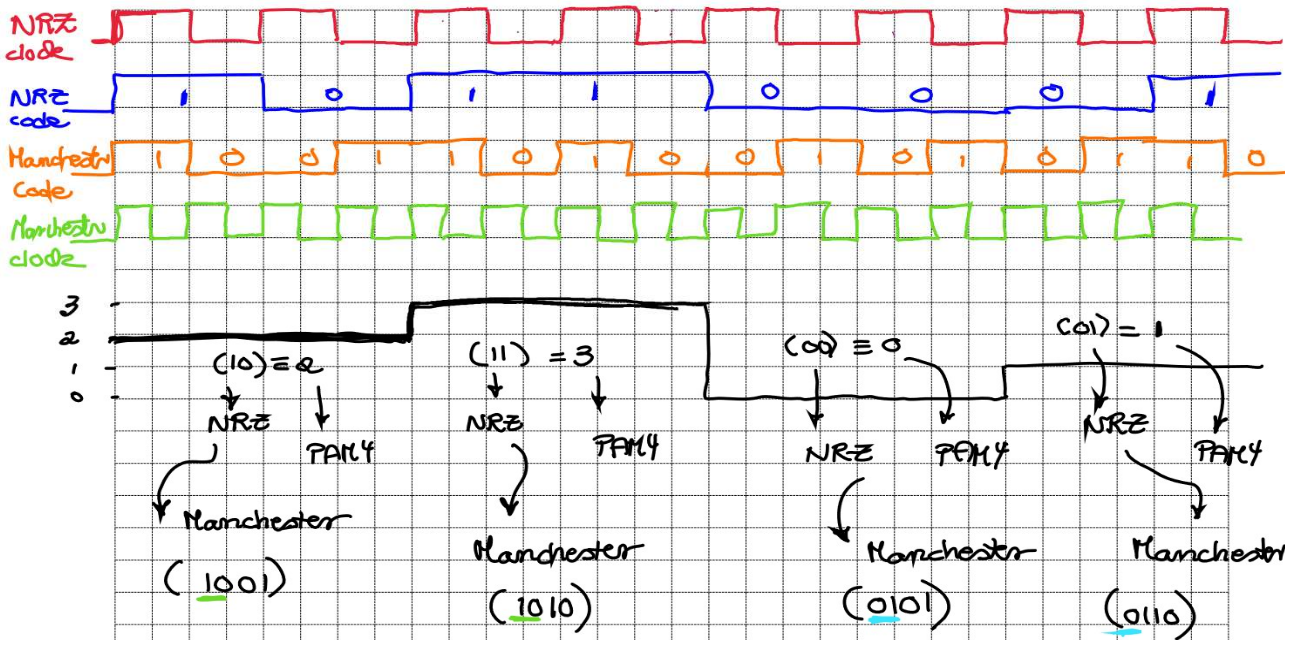
EE417 Advanced Digital Logic with Verilog HDL

L5 Assignment 1: Convert serial single-bit Manchester code to PAM4 (2-bit outputs) voltage levels

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.

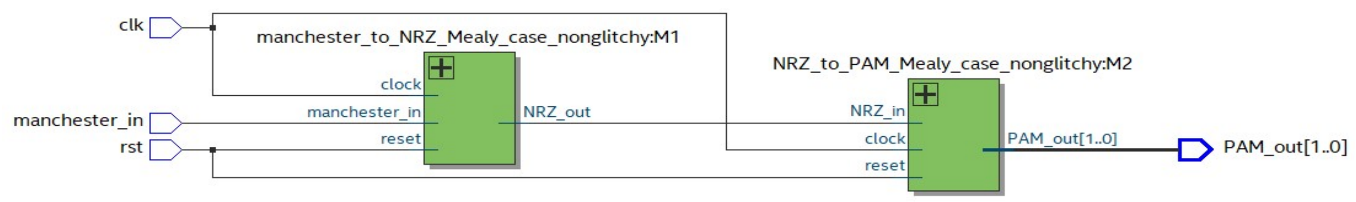
Design Code

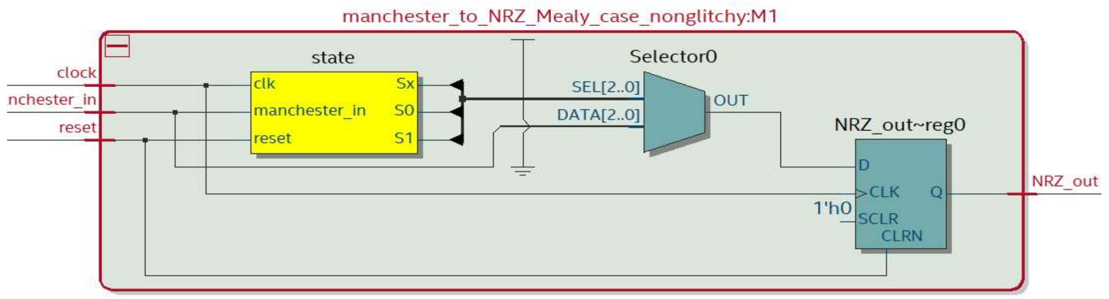
```
//ee417 lesson 5 Assignment 1 L5A1
// Name: Ron Kalin, Date: 06-13-24  Group: Kalin/Jammeh
// Design: manchester to PAM4 converter using
// manchester to NRZ converter then NRZ to PAM4 converter
//mealy, top level module, output PAM_out, input clock, reset, manchester_in
module manchester_to_pam4  (
    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

```

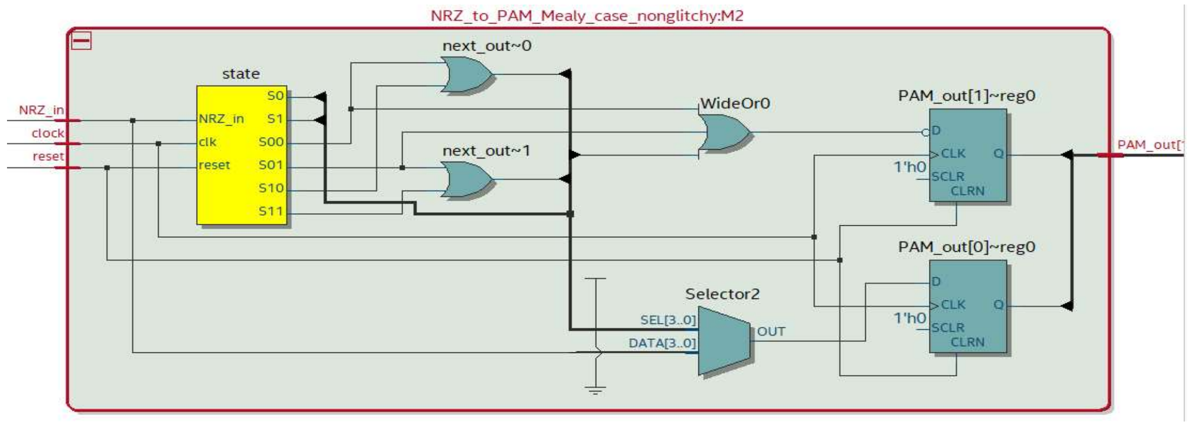
```
     S0 :                      begin
                               next_state = Sx;  //manchester_in has to be 1
                               next_out = 1'b0; end

     S1 :                      begin
                               next_state = Sx;  //manchester_in has to be 0
                               next_out = 1'b1; end

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

endmodule


//convert NRZ to PAM4
module NRZ_to_PAM_Mealy_case_nonglitchy  (PAM_out,
                                          NRZ_in,
                                          clock, reset);
output [1:0] PAM_out;
input NRZ_in, clock, reset;
                              // 1 bit=2 states, 2bits=4 states, 3bits=8 states, nbits=2^n
states
reg [2:0] state, next_state; // 6 total states from state diagram = 3 bits
reg [1:0] next_out, PAM_out; // assign values within always block
                              // using next_out as a register prevents glitches

parameter S00 = 3'b000;// waiting for new NRZ input
parameter S0  = 3'b101;
parameter S01 = 3'b001;
parameter S10 = 3'b010;
parameter S1  = 3'b111;
parameter S11 = 3'b011;


// Sequential logic updating the state
always @ (posedge clock or posedge reset) //asynchronous reset
  if (reset) begin state <= S00;
                   PAM_out <= 2'b00; end
  else begin state <= next_state;
               PAM_out <= next_out; end

// Combinational logic to find next_state and NRZ_out
always @ *  //if state or NRZ_in change
  case (state)
    S00 : if(NRZ_in) begin
                        next_state = S1;
                        next_out = 2'b00; end
          else       begin
                        next_state = S0;
                        next_out = 2'b00; end

     S0 : if(NRZ_in) begin
                        next_state = S01;
                        next_out = 2'b01; end
          else       begin
                        next_state = S00;
                        next_out = 2'b00; end

     S01: if(NRZ_in) begin
                        next_state = S1;
                        next_out = 2'b01; end
          else       begin
                        next_state = S0;
                        next_out = 2'b01; end

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

RTL Viewer

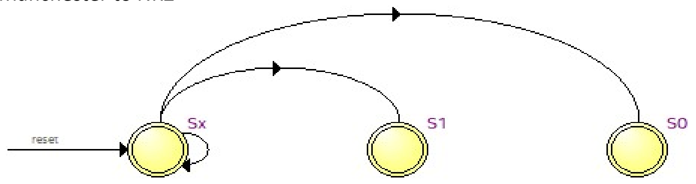


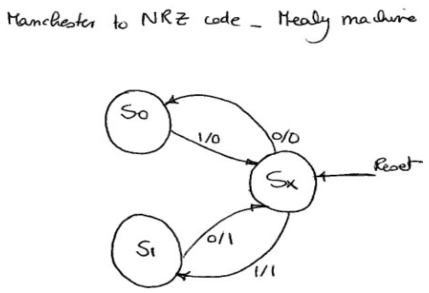




State Machine Viewer

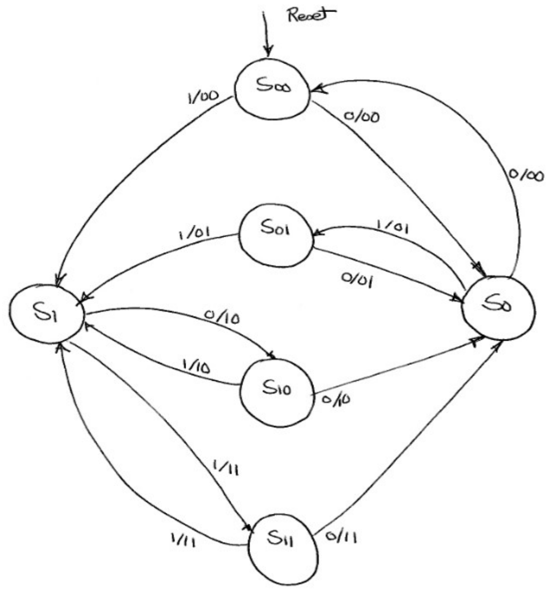
Manchester to NRZ



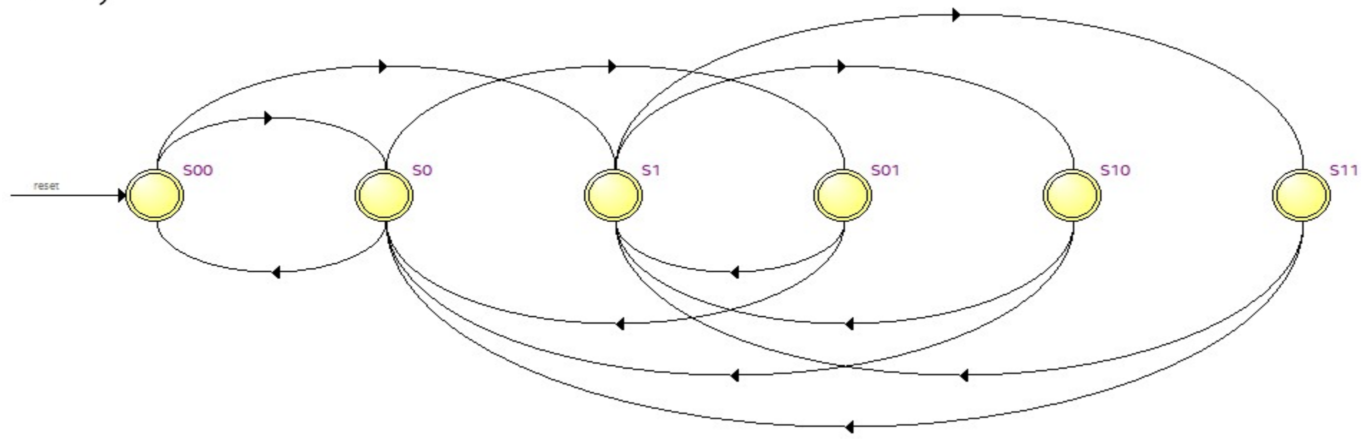


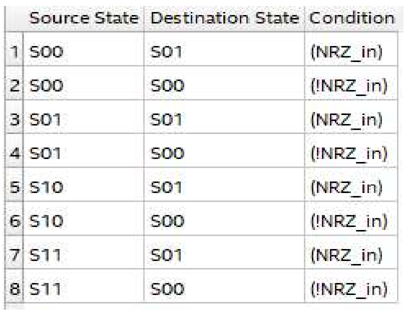
	Source State	Destination State	Condition
1	S0	Sx	
2	Sx	S0	(!manchester_in)
3	Sx	Sx	(manchester_in)

NRZ to PAM



NRZ - PAM4 Mealy machine
(6 states)





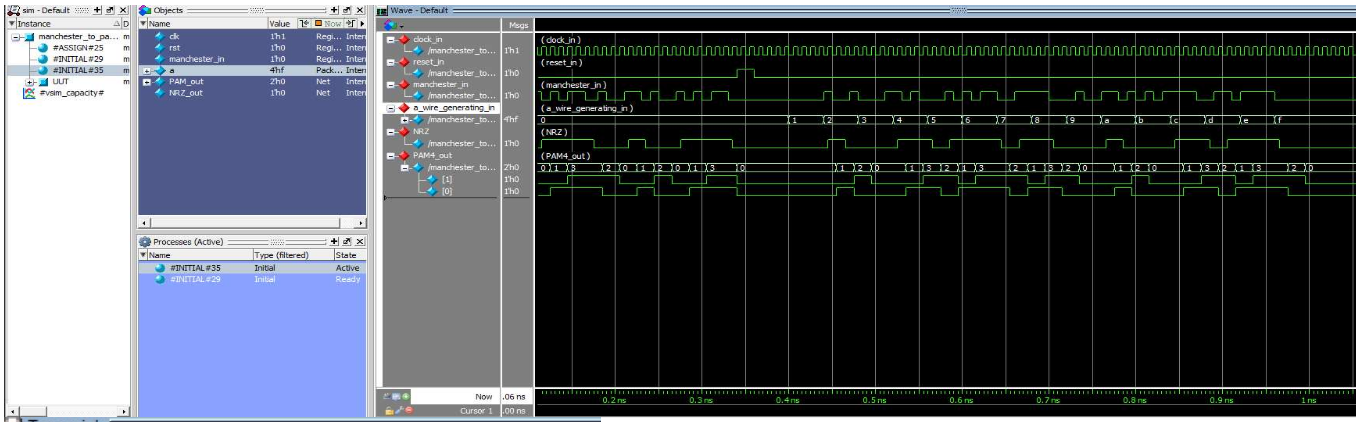
	Source State	Destination State	Condition
1	S00	S01	(NRZ_in)
2	S00	S00	(!NRZ_in)
3	S01	S01	(NRZ_in)
4	S01	S00	(!NRZ_in)
5	S10	S01	(NRZ_in)
6	S10	S00	(!NRZ_in)
7	S11	S01	(NRZ_in)
8	S11	S00	(!NRZ_in)

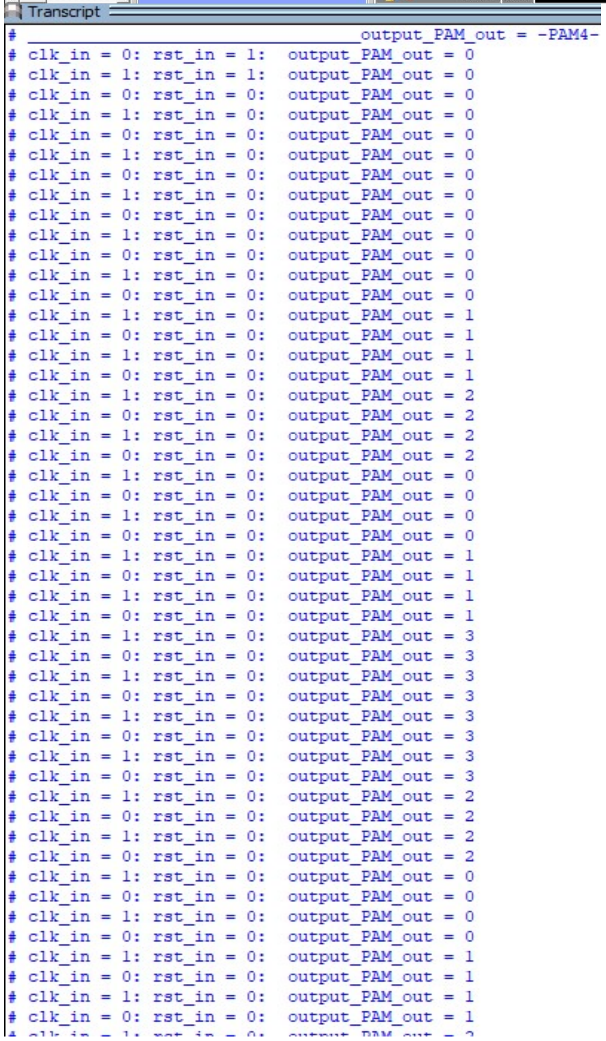
Testbench Code

```
//ee417 lesson 5 Assignment 1, L5A1
// Name: Ron Kalin, Date: 06-13-24  Group: Kalin/Jammeh
// Testbench for Design: manchester to PAM4 converter using
// manchester to NRZ converter then NRZ to PAM4 converter
//Step1 define test bench name
module manchester_to_pam4_tb ();

/*original module declaration
module manchester_to_pam4 (
    output[2:0] PAM_out, // 3-bit PAM4 output
    input clk, // Clock for sampling
    input rst, // Reset
    input manchester_in); // Manchester-encoded 1bit serial input*/
//Step2 define inputs as registers, outputs as wires
reg clk, rst, manchester_in;
reg [3:0] a;
wire [1:0] PAM_out;
//internal probe wires: observe change in state..Questa error if not correct no. of
bits
wire NRZ_out;

//Step3 define unit under test
manchester_to_pam4  UUT (PAM_out,clk,rst,manchester_in);

//internal probes to track logic and troubleshoot
assign NRZ_out= UUT.NRZ_out;

//Step4 open initial block, define all possible input combinations
// Clock generation (adjust the period as needed)
initial begin
  clk=0;
  forever
  #5 clk = ~clk;
end

initial   //reset is active high, longer time to count when reset is inactive (low)
 begin   //4 cases with two selects
  rst  = 1'b1; //reset on
  a=4'b0000;
  # 10 rst = 1'b0;  //reset off
//start manchester sequence
  repeat (2) begin  // repeat x times
  #5   manchester_in=1'b1;
  #10  manchester_in=1'b0;
  #10  manchester_in=1'b0;
  #10  manchester_in=1'b1;  //PAM 4=2

  #10  manchester_in=1'b1;
  #10  manchester_in=1'b0;
  #10  manchester_in=1'b1;
  #10  manchester_in=1'b0; //PAM 4=3

  #10  manchester_in=1'b0;
  #10  manchester_in=1'b1;
  #10  manchester_in=1'b0;
  #10  manchester_in=1'b1; //PAM 4=0

  #10  manchester_in=1'b0;
  #10  manchester_in=1'b1;
  #10  manchester_in=1'b1;
  #10  manchester_in=1'b0; //PAM 4=1
  #10;
  end

  rst = 1'b1;
  #20 rst=1'b0;

  repeat (15) begin //cycle thru every possible combination of four series inputs
    #10 manchester_in=a[3];
    #10 manchester_in=a[2];
```

```
    #10 manchester_in=a[1];
    #10 manchester_in=a[0];
    a=a+1;
  end

  #100 $stop; //close debug window to view waveform viewer
 end

//Step5 Display the results
initial begin //monitor counter value
  $display("________________________________output_PAM_out = -PAM4-" );
  $monitor("clk_in = %b: rst_in = %b:  output_PAM_out = %d " ,
  clk, rst, PAM_out);
 end
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 = 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
```

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