameter S301    = 4'b0101;
89 parameter S310    = 4'b0110;
90 parameter S3010   = 4'b1010;
91 parameter S3011   = 4'b1011;
92 parameter S3100   = 4'b1100;
93 parameter S3101   = 4'b1101;
94
95 // Sequential logic updating the state
96
97 always @ (posedge clock or posedge reset)
98     if (reset) state <= S0;
99     else state <= next_state;
100
101 // Combinational logic to find next_state and PAM_out
102
103 assign next_state [0] = manchester_in | (~state[1] & ~ state[0]);
104 assign next_state [1] = ~state[1] & manchester_in;
105 assign PAM_out = ~state[0] | (manchester_in & state[1]);
106
107
108 endmodule
109
110
111 module manchester_to_PAM_Mealy_case_glitchy(PAM_out,
112                                             manchester_in,
113                                             clock, reset);
114 output PAM_out;
115 input manchester_in, clock, reset;
116
117 reg [2:0] state, next_state; //number of state bits required = number of states in state
118 reg PAM_out; // ex: 5 to 8 states = 3 bits, 9 to 16 states=4bits
119 //to assign it values within always block
120
121 parameter Sx = 2'b01; //waiting for new manchester input
122 parameter S0 = 2'b01; //manchester 0 is being converter to 01
123 parameter S1 = 2'b11; //manchester 1 is being converter to 10
124
125 // Sequential logic updating the state
126
127 always @ (posedge clock or posedge reset) //asynchronous reset
128     if (reset) state <= Sx;
129     else state <= next_state;
130
131 // Combinational logic to find next_state and PAM_out
132
133 always @ * //if state or manchester_in change
134     case (state)
135         Sx : if(manchester_in) begin
136                 next_state = S1;
137                 PAM_out = 3'b001; end

```

```
138         else          begin
139             next_state = S0;
140             PAM_out = 3'b000; end
141
142     S0 :    begin      next_state = Sx; //manchester_in has to be 0
143                     PAM_out = 3'b001; end
144
145     S1 :    begin      next_state = Sx; //manchester_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
159 output reg PAM_out;
160 input manchester_in, clock, reset;
161
162 reg [2:0] state; //the use of register assures no glitches
163 wire [2:0] next_state;
164 wire      next_out;
165
166 parameter Sx = 2'b01; //waiting for new manchester input
167 parameter S0 = 2'b00; // manchester 0 is being converted to 01
168 parameter S1 = 2'b00; // manchester 1 is being converted to 10
169
170 // sequential logic updating the state
171
172 always @ (posedge clock or posedge reset) //asynchronous reset
173     if (reset) begin state <= Sx;
174                     PAM_out <= 3'b000; end
175     else          begin state <= next_state;
176                     PAM_out <= next_out; end
177
178 // combinational logic to find next_state and PAM_out
179
180 assign next_state[0] = manchester_in | (~state[1] & ~state[0]);
181 assign next_state[1] = ~state[1] & manchester_in;
182 assign next_out = ~next_state[0] | (manchester_in & ~next_state[1]);
183
184 endmodule
185
186 module manchester_to_PAM_Mealy_case_nonglitchy (PAM_out,
187                                                    manchester_in,
188                                                    clock, reset);
189
190 output PAM_out;
191 input manchester_in, clock, reset;
192
193 reg [2:0] state, next_state;
194 reg      next_out, PAM_out; // assign values within always block
195
196 parameter Sx = 2'b01; //waiting for new manchester input
197 parameter S0 = 2'b00; // manchester 0 is being converted to 01
198 parameter S1 = 2'b00; // manchester 1 is being converted to 10
199
200 // Sequential logic updating the state
201 always @ (posedge clock or posedge reset) //asynchronous reset
202     if (reset) begin state <= Sx;
203                     PAM_out <= 1'b0; end
204     else begin state <= next_state;
205                     PAM_out <= next_out; end
206
207 // 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
210             next_state = S1;
211             PAM_out = 3'b001; end
212         else
213             begin
214                 next_state = S0;
215                 PAM_out = 3'b000; end
216         S0 : begin
217             next_state = Sx; //manchester_in has to be 0
218             next_out = 3'b001; end
219         S1 : begin
220             next_state = Sx; //manchester_in has to be 1
221             next_out = 3'b000; end
222     default : begin
223         next_state = Sx; //default case
224         next_out = 3'b000; end
225     endcase
226 endmodule*/
227
228 //ee417 lesson 5 Assignment 1 L5A1
229 // Name: Ron Kalin, Date: 06-13-24 Group: Kalin/Jammeh
230 // Design: manchester to PAM4 converter using
231 // Manchester to NRZ converter then NRZ to PAM4 converter
232 //mealy, top level module, output PAM_out, input clock, reset, manchester_in
233 module manchester_to_pam4 (
234     output[1:0] PAM_out, // 2-bit PAM4 output
235     input clk, // Clock for sampling
236     input rst, // Reset
237     input manchester_in); // Manchester-encoded 1bit serial input
238
239 //define internal wires
240 wire NRZ_out;
241
242 //instantiate submodules
243 manchester_to_NRZ_Mealy_case_nonglitchyM1 (NRZ_out, manchester_in, clk, rst);
244 NRZ_to_PAM_Mealy_case_nonglitchyM2 (PAM_out, NRZ_out, clk, rst);
245
246 endmodule
247
248 //convert manchester to ZRZ
249 module manchester_to_NRZ_Mealy_case_nonglitchy (NRZ_out,
250                                                 manchester_in,
251                                                 clock, reset);
252
253 output NRZ_out;
254 input manchester_in, clock, reset;
255
256 reg [1:0] state, next_state; // 3 total states from state diagram = 2 bits
257 reg next_out, NRZ_out; // assign values within always block
258
259 parameter Sx = 2'b00; // waiting for new manchester input
260 parameter S0 = 2'b01; // manchester 01 is being converted to NRZ 0
261 parameter S1 = 2'b10; // manchester 10 is being converted to NRZ 1
262
263 // Sequential logic updating the state
264 always @ (posedge clock or posedge reset) //asynchronous reset
265     if (reset) begin state <= Sx;
266                     NRZ_out <= 1'b0; end
267     else begin state <= next_state;
268              NRZ_out <= next_out; end
269
270 // Combinational logic to find next_state and NRZ_out
271 always @ * //if state or manchester_in change
272     case (state)
273         Sx : if(manchester_in) begin
274             next_state = S1;
275             next_out = 1'b1; end

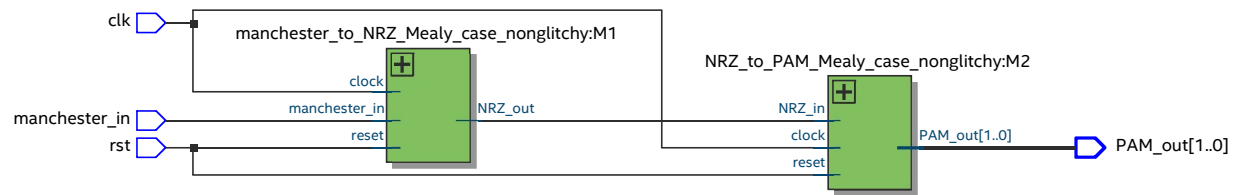
```

```

276         else                begin
277             next_state = S0;
278             next_out = 1'b0; end
279
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
300 output [1:0] PAM_out;
301 input NRZ_in, clock, reset;
302                                     // 1 bit=2 states, 2bits=4 states, 3bits=8 states, nbits=2^nb
303
304 states
305 reg [2:0] state, next_state; // 6 total states from state diagram = 3 bits
306 reg [1:0] next_out, PAM_out; // assign values within always block
307                                     // using next_out as a register prevents glitches
308
309 parameter S00 = 3'b000; // waiting for new NRZ input
310 parameter S0  = 3'b101;
311 parameter S01 = 3'b001;
312 parameter S10 = 3'b010;
313 parameter S1  = 3'b111;
314 parameter S11 = 3'b011;
315
316 // Sequential logic updating the state
317 always @ (posedge clock or posedge reset) //asynchronous reset
318 if (reset) begin state <= S00;
319                 PAM_out <= 2'b00; end
320 else begin state <= next_state;
321           PAM_out <= next_out; end
322
323 // Combinational logic to find next_state and NRZ_out
324 always @ * //if state or NRZ_in change
325 case (state)
326     S00 : if(NRZ_in) begin
327             next_state = S1;
328             next_out = 2'b00; end
329         else begin
330             next_state = S0;
331             next_out = 2'b00; end
332
333     S0 : if(NRZ_in) begin
334             next_state = S01;
335             next_out = 2'b01; end
336         else begin
337             next_state = S00;
338             next_out = 2'b00; end
339
340     S01: if(NRZ_in) begin
341             next_state = S1;
342             next_out = 2'b01; end
343         else begin
344             next_state = S0;
345             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
349         next_state = s10;
350         next_out = 2'b10; end
351
352     s10: if(NRZ_in) begin
353         next_state = s1;
354         next_out = 2'b10; end
355     else begin
356         next_state = s0;
357         next_out = 2'b10; end
358
359     s11: if(NRZ_in) begin
360         next_state = s1;
361         next_out = 2'b11; end
362     else begin
363         next_state = s0;
364         next_out = 2'b11; end
365
366     default : begin
367         next_state = s00; //default case
368         next_out = 2'b00; end
369     endcase
370
371 endmodule
372
373
374
```



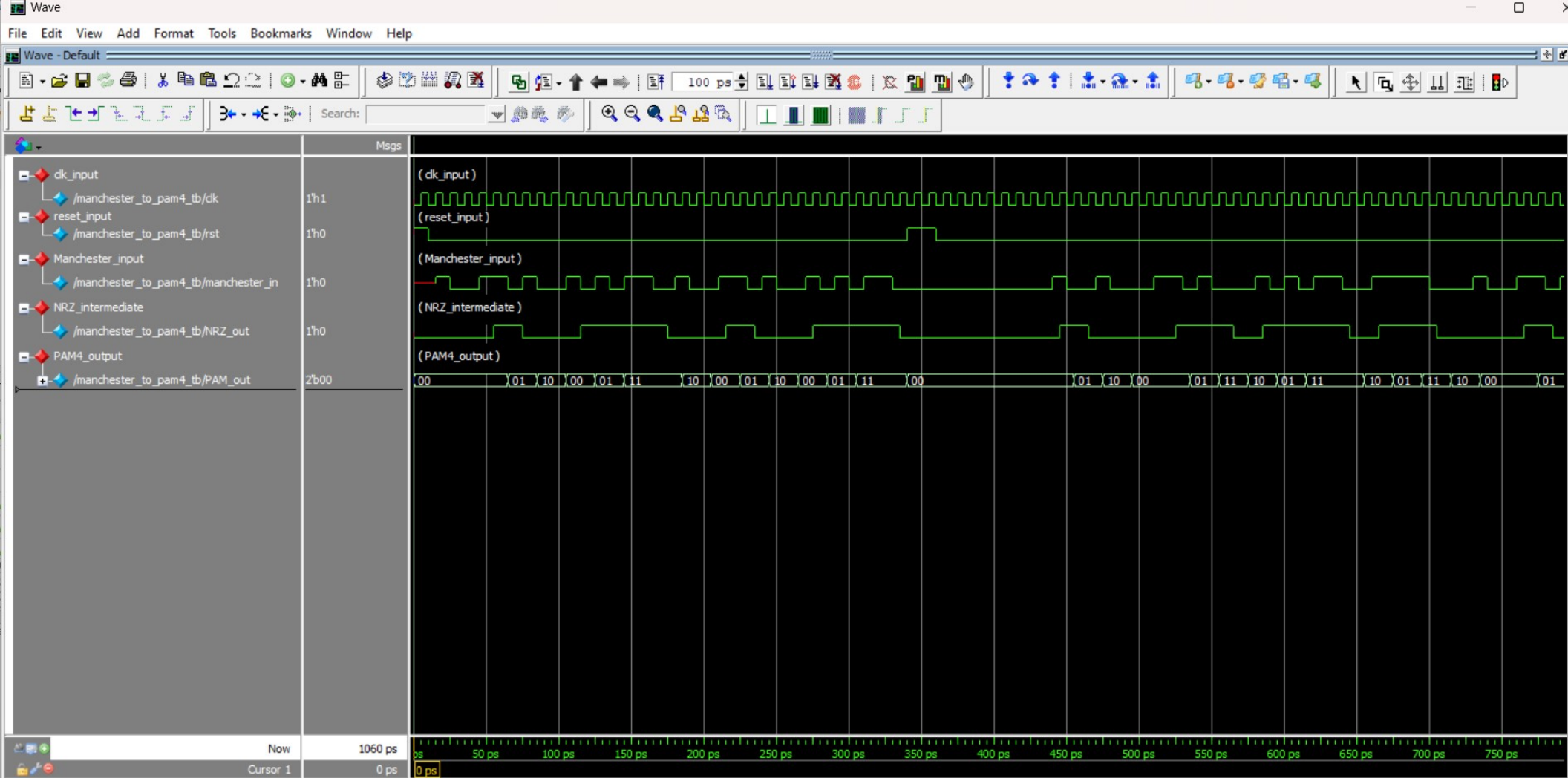
```

1 //ee417 lesson 5 Assignment 1, L5A1
2 // Name: Ron Kalin, Date: 06-13-24 Group: Kalin/Jammeh
3 // Testbench for Design: manchester to PAM4 converter using
4 // manchester to NRZ converter then NRZ to PAM4 converter
5 //Step1 define test bench name
6 module manchester_to_pam4_tb();
7
8 /*original module declaration
9 module manchester_to_pam4 (
10     output[2:0] PAM_out, // 3-bit PAM4 output
11     input clk, // Clock for sampling
12     input rst, // Reset
13     input manchester_in); // Manchester-encoded 1bit serial input*/
14 //Step2 define inputs as registers, outputs as wires
15 reg clk, rst, manchester_in;
16 reg [3:0] a;
17 wire [1:0] PAM_out;
18 //internal probe wires: observe change in state..Questo error if not correct no. of
19 wire NRZ_out;
20
21 //Step3 define unit under test
22 manchester_to_pam4 UUT (PAM_out,clk,rst,manchester_in);
23
24 //internal probes to track logic and troubleshoot
25 assign NRZ_out= UUT.NRZ_out;
26
27 //Step4 open initial block, define all possible input combinations
28 // Clock generation (adjust the period as needed)
29 initial begin
30     clk=0;
31     forever
32         #5 clk = ~clk;
33 end
34
35 initial //reset is active high, longer time to count when reset is inactive (low)
36 begin //4 cases with two selects
37     rst = 1'b1; //reset on
38     a=4'b0000;
39     # 10 rst = 1'b0; //reset off
40 //start manchester sequence
41 repeat (2) begin // repeat x times
42     #5 manchester_in=1'b1;
43     #10 manchester_in=1'b0;
44     #10 manchester_in=1'b0;
45     #10 manchester_in=1'b1; //PAM 4=2
46
47     #10 manchester_in=1'b1;
48     #10 manchester_in=1'b0;
49     #10 manchester_in=1'b1;
50     #10 manchester_in=1'b0; //PAM 4=3
51
52     #10 manchester_in=1'b0;
53     #10 manchester_in=1'b1;
54     #10 manchester_in=1'b0;
55     #10 manchester_in=1'b1; //PAM 4=0
56
57     #10 manchester_in=1'b0;
58     #10 manchester_in=1'b1;
59     #10 manchester_in=1'b1;
60     #10 manchester_in=1'b0; //PAM 4=1
61     #10;
62 end
63
64 rst = 1'b1;
65 #20 rst=1'b0;
66
67 repeat (15) begin //cycle thru every possible combination of four series inputs
68     #10 manchester_in=a[3];

```



```
69     #10 manchester_in=a[2];
70     #10 manchester_in=a[1];
71     #10 manchester_in=a[0];
72     a=a+1;
73 end
74
75     #100 $stop; //close debug window to view waveform viewer
76 end
77
78 //Step5 Display the results
79 initial begin //monitor counter value
80     $display("_____output_PAM_out = -PAM4-");
81     $monitor("clk_in = %b: rst_in = %b:  output_PAM_out = %d ",
82     clk, rst, PAM_out);
83 end
84 endmodule
```



Output table

_____output_PAM_out = -PAM4-

clk_in = 0: rst_in = 1: output_PAM_out = 0

clk_in = 1: rst_in = 1: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 1

clk_in = 0: rst_in = 0: output_PAM_out = 1

clk_in = 1: rst_in = 0: output_PAM_out = 1

clk_in = 0: rst_in = 0: output_PAM_out = 1

clk_in = 1: rst_in = 0: output_PAM_out = 2

clk_in = 0: rst_in = 0: output_PAM_out = 2

clk_in = 1: rst_in = 0: output_PAM_out = 2

clk_in = 0: rst_in = 0: output_PAM_out = 2

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 0

clk_in = 0: rst_in = 0: output_PAM_out = 0

clk_in = 1: rst_in = 0: output_PAM_out = 1

clk_in = 0: rst_in = 0: output_PAM_out = 1

```
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
```

```
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 1: output_PAM_out = 0
# clk_in = 1: rst_in = 1: output_PAM_out = 0
# clk_in = 0: rst_in = 1: output_PAM_out = 0
# clk_in = 1: rst_in = 1: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
```

```
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
```

clk_in = 0: rst_in = 0: output_PAM_out = 3
clk_in = 1: rst_in = 0: output_PAM_out = 2
clk_in = 0: rst_in = 0: output_PAM_out = 2
clk_in = 1: rst_in = 0: output_PAM_out = 2
clk_in = 0: rst_in = 0: output_PAM_out = 2
clk_in = 1: rst_in = 0: output_PAM_out = 1
clk_in = 0: rst_in = 0: output_PAM_out = 1
clk_in = 1: rst_in = 0: output_PAM_out = 1
clk_in = 0: rst_in = 0: output_PAM_out = 1
clk_in = 1: rst_in = 0: output_PAM_out = 3
clk_in = 0: rst_in = 0: output_PAM_out = 3
clk_in = 1: rst_in = 0: output_PAM_out = 3
clk_in = 0: rst_in = 0: output_PAM_out = 3
clk_in = 1: rst_in = 0: output_PAM_out = 3
clk_in = 0: rst_in = 0: output_PAM_out = 3
clk_in = 1: rst_in = 0: output_PAM_out = 3
clk_in = 0: rst_in = 0: output_PAM_out = 3
clk_in = 1: rst_in = 0: output_PAM_out = 2
clk_in = 0: rst_in = 0: output_PAM_out = 2
clk_in = 1: rst_in = 0: output_PAM_out = 2
clk_in = 0: rst_in = 0: output_PAM_out = 2
clk_in = 1: rst_in = 0: output_PAM_out = 1
clk_in = 0: rst_in = 0: output_PAM_out = 1
clk_in = 1: rst_in = 0: output_PAM_out = 1
clk_in = 0: rst_in = 0: output_PAM_out = 1
clk_in = 1: rst_in = 0: output_PAM_out = 3
clk_in = 0: rst_in = 0: output_PAM_out = 3
clk_in = 1: rst_in = 0: output_PAM_out = 3
clk_in = 0: rst_in = 0: output_PAM_out = 3

```
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 1
```



```
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 1
# clk_in = 0: rst_in = 0: output_PAM_out = 1
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 3
# clk_in = 0: rst_in = 0: output_PAM_out = 3
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 2
# clk_in = 0: rst_in = 0: output_PAM_out = 2
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
```

```
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
# clk_in = 0: rst_in = 0: output_PAM_out = 0
# clk_in = 1: rst_in = 0: output_PAM_out = 0
```

Summary

- This design takes in Manchester code and converts it to PAM4 data.
- There are 2 modules inside the design.
- The first module converts the Manchester to NRZ
- The second module converts the NRZ data to PAM4 data.