```
/*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;
```

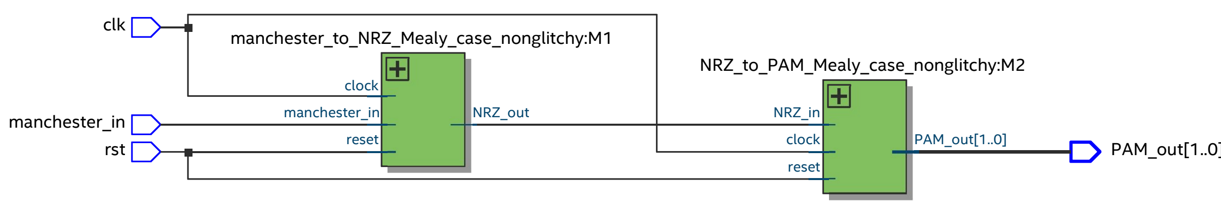
```
parameter S1010 = 4'b0011;
parameter S1011 = 4'b0100;
parameter S1100 = 4'b1100;
parameter S1101 = 4'b1101;

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;
                          PAM_out = 3'b000; end

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

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

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

endmodule


module manchester_to_PAM_Mealy_assign_nonglitchy (PAM_out,
                                                   manchester_in,
                                                  clock, reset);
output reg PAM_out;
input manchester_in, clock, reset;

reg [2:0] state; //the use of register assures no glitches
wire [2:0] next_state;
wire       next_out;

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

// sequential logic updating the state

always @ (posedge clock or posedge reset)  //asynchronous reset
  if (reset) begin state <= Sx;
                   PAM_out <= 3'b000;  end
  else       begin state <= next_state;
                   PAM_out <= next_out; end

// combinational logic 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 next_out = ~next_state[0] | (manchester_in & ~next_state[1]);

endmodule


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

reg [2:0] state, next_state;
reg       next_out, PAM_out; // assign values within always block

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

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

// Combinational logice 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;
                           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
```

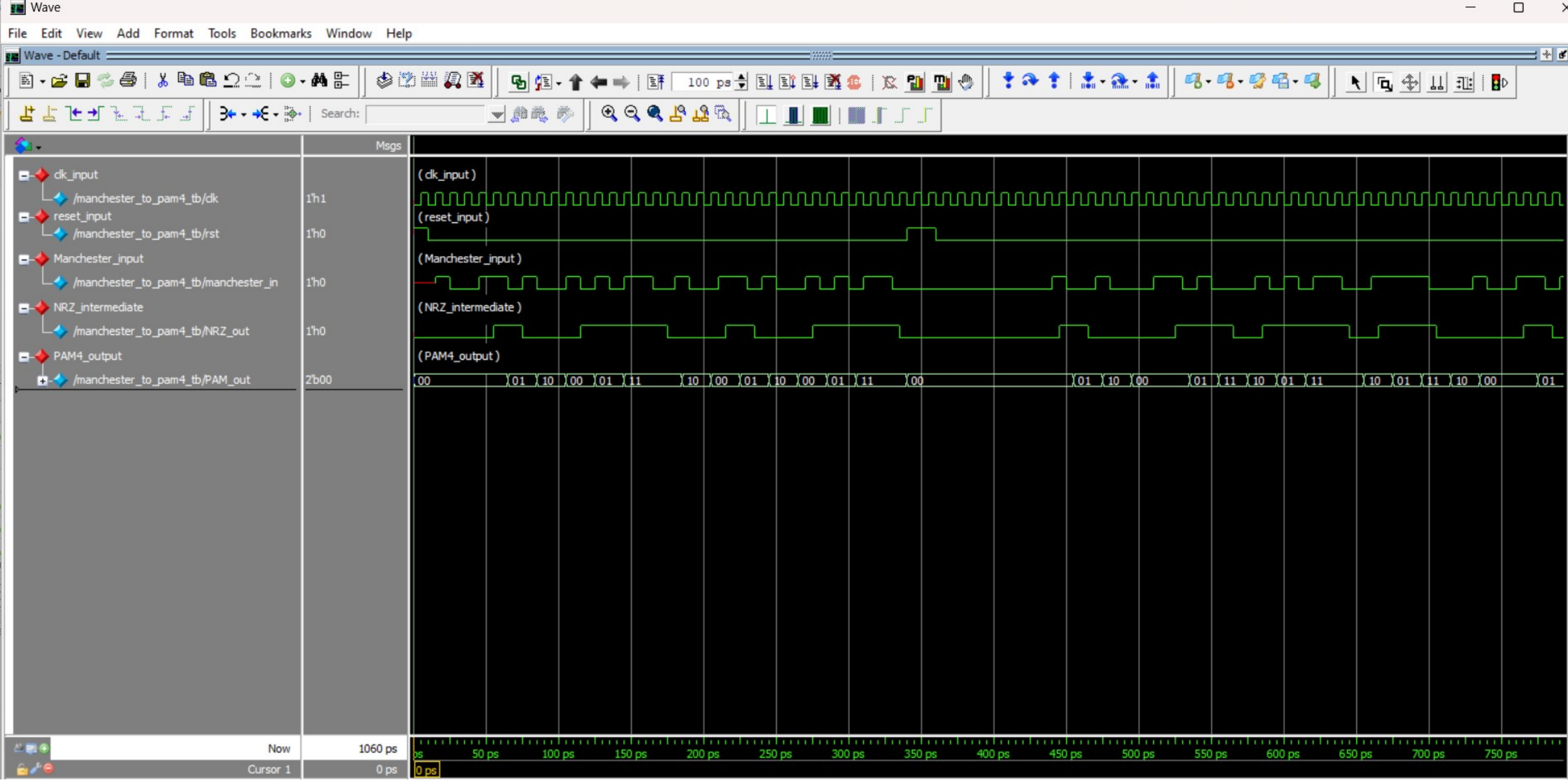
```
              else            begin
                              next_state = S0;
                              next_out = 1'b0; end

        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



```


clk
manchester_in
rst
manchester_to_NRZ_Mealy_case_nonglitchy:M1
clock
manchester_in
reset
NRZ_out
NRZ_to_PAM_Mealy_case_nonglitchy:M2
NRZ_in
clock
reset
PAM_out[1..0]
PAM_out[1..0]

```
//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
```



Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
clk_input
/manchester_to_pam4_tb/clk
1'h1
reset_input
/manchester_to_pam4_tb/rst
1'h0
Manchester_input
/manchester_to_pam4_tb/manchester_in
1'h0
NRZ_intermediate
/manchester_to_pam4_tb/NRZ_out
1'h0
PAM4_output
/manchester_to_pam4_tb/PAM_out
2'b00
(clk_input)
(reset_input)
(Manchester_input)
(NRZ_intermediate)
(PAM4_output)
00 01 10 00 01 11 10 00 01 10 00 01 11 00 01 10 00 01 11 10 01 11 10 01 11 10 00 01
Now
1060 ps
Cursor 1
0 ps
ps 50 ps 100 ps 150 ps 200 ps 250 ps 300 ps 350 ps 400 ps 450 ps 500 ps 550 ps 600 ps 650 ps 700 ps 750 ps
0 ps

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 = 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 = 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 = 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.