DataPath Controller - Design Example - NRZ to PAM8 Serial Code Line Conversion

Consider the following datapath module.

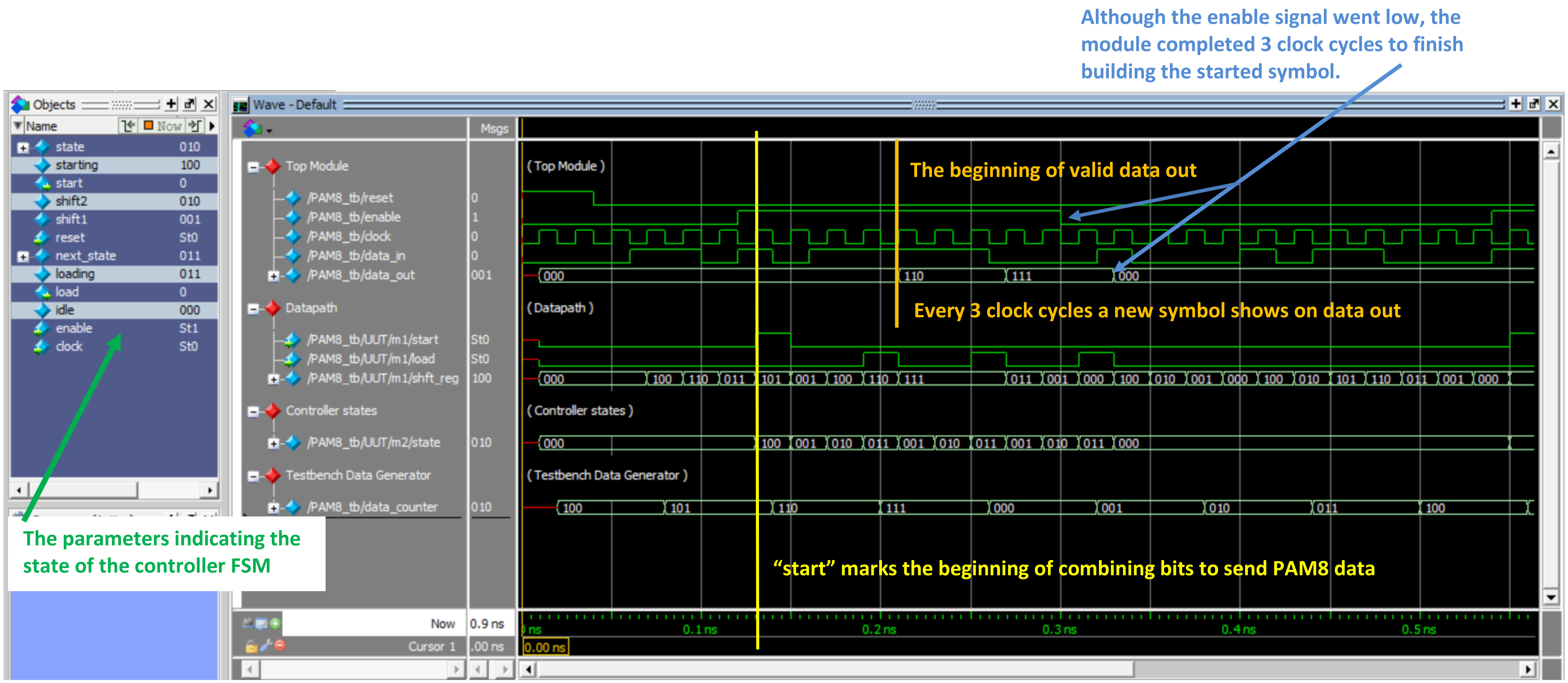
1. Draw the internal structure of the module showing the registers and explaining the idea of operation.
2. Design a controller module that would control the conversion from NRZ to PAM8. The top module providing access to the host should have a reset that would reset the output to zero and should have an enable input to start the conversion and transmission process.
   If the module is disabled during the construction of one 3-bit symbol, the controller should ignore the request until the symbol is completed.
   Draw a State Graph for the controller FSM.
3. Design the top module and sketch the block diagram for it.
4. Create a testbench that verifies the functionality of your design.

   Bonus: Modify your designs (datapath and/or controller) to down-sample the symbol rate by 2. This means that you would receive 3 bits to build one symbol and hold it while skipping the following three bits.

```
module PAM8_Datapath   (data_out, data_in, clock, reset, start, load);

output [2:0] data_out;
input        data_in, clock, reset, start, load;

reg    [2:0] shft_reg, out_reg;

assign data_out = out_reg;

always @ (posedge clock)

          if (reset) begin
                     shft_reg     <= 3'b0;
                     out_reg      <= 3'b0; end
     else if (start) begin
                     shft_reg     <= {data_in, shft_reg[1:0]};
                     out_reg      <= 3'b0; end
     else if (load)  begin
                     shft_reg     <= {data_in, shft_reg[2:1]};
                     out_reg      <= shft_reg; end
     else            shft_reg     <= {data_in, shft_reg[2:1]};

endmodule
```

```
module PAM8_Datapath_controller (data_in, data_out, reset, clock, enable);

input        data_in, reset, clock, enable;
output [2:0] data_out;

wire start, load;

PAM8_Datapath   m1   (data_out, data_in, clock, reset, start, load);

PAM8_Controller m2   (start, load, enable, reset, clock);

endmodule


module PAM8_Controller  (start, load, enable, reset, clock);

output reg   start, load;
input        enable, reset, clock;

reg [2:0]   state, next_state;

parameter   idle     = 3'b000;
parameter   starting = 3'b100;
parameter   shift1   = 3'b001;
parameter   shift2   = 3'b010;
parameter   loading  = 3'b011;

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

always @ *
    case (state)
      idle      : begin
                  start = 1'b0;
                  load  = 1'b0;
                  if (enable) next_state = starting;
                         else next_state = idle;        end
      starting : begin
                      start = 1'b1;         // The LSB of the first symbol is loaded,
                      load  = 1'b0;         // the output remains 0.
                      next_state = shift1;  end
      shift1   : begin
                        start = 1'b0;       // The 2nd bit is received
                        load  = 1'b0;
                        next_state = shift2;    end
      shift2   : begin
                        start = 1'b0;       // The MSB is received
                        load  = 1'b0;
                        next_state = loading;   end

      loading  : begin                      // The symbol is loaded to the output
                       start = 1'b0;        // and the next LSB is loaded
                       load  = 1'b1;
                       if(enable) next_state = shift1;
                           else   next_state = idle;      end
    endcase
endmodule
```

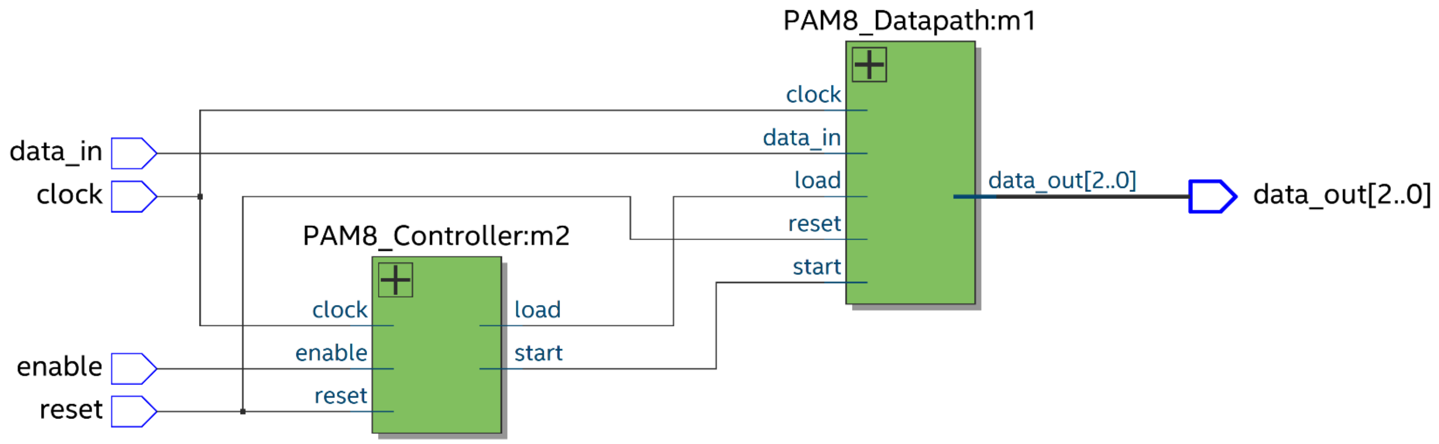
```
module PAM8_tb ();

reg clock, reset, enable;
reg data_in;

wire [2:0] data_out;

reg  [2:0] data_counter;

PAM8_Datapath_controller UUT (data_in, data_out, reset, clock, enable);

initial
begin
clock        = 1'b0;
reset        = 1'b1;
enable       = 1'b0;
data_in      = 1'b0;
end

always
#10 clock = ~clock;

initial
begin
#20   data_counter = 3'b101;
forever
#60  data_counter = data_counter + 1'b1;
end

always
begin
#20   data_in = data_counter [0];
#20   data_in = data_counter [1];
#20   data_in = data_counter [2];
end

initial
begin
#40  reset  = 1'b0;
#80  enable = 1'b1;
#180 enable = 1'b0;
#240 enable = 1'b1;
end

endmodule
```

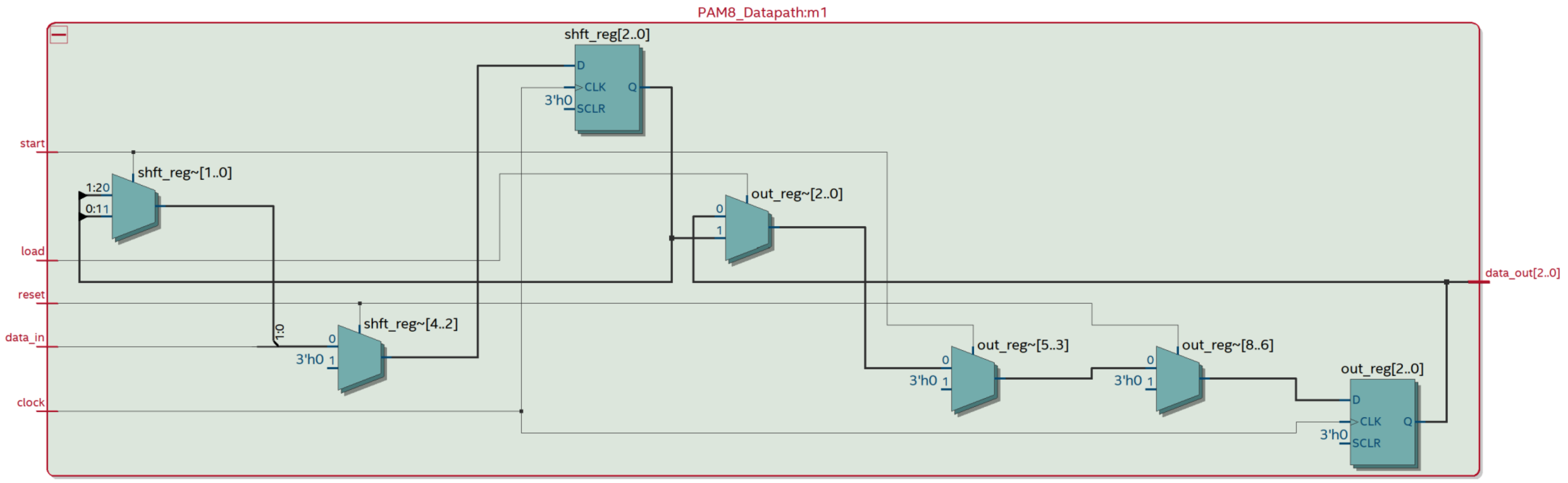


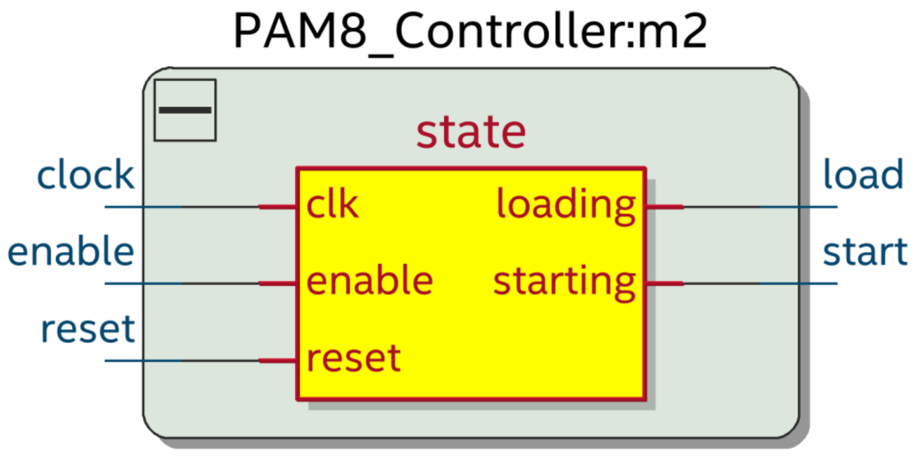
Name	
state	010
starting	100
start	0
shift2	010
shift1	001
reset	St0
next_state	011
loading	011
load	0
idle	000
enable	St1
clock	St0

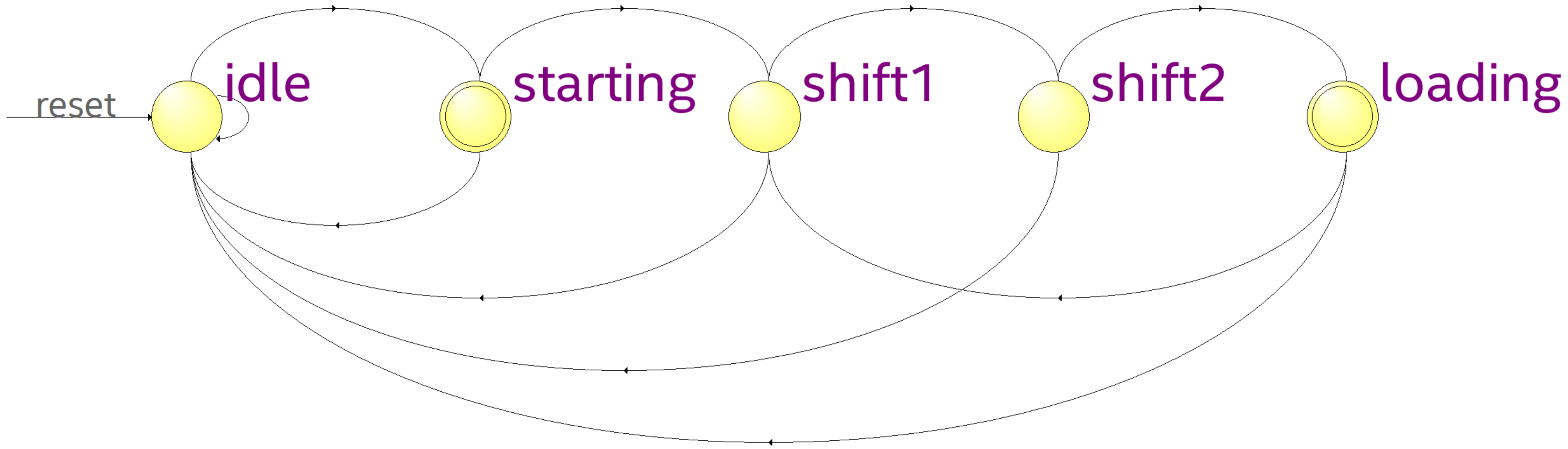
When the reset is active, the output and the shift register are both equal to zero. The reset has direct control over the datapath of the PAM8 module. When the reset is deactivated, while the enable is zero, the datapath load the data_in to the MSB of the shift register and load the data in the shift register to the right. The output register remains unchanged, and holds its value. The output register would only update with the value passed from the shift register when it receives a “load” command from the controller.

At the first positive clock edge, that has the reset inactive and the enable active, the FSM of the controller exits the idle state and enters the “starting” state, where the “start” signal is set to high and the data_in is copied to the MSB of the shift register, while holding the old value of the output register.


PAM8_Datapath:m1
clock
data_in
load
reset
start
data_out[2..0]
PAM8_Controller:m2
clock
enable
reset
load
start
data_in
clock
enable
reset
data_out[2..0]


PAM8_Datapath:m1
shft_reg[2..0]
D
CLK
Q
3'h0
SCLR
start
shft_reg~[1..0]
1:2
0:1
load
out_reg~[2..0]
reset
data_in
shft_reg~[4..2]
1:0
3'h0
clock
out_reg~[5..3]
out_reg~[8..6]
out_reg[2..0]
data_out[2..0]

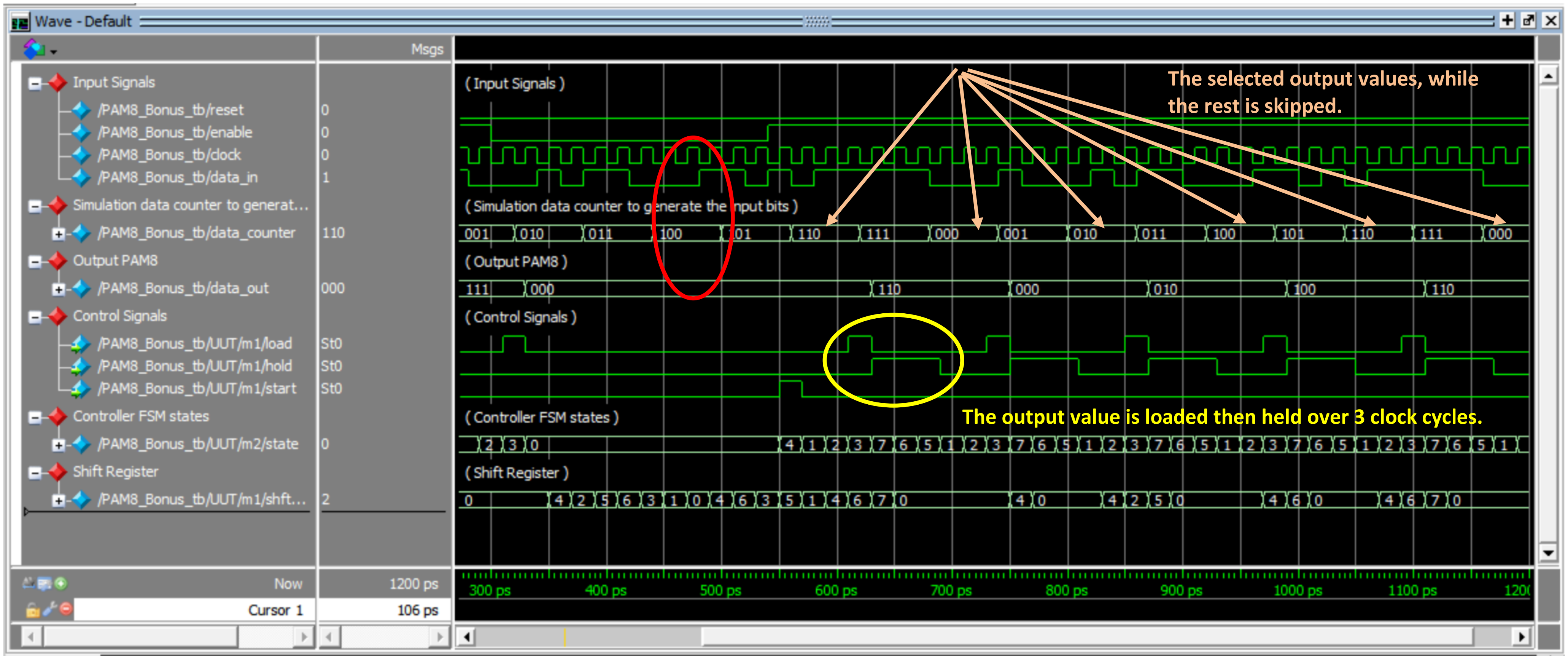

PAM8_Controller:m2
state
clock
enable
reset
clk
enable
reset
loading
starting
load
start


reset
idle
starting
shift1
shift2
loading

Bonus Question:

```
/*-----------------------------------------------------------------------------------------
    Bonus Question:
    ---------------
    Modify your designs (datapath and/or controller) to down-sample the symbol rate by 2.
    This means that you would receive 3 bits to build one symbol and hold it while skipping the
    following three bits.
-----------------------------------------------------------------------------------------*/


    module PAM8_Datapath_controller_Bonus (data_in, data_out, reset, clock, enable);

    input        data_in, reset, clock, enable;
    output [2:0] data_out;

    wire start, load;

    PAM8_Datapath_Bonus    m1   (data_out, data_in, clock, reset, start, load, hold);

    PAM8_Controller_Bonus m2    (start, load, enable, reset, clock, hold);

    endmodule


    module PAM8_Datapath_Bonus   (data_out, data_in, clock, reset, start, load, hold);

    output [2:0] data_out;
    input        data_in, clock, reset, start, load;
    input        hold;

    reg    [2:0] shft_reg, out_reg;

    assign data_out = out_reg;

    always @ (posedge clock)

              if (reset) begin
                         shft_reg     <= 3'b0;
                         out_reg      <= 3'b0; end
         else if (start) begin
                         shft_reg     <= {data_in, shft_reg[1:0]};
                         out_reg      <= 3'b0; end
         else if (load)  begin
                         shft_reg     <= {data_in, shft_reg[2:1]};
                         out_reg      <= shft_reg; end
         else if (hold)  begin
                         out_reg      <= out_reg;
                         shft_reg     <= 3'b0; end
         else            shft_reg     <= {data_in, shft_reg[2:1]};

    endmodule


    module PAM8_Controller_Bonus  (start, load, enable, reset, clock, hold);

    output reg    start, load, hold;
    input         enable, reset, clock;

    reg [2:0]   state, next_state;

    parameter   idle     = 3'b000;
    parameter   starting = 3'b100;
    parameter   shift1   = 3'b001;
    parameter   shift2   = 3'b010;
    parameter   loading  = 3'b011;

    parameter   skip1    = 3'b111;
    parameter   skip2    = 3'b110;
    parameter   skip3    = 3'b101;

    always @ (posedge clock)
    if (reset)  state <= idle;
          else state  <= next_state;
```

```
always @ *

    begin
    start = 1'b0;
    load  = 1'b0;
    hold  = 1'b0;

    case (state)
      idle     : begin
                 if (enable) next_state = starting;
                        else next_state = idle;        end
      starting : begin
                     start = 1'b1;         // The LSB of the first symbol is loaded,
                     next_state = shift1;   end
      shift1   :     next_state = shift2;
      shift2   :     next_state = loading;
      loading  : begin                     // The symbol is loaded to the output
                     load  = 1'b1;
                      if(enable) next_state = skip1;
                          else   next_state = idle;      end
      skip1    : begin
                     hold = 1'b1;
                     next_state = skip2; end
      skip2    : begin
                     hold = 1'b1;
                     next_state = skip3; end
      skip3    : begin
                     hold = 1'b1;
                     if(enable) next_state = shift1;
                          else   next_state = idle;      end
     default   : next_state = idle;

     endcase
    end
endmodule


module PAM8_Bonus_tb ();

reg clock, reset, enable;
reg data_in;

wire [2:0] data_out;

reg  [2:0] data_counter;

PAM8_Datapath_controller_Bonus UUT (data_in, data_out, reset, clock, enable);

initial
begin
clock        = 1'b0;
reset        = 1'b1;
enable       = 1'b0;
data_in      = 1'b0;
end

always
#10 clock = ~clock;

initial
begin
#20   data_counter = 3'b101;
forever
#60  data_counter = data_counter + 1'b1;
end

always
begin
#20   data_in = data_counter [0];
#20   data_in = data_counter [1];
#20   data_in = data_counter [2];
end

initial
begin
#40  reset  = 1'b0;
#80  enable = 1'b1;
#180 enable = 1'b0;
#240 enable = 1'b1;
end

endmodule
```



Input Signals	
/PAM8_Bonus_tb/reset	0
/PAM8_Bonus_tb/enable	0
/PAM8_Bonus_tb/clock	0
/PAM8_Bonus_tb/data_in	1
Simulation data counter to generat...	
/PAM8_Bonus_tb/data_counter	110
Output PAM8	
/PAM8_Bonus_tb/data_out	000
Control Signals	
/PAM8_Bonus_tb/UUT/m1/load	St0
/PAM8_Bonus_tb/UUT/m1/hold	St0
/PAM8_Bonus_tb/UUT/m1/start	St0
Controller FSM states	
/PAM8_Bonus_tb/UUT/m2/state	0
Shift Register	
/PAM8_Bonus_tb/UUT/m1/shft...	2
Now	1200 ps
Cursor 1	106 ps

The selected output values, while the rest is skipped.

The output value is loaded then held over 3 clock cycles.