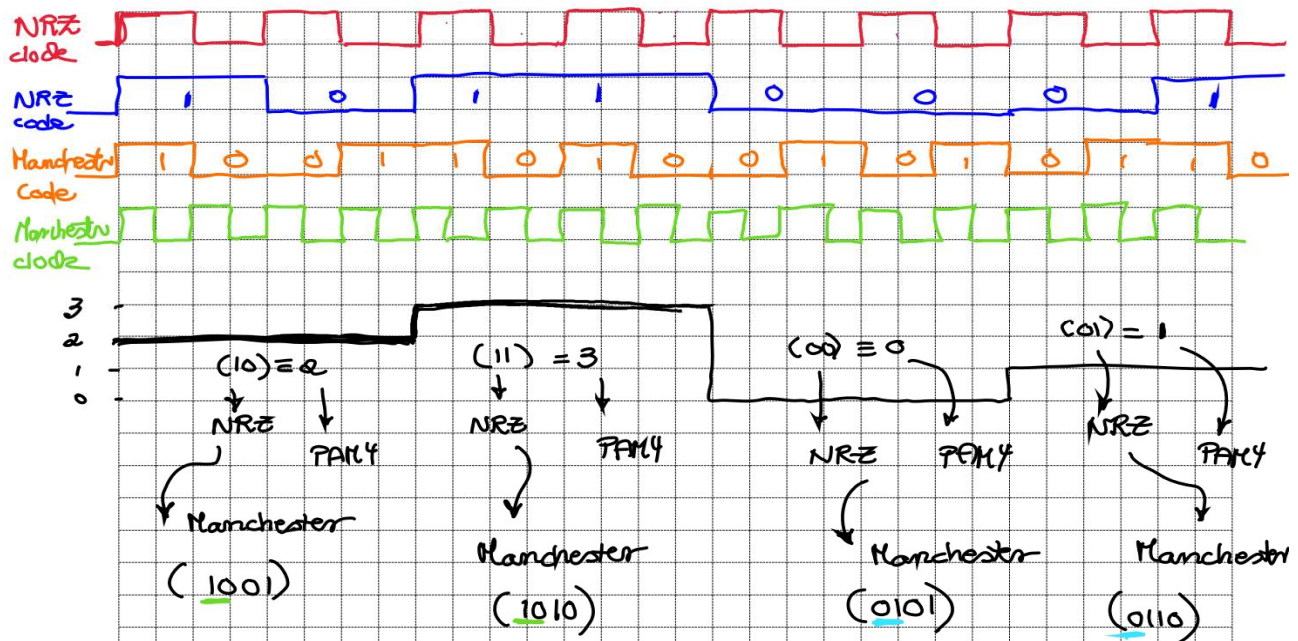


Create a test bench to test the functionality of the module. In your test bench, change the input data during the clock cycle, and test how your design is dealing with possible glitches, when the output is connected to combinational logic directly without any registers. Clearly show how you can modify the code to avoid output glitches. Test your updated design approach. Document your work well in the final pdf submission. The deliverables should include the code, test bench and simulation results.



Design Methodology

Design selected was to create a converter from manchester to NRZ, then from NRZ to PAM4.

The reason was that there would be fewer number of states.

To go direct from Manchester to PAM4 would take 36 states.

Design Code

```
229 //ee417 lesson 5 Assignment 1 L5A1
230 // Name: Ron Kalin, Date: 06-13-24 Group: Kalin/Jammeh
231 // Design: manchester to PAM4 converter using
232 // manchester to NRZ converter then NRZ to PAM4 converter
233 //mealy, top level module, output PAM_out, input clock, reset, manchester_in
234 module manchester_to_pam4 (
235     output[1:0] PAM_out, // 2-bit PAM4 output
236     input clk, // Clock for sampling
237     input rst, // Reset
238     input manchester_in); // Manchester-encoded 1bit serial input
239
240 //define internal wires
241 wire NRZ_out;
242
243 //instantiate submodules
244 manchester_to_NRZ_Mealy_case_nonglitchy M1 (NRZ_out, manchester_in, clk, rst);
245 NRZ_to_PAM_Mealy_case_nonglitchy M2 (PAM_out, NRZ_out, clk, rst);
246
247 endmodule
248
249 //convert manchester to NRZ
250 module manchester_to_NRZ_Mealy_case_nonglitchy (NRZ_out,
251                                                 manchester_in,
252                                                 clock, reset);
253
254 output NRZ_out;
255 input manchester_in, clock, reset;
256
257 reg [1:0] state, next_state; // 3 total states from state diagram = 2 bits
258 reg next_out, NRZ_out; // assign values within always block
259
260 parameter Sx = 2'b00; // waiting for new manchester input
261 parameter S0 = 2'b01; // manchester 01 is being converted to NRZ 0
262 parameter S1 = 2'b10; // manchester 10 is being converted to NRZ 1
263
264 // Sequential logic updating the state
265 always @ (posedge clock or posedge reset) //asynchronous reset
266     if (reset) begin state <= Sx;
267                     NRZ_out <= 1'b0; end
268     else begin state <= next_state;
269             NRZ_out <= next_out; end
270
271 // Combinational logic to find next_state and NRZ_out
272 always @ * //if state or manchester_in change
273     case (state)
274         Sx : if(manchester_in) begin
275                 next_state = S1;
276                 next_out = 1'b1; end
277         else
278             begin
279                 next_state = S0;
280                 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
289         next_state = Sx; //default case
290         next_out = 1'b0; end
291
292     endcase
293 endmodule
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
303     // 1 bit=2 states, 2bits=4 states, 3bits=8 states, nbits=2^N
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
346
347     S1 : if(NRZ_in) begin
348         next_state = S11;
349         next_out = 2'b11; end
350     else begin
351

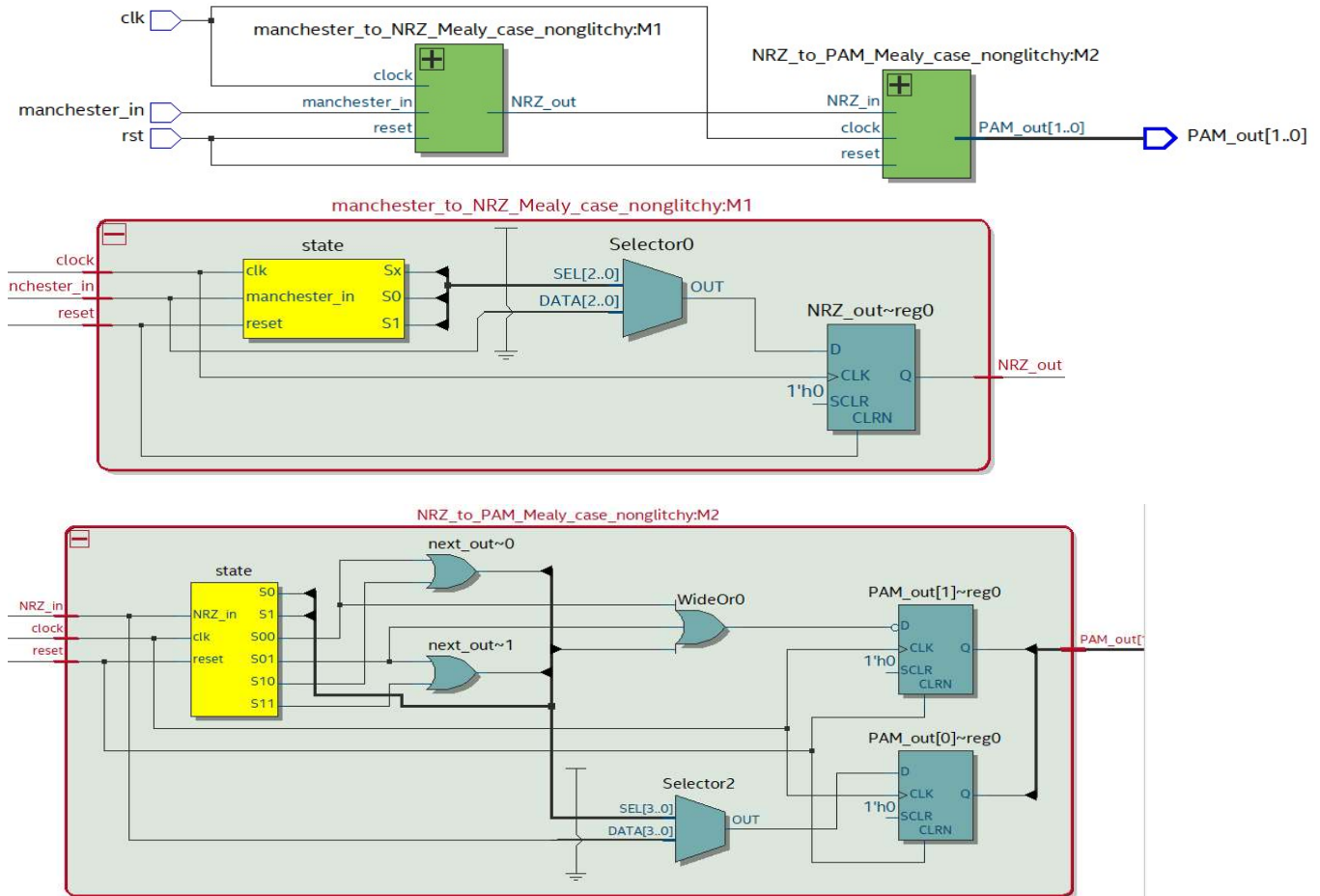
```

```

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 :
367         begin
368             next_state = S00; //default case
369             next_out = 2'b00; end
370
371     endcase
372 endmodule
373
374

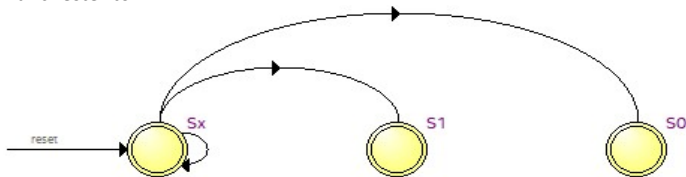
```

RTL Viewer



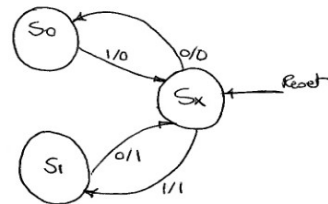
State Machine Viewer

Manchester to NRZ

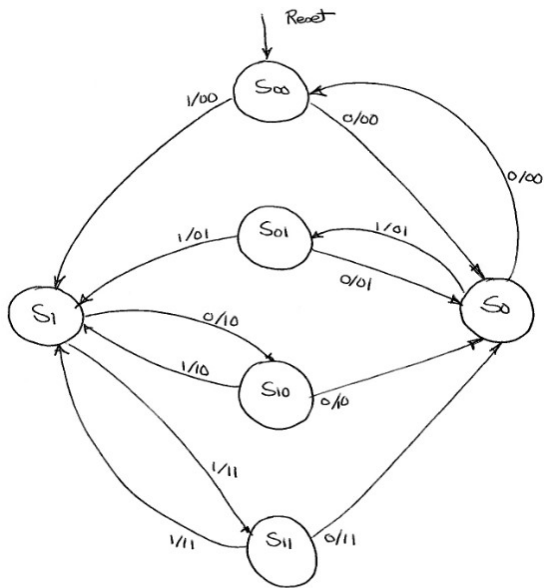


	Source State	Destination State	Condition
1	S0	Sx	(!manchester_in)
2	Sx	S0	(!manchester_in)
3	Sx	Sx	(manchester_in)

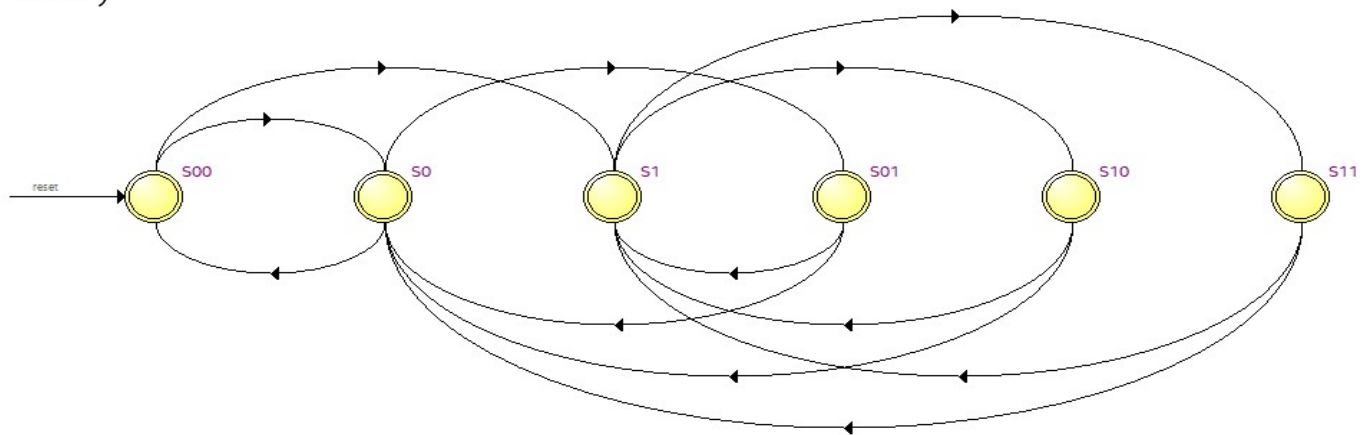
Manchester to NRZ code - Mealy machine



NRZ to PAM



NRZ-PAM9 Mealy machine
(6 states)



	Source State	Destination State	Condition
1	S00	S01	(NRZ_in)
2	S00	S00	(!NRZ_in)
3	S01	S01	(NRZ_in)
4	S01	S00	(!NRZ_in)
5	S10	S01	(NRZ_in)
6	S10	S00	(!NRZ_in)
7	S11	S01	(NRZ_in)
8	S11	S00	(!NRZ_in)

Testbench Code

Date: June 13, 2024

manchester_to_pam4_tb.v

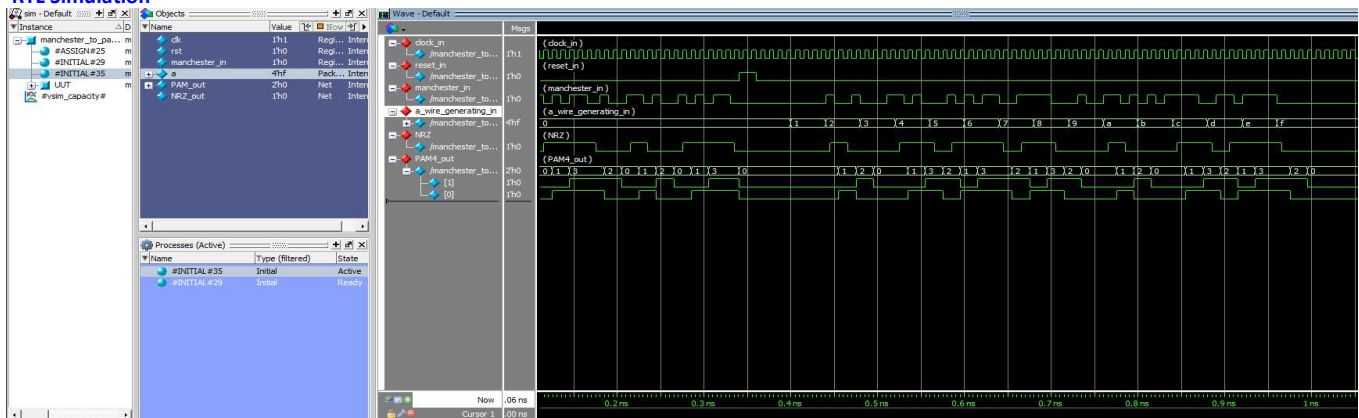
Project: manche

```

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..Questa error if not correct no. of
19 //bits
20 wire NRZ_out;
21
22 //Step3 define unit under test
23 manchester_to_pam4 UUT (PAM_out,clk,rst,manchester_in);
24
25 //internal probes to track logic and troubleshoot
26 assign NRZ_out= UUT.NRZ_out;
27
28 //Step4 open initial block, define all possible input combinations
29 //Clock generation (adjust the period as needed)
30 initial begin
31     clk=0;
32     forever
33         #5 clk = ~clk;
34 end
35
36 initial //reset is active high, longer time to count when reset is inactive (low)
37 begin //4 cases with two selects
38     rst = 1'b1; //reset on
39     a=4'b0000;
40     # 10 rst = 1'b0; //reset off
41 //start manchester sequence
42 repeat (2) begin // repeat x times
43     #5 manchester_in=1'b1;
44     #10 manchester_in=1'b0;
45     #10 manchester_in=1'b0;
46     #10 manchester_in=1'b1; //PAM 4=2
47
48     #10 manchester_in=1'b1;
49     #10 manchester_in=1'b0;
50     #10 manchester_in=1'b1;
51     #10 manchester_in=1'b0; //PAM 4=3
52
53     #10 manchester_in=1'b0;
54     #10 manchester_in=1'b1;
55     #10 manchester_in=1'b0;
56     #10 manchester_in=1'b1; //PAM 4=0
57
58     #10 manchester_in=1'b0;
59     #10 manchester_in=1'b1;
60     #10 manchester_in=1'b1;
61     #10 manchester_in=1'b0; //PAM 4=1
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

```

RTL Simulation



```
#                                     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 = 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 = 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
```

Conclusion

As can be clearly seen from the simulation above, the manchester input sequence match the chart that was given at the beginning of the assignment.

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