Introduction and Theory:

The purpose of this lab is to understand and modify a Datapath-controller UART design. A testbench will be created to test the functionality of the UART top module and then odd parity will be implemented to the Datapath.

Experimental Methods:

1. **Create a testbench for the given UART design and verify the functionality of the datapath and the controller modules.**

The test bench implemented into the UART code provided is found highlighted red on page 4 in the appendix. The waveforms from the test bench are shown in **Figure 1**. In order to properly sequence the UART Datapath-controller, the data byte is input when the LOAD TX DATA goes high for one clock cycle. This loads the input data to the TX Data register. Next clock cycle, the Byte Ready goes high for one clock cycle placing controller into IDLE state. Finally, the TRANSMIT BYTE goes high starting the UART transmission. In this testbench, two bytes were sent in succession to prove continuous operation. 0x55 and 0xBA were transmitted successfully as shown in the SERIAL DATA OUT waveform in **Figure 1**. The Start bit goes low telling the receiver that data is next. Then the data bits are sent in LSB to MSB order. (Big Endian)

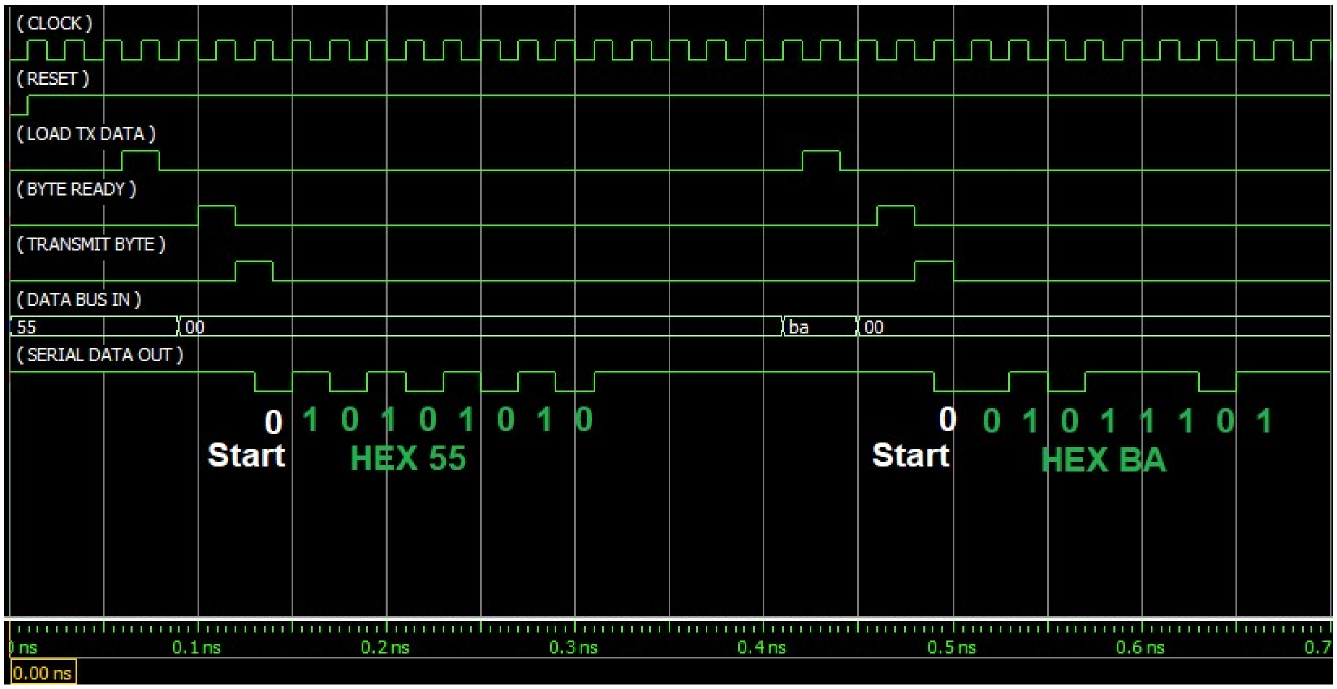


Figure 1: UART Testbench Part 1

2. Modify the design to include an odd parity bit in the transmission data.

The odd parity was implemented on in the Datapath module. A wire (odd_parity) was added and TX_datareg, TX_shiftreg were expanded by one bit in the code shown in orange on appendix code page 3. Odd_parity is continually assigned to the summation of all the data bits. Even number of bits will be parity 0, and odd number bits will be 1. Finally, the data shifting needed adjusted for the extra bit. That change is shown in appendix page 4 code in orange. The results from using the same testbench are shown in **Figure 2**. The testbench provided a byte with even and odd parity with the parity bits shown in red. Comparing **Figure 1** to **Figure 2** shows the proper implementation of odd parity.

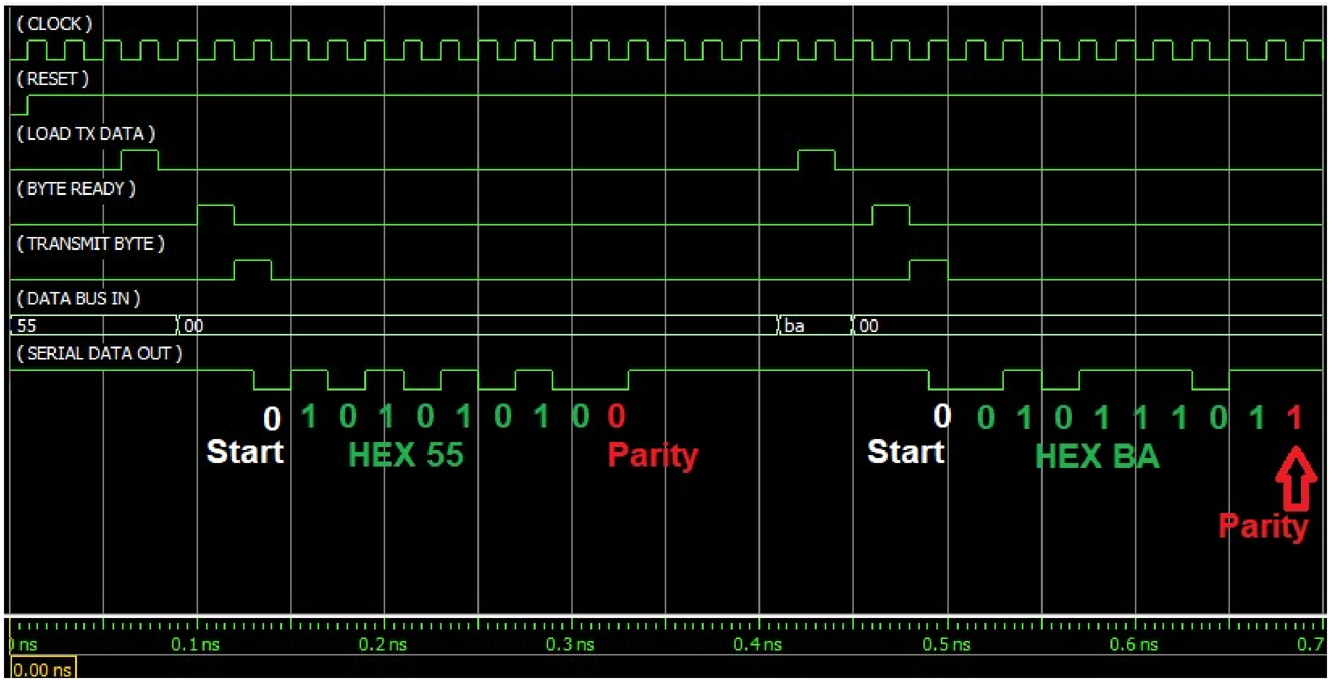
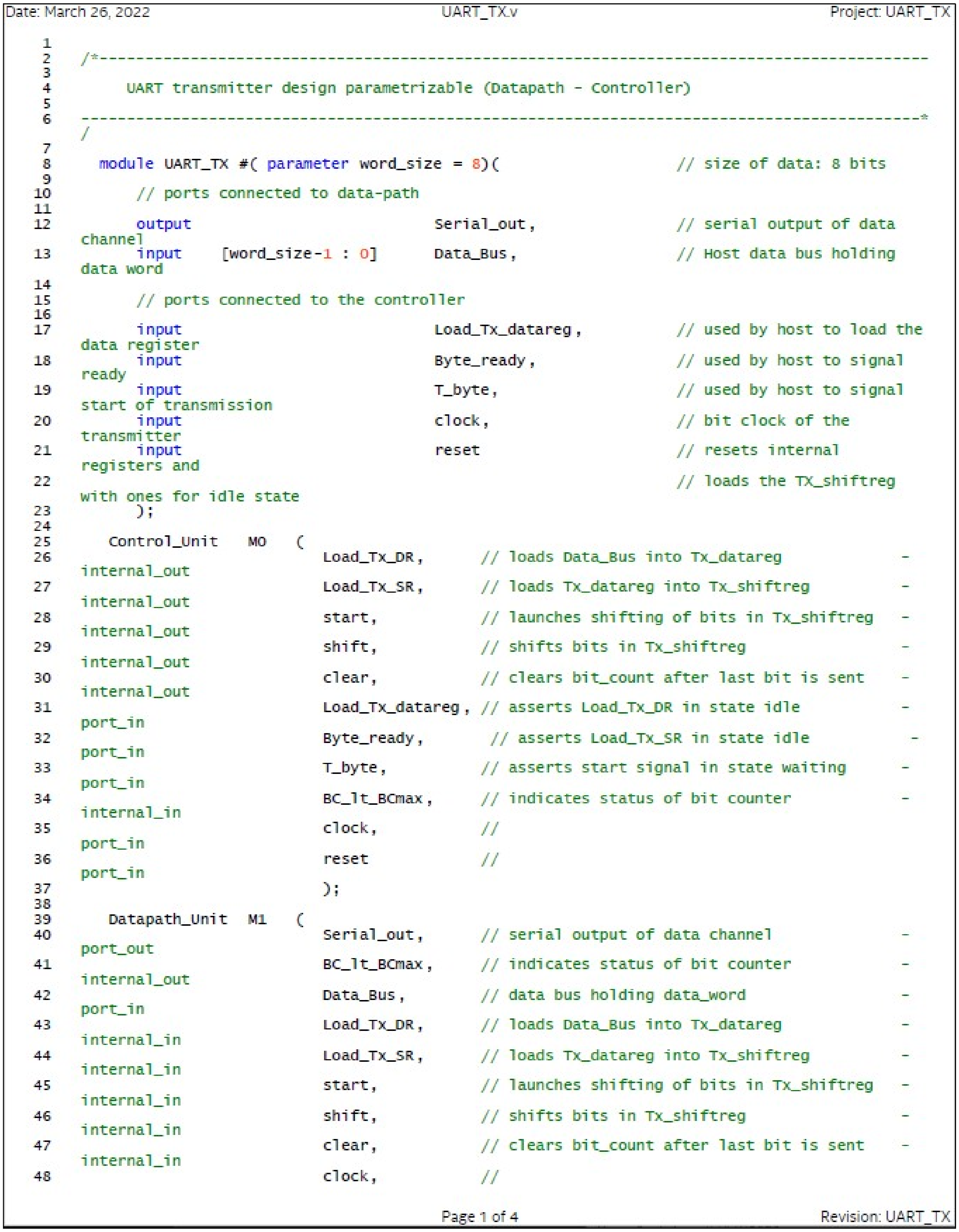


Figure 2: UART Testbench Part 2

Conclusion:

In this lab, a testbench was created to test the functionality of the UART Datapath-controller provided in the lab handout. **Figure 1** clearly shows the sequencing of the control lines, input data, and the serial UART output data. The same testbench was used after the odd parity bit was added to the datapath module. **Figure 2** also shows how the parity bit was implemented properly adding a bit to the UART transmission. Hexadecimal values of 0x55 and 0xBA were used to test the parity because 0x55 is even while 0xBA is odd.

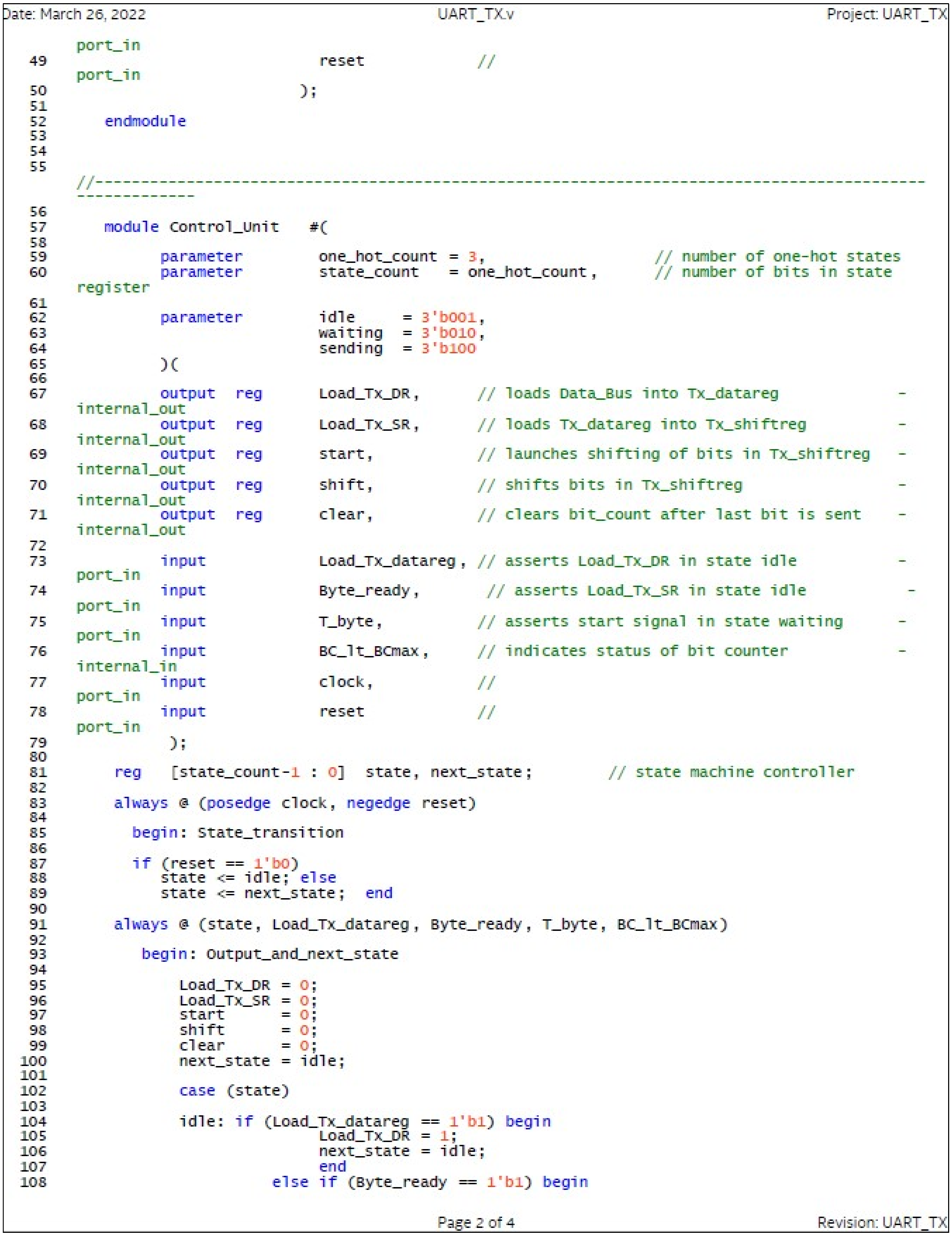
Appendix:



Date: March 26, 2022 UART_TX.v Project: UART_TX

```

/*--------------------------------------------------------------------------------------

     UART transmitter design parametrizable (Datapath - Controller)

--------------------------------------------------------------------------------------*/

  module UART_TX #( parameter word_size = 8)(                // size of data: 8 bits

      // ports connected to data-path

      output                           Serial_out,           // serial output of data channel
      input     [word_size-1 : 0]      Data_Bus,             // Host data bus holding data word

      // ports connected to the controller

      input                            Load_Tx_datareg,      // used by host to load the data register
      input                            Byte_ready,           // used by host to signal ready
      input                            T_byte,               // used by host to signal start of transmission
      input                            clock,                // bit clock of the transmitter
      input                            reset                 // resets internal registers and
                                                             // loads the TX_shiftreg with ones for idle state
      );

     Control_Unit   M0   (  Load_Tx_DR,      // loads Data_Bus into Tx_datareg          - internal_out
                            Load_Tx_SR,      // loads Tx_datareg into Tx_shiftreg       - internal_out
                            start,           // launches shifting of bits in Tx_shiftreg - internal_out
                            shift,           // shifts bits in Tx_shiftreg              - internal_out
                            clear,           // clears bit_count after last bit is sent  - internal_out
                            Load_Tx_datareg, // asserts Load_Tx_DR in state idle         - port_in
                            Byte_ready,       // asserts Load_Tx_SR in state idle        - port_in
                            T_byte,          // asserts start signal in state waiting    - port_in
                            BC_lt_BCmax,     // indicates status of bit counter          - internal_in
                            clock,           //                                           port_in
                            reset            //                                           port_in
                            );

     Datapath_Unit  M1   (  Serial_out,      // serial output of data channel            - port_out
                            BC_lt_BCmax,     // indicates status of bit counter          - internal_out
                            Data_Bus,        // data bus holding data_word               - port_in
                            Load_Tx_DR,      // loads Data_Bus into Tx_datareg           - internal_in
                            Load_Tx_SR,      // loads Tx_datareg into Tx_shiftreg        - internal_in
                            start,           // launches shifting of bits in Tx_shiftreg - internal_in
                            shift,           // shifts bits in Tx_shiftreg               - internal_in
                            clear,           // clears bit_count after last bit is sent   - internal_in
                            clock,           //
```

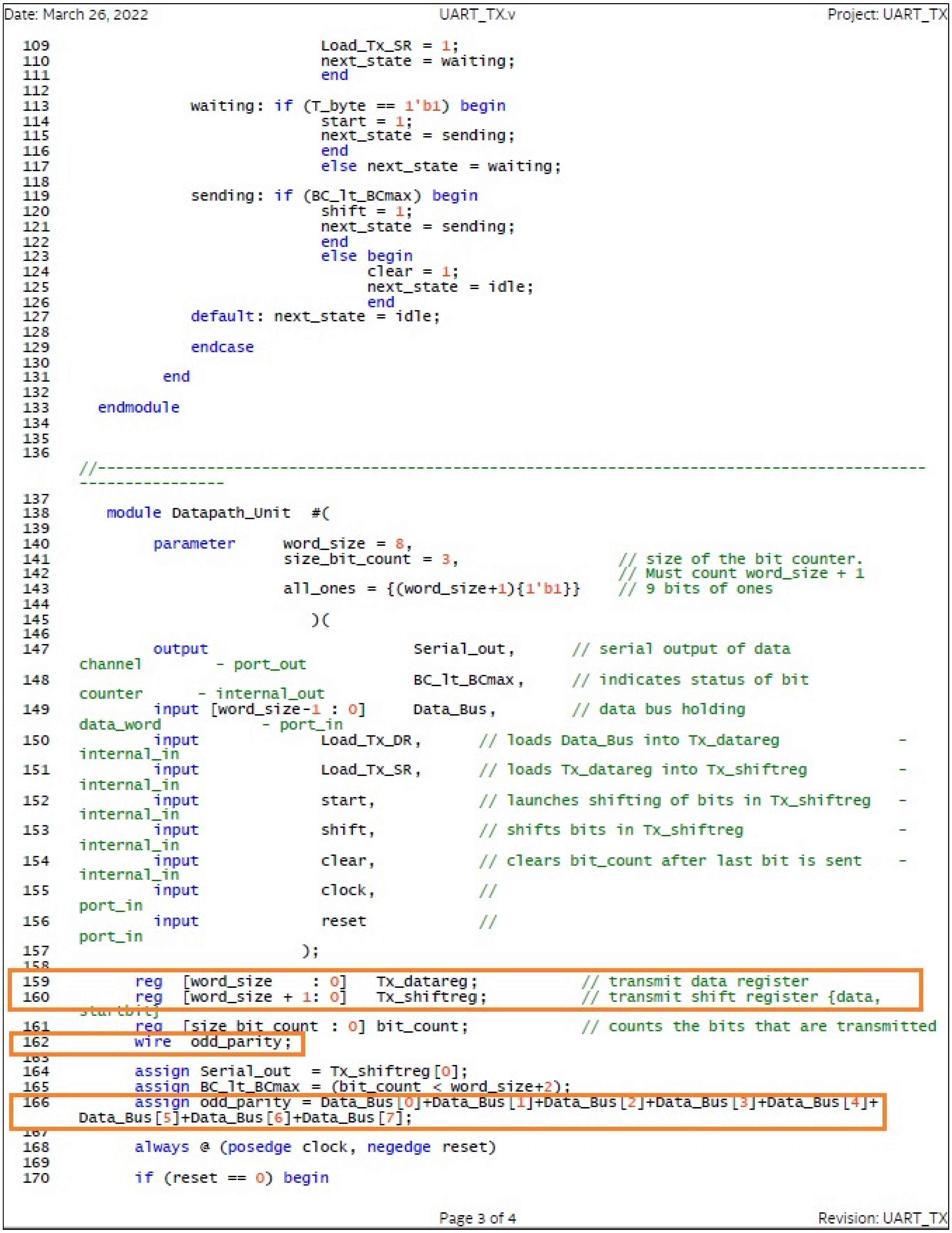
Page 1 of 4 Revision: UART_TX



Date: March 26, 2022 UART_TX.v Project: UART_TX

```
port_in
                           reset           //
port_in
                         );

   endmodule



//----------------------------------------------------------------------------------------
-------------

   module Control_Unit   #(

         parameter        one_hot_count = 3,           // number of one-hot states
         parameter        state_count   = one_hot_count,      // number of bits in state
register

         parameter        idle     = 3'b001,
                          waiting  = 3'b010,
                          sending  = 3'b100
         )(

         output  reg      Load_Tx_DR,      // loads Data_Bus into Tx_datareg          -
internal_out
         output  reg      Load_Tx_SR,      // loads Tx_datareg into Tx_shiftreg       -
internal_out
         output  reg      start,           // launches shifting of bits in Tx_shiftreg  -
internal_out
         output  reg      shift,           // shifts bits in Tx_shiftreg              -
internal_out
         output  reg      clear,           // clears bit_count after last bit is sent   -
internal_out

         input            Load_Tx_datareg, // asserts Load_Tx_DR in state idle        -
port_in
         input            Byte_ready,       // asserts Load_Tx_SR in state idle        -
port_in
         input            T_byte,          // asserts start signal in state waiting     -
port_in
         input            BC_lt_BCmax,     // indicates status of bit counter         -
internal_in
         input            clock,           //
port_in
         input            reset            //
port_in
          );

    reg   [state_count-1 : 0]  state, next_state;        // state machine controller

    always @ (posedge clock, negedge reset)

      begin: State_transition

      if (reset == 1'b0)
         state <= idle; else
         state <= next_state;  end

    always @ (state, Load_Tx_datareg, Byte_ready, T_byte, BC_lt_BCmax)

       begin: Output_and_next_state

           Load_Tx_DR = 0;
           Load_Tx_SR = 0;
           start      = 0;
           shift      = 0;
           clear      = 0;
           next_state = idle;

           case (state)

           idle: if (Load_Tx_datareg == 1'b1) begin
                        Load_Tx_DR = 1;
                        next_state = idle;
                        end
                     else if (Byte_ready == 1'b1) begin
```

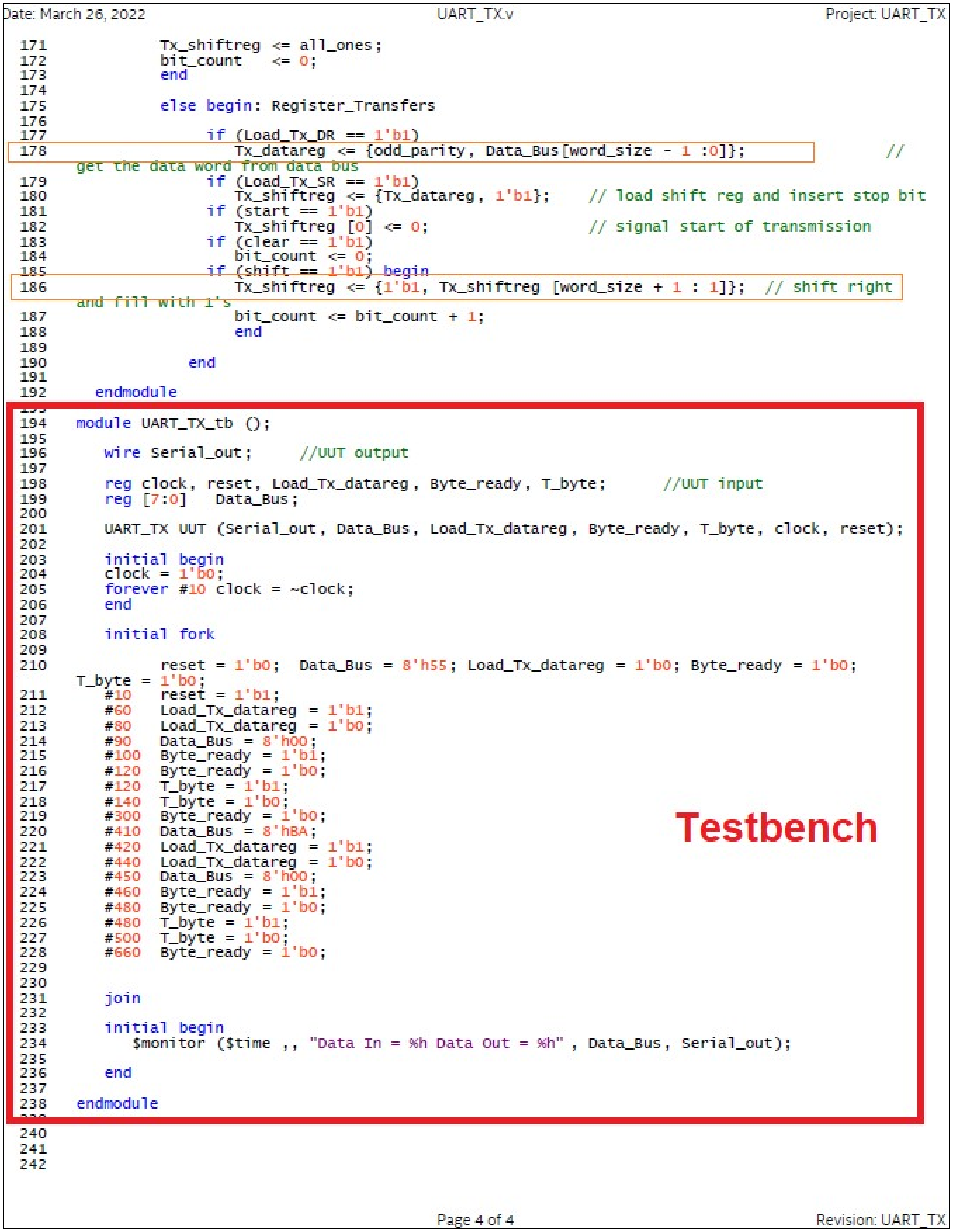
Page 2 of 4 Revision: UART_TX



Date: March 26, 2022   UART_TX.v   Project: UART_TX

```
                            Load_Tx_SR = 1;
                            next_state = waiting;
                            end

              waiting: if (T_byte == 1'b1) begin
                            start = 1;
                            next_state = sending;
                            end
                            else next_state = waiting;

              sending: if (BC_lt_BCmax) begin
                            shift = 1;
                            next_state = sending;
                            end
                            else begin
                                 clear = 1;
                                 next_state = idle;
                                 end
              default: next_state = idle;

              endcase

           end

   endmodule



//------------------------------------------------------------------------------------------
----------------

    module Datapath_Unit  #(

         parameter      word_size = 8,
                        size_bit_count = 3,              // size of the bit counter.
                                                         // Must count word_size + 1
                        all_ones = {(word_size+1){1'b1}}   // 9 bits of ones

                           )(

         output                       Serial_out,      // serial output of data
channel        - port_out
                                      BC_lt_BCmax,     // indicates status of bit
counter       - internal_out
         input [word_size-1 : 0]      Data_Bus,        // data bus holding
data_word            - port_in
         input              Load_Tx_DR,      // loads Data_Bus into Tx_datareg          -
internal_in
         input              Load_Tx_SR,      // loads Tx_datareg into Tx_shiftreg       -
internal_in
         input              start,           // launches shifting of bits in Tx_shiftreg  -
internal_in
         input              shift,           // shifts bits in Tx_shiftreg              -
internal_in
         input              clear,           // clears bit_count after last bit is sent    -
internal_in
         input              clock,           //
port_in
         input              reset            //
port_in
                           );

        reg  [word_size    : 0]    Tx_datareg;           // transmit data register
        reg  [word_size + 1: 0]    Tx_shiftreg;          // transmit shift register {data,
startbit}
        reg  [size_bit_count : 0] bit_count;             // counts the bits that are transmitted
        wire  odd_parity;

        assign Serial_out  = Tx_shiftreg[0];
        assign BC_lt_BCmax  = (bit_count < word_size+2);
        assign odd_parity = Data_Bus[0]+Data_Bus[1]+Data_Bus[2]+Data_Bus[3]+Data_Bus[4]+
Data_Bus[5]+Data_Bus[6]+Data_Bus[7];

        always @ (posedge clock, negedge reset)

        if (reset == 0) begin
```

Page 3 of 4   Revision: UART_TX



```
            Tx_shiftreg  <= all_ones;
            bit_count    <= 0;
            end

            else begin: Register_Transfers

                if (Load_Tx_DR == 1'b1)
                   Tx_datareg <= {odd_parity, Data_Bus[word_size - 1 :0]};          // get the data word from data bus
                if (Load_Tx_SR == 1'b1)
                   Tx_shiftreg <= {Tx_datareg, 1'b1};    // load shift reg and insert stop bit
                if (start == 1'b1)
                   Tx_shiftreg [0] <= 0;                 // signal start of transmission
                if (clear == 1'b1)
                   bit_count <= 0;
                if (shift == 1'b1) begin
                   Tx_shiftreg <= {1'b1, Tx_shiftreg [word_size + 1 : 1]};  // shift right and fill with 1's
                   bit_count <= bit_count + 1;
                   end

              end

    endmodule

  module UART_TX_tb ();

     wire Serial_out;     //UUT output

     reg clock, reset, Load_Tx_datareg, Byte_ready, T_byte;      //UUT input
     reg [7:0]   Data_Bus;

     UART_TX UUT (Serial_out, Data_Bus, Load_Tx_datareg, Byte_ready, T_byte, clock, reset);

     initial begin
     clock = 1'b0;
     forever #10 clock = ~clock;
     end

     initial fork

           reset = 1'b0;  Data_Bus = 8'h55; Load_Tx_datareg = 1'b0; Byte_ready = 1'b0; T_byte = 1'b0;
     #10   reset = 1'b1;
     #60   Load_Tx_datareg  = 1'b1;
     #80   Load_Tx_datareg  = 1'b0;
     #90   Data_Bus = 8'h00;
     #100  Byte_ready = 1'b1;
     #120  Byte_ready = 1'b0;
     #120  T_byte = 1'b1;
     #140  T_byte = 1'b0;
     #300  Byte_ready = 1'b0;
     #410  Data_Bus = 8'hBA;
     #420  Load_Tx_datareg  = 1'b1;
     #440  Load_Tx_datareg  = 1'b0;
     #450  Data_Bus = 8'h00;
     #460  Byte_ready = 1'b1;
     #480  Byte_ready = 1'b0;
     #480  T_byte = 1'b1;
     #500  T_byte = 1'b0;
     #660  Byte_ready = 1'b0;


     join

     initial begin
       $monitor ($time ,, "Data In = %h Data Out = %h" , Data_Bus, Serial_out);

     end

  endmodule
```