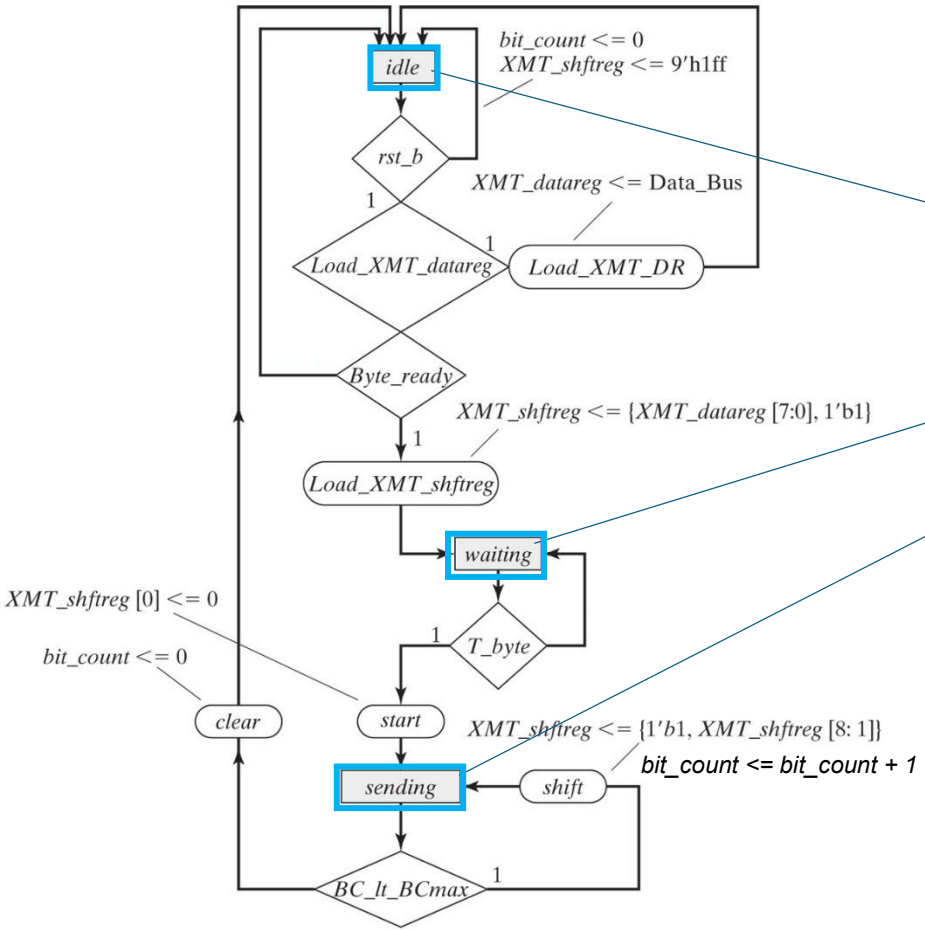
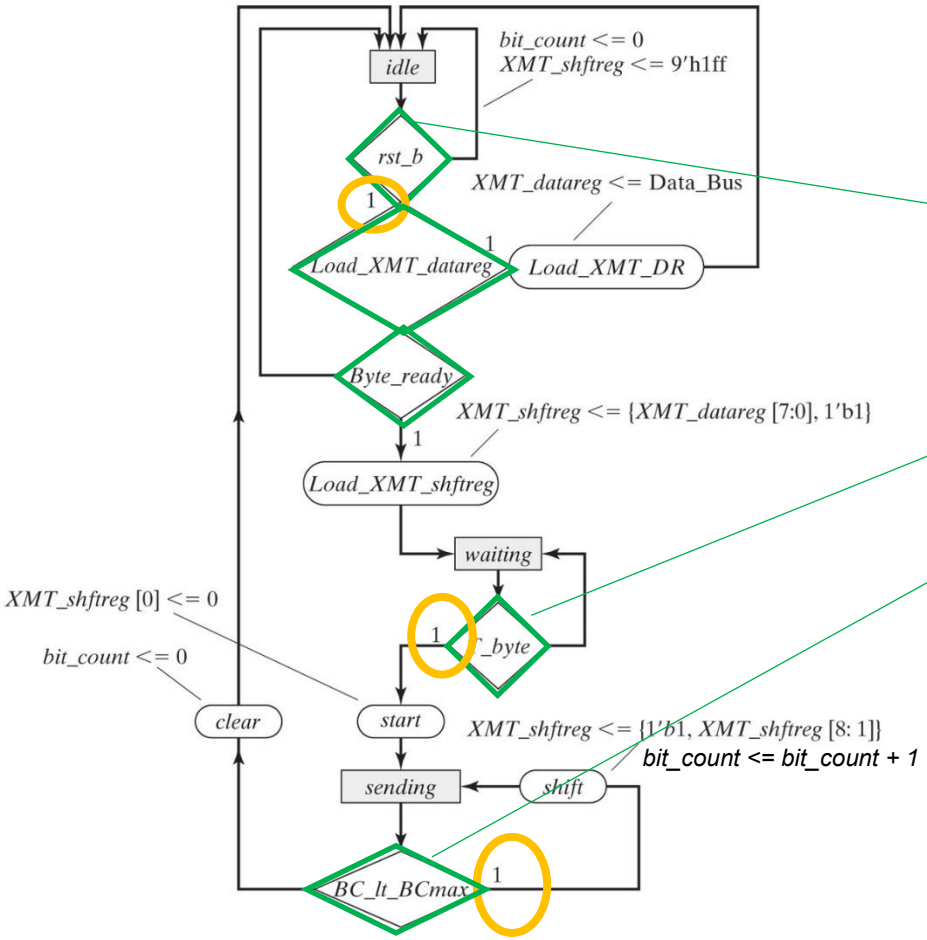
Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Controller States



States of the controller are enclosed in the rectangular (shaded) box. The controller has 3 states: idle, waiting, and sending.

Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.

Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Controller Inputs

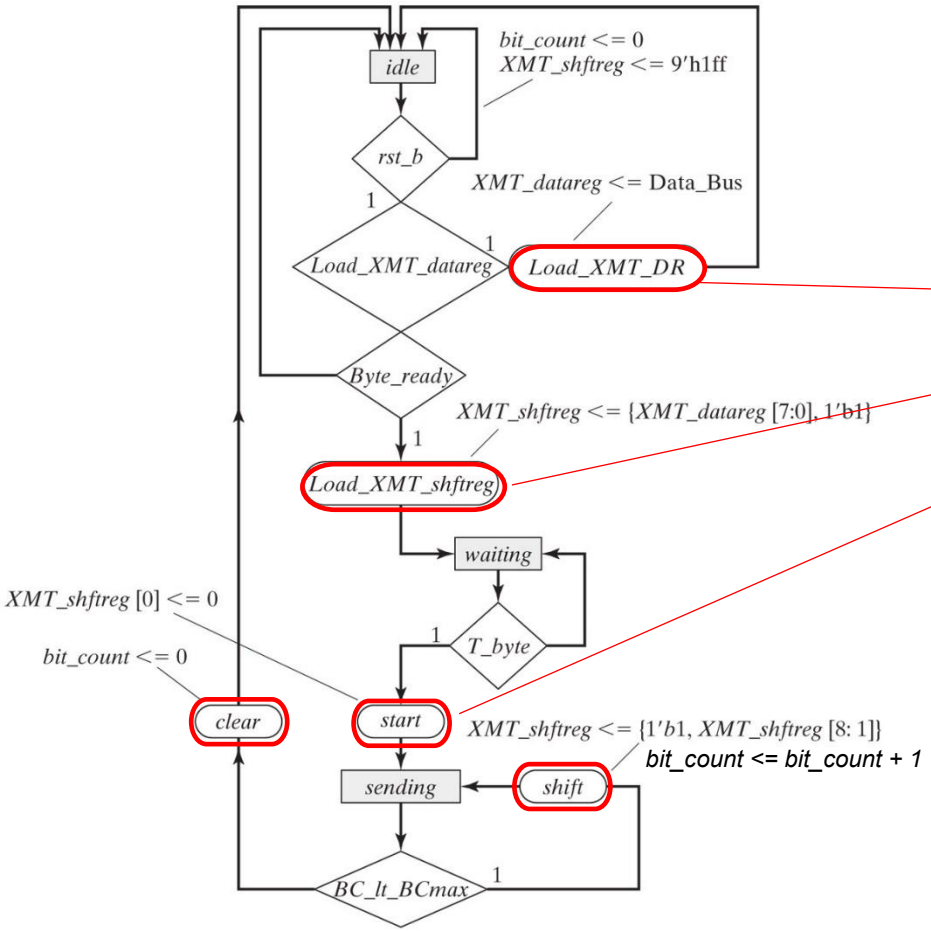


Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.

Diamond shapes are used for checking the value of a controller **input**. Controller inputs are always single bits signals, so they can be either '1' or '0'. Based on their logic value the transition arrow destination state and controller outputs are determined. The path followed in case the input is a logic '1' is annotated with a **1**. Otherwise the path has no label.

The inputs to the controller can be coming from the user as an overall input to the top module (Byte_ready), or they can be internal signals generated by the DataPath (BC_lt_BCMax).

Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Controller Outputs



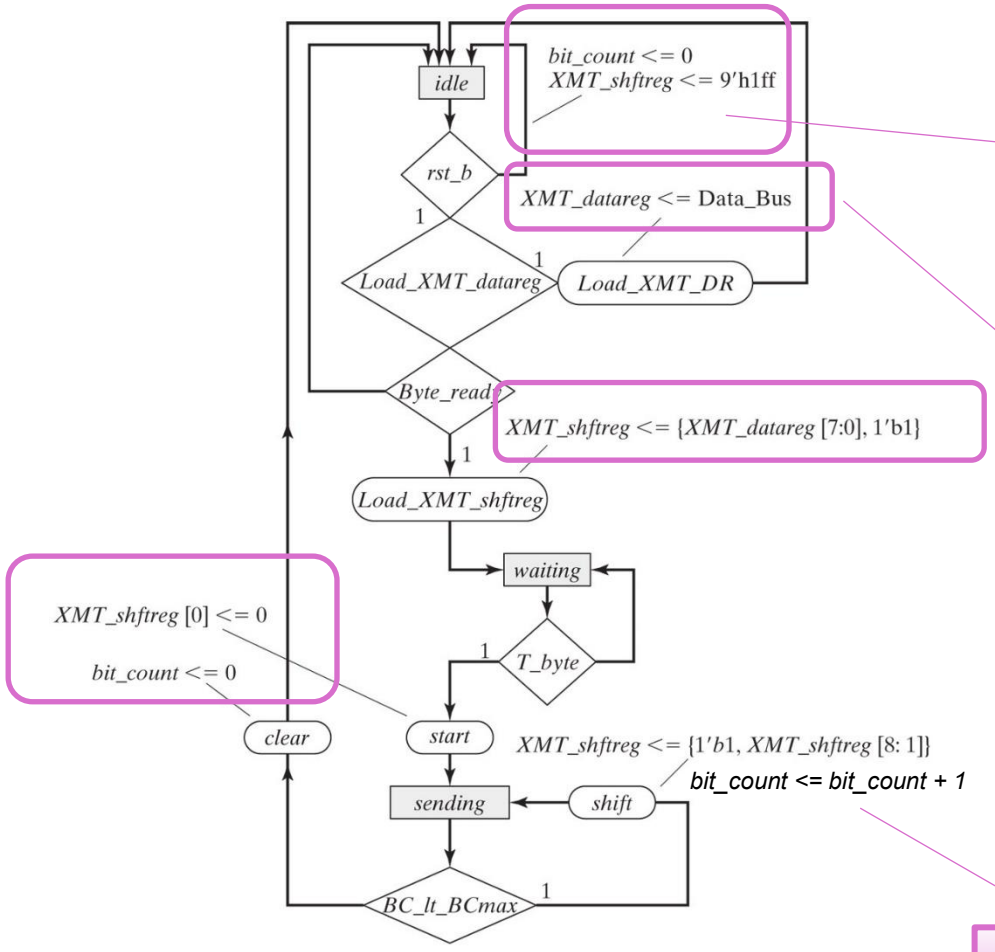
Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.

Controller outputs are single bit signals. They appear on the transition arrows if they will be set to a logic '1'.

They are enclosed in a rectangular shape with two rounded sides.

Output signals not featured on the transition line are just kept inactive (0 level).

Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: DataPath Registers



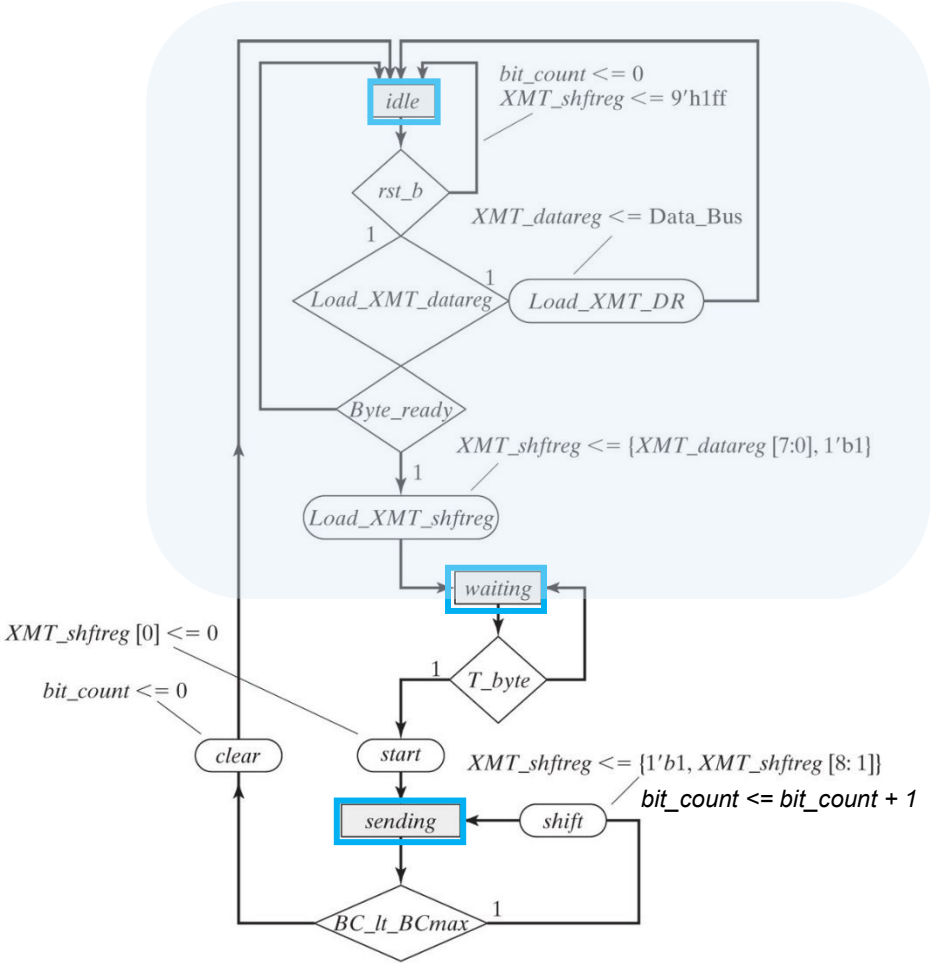
Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.

The registers in the DataPath also appear on the ASMD graph. When the rst_b is '0' (active low) the controller transitions to the idle state and the rst_b and the bit_count register becomes 0 and the XMT_shftreg is filled with all ones.

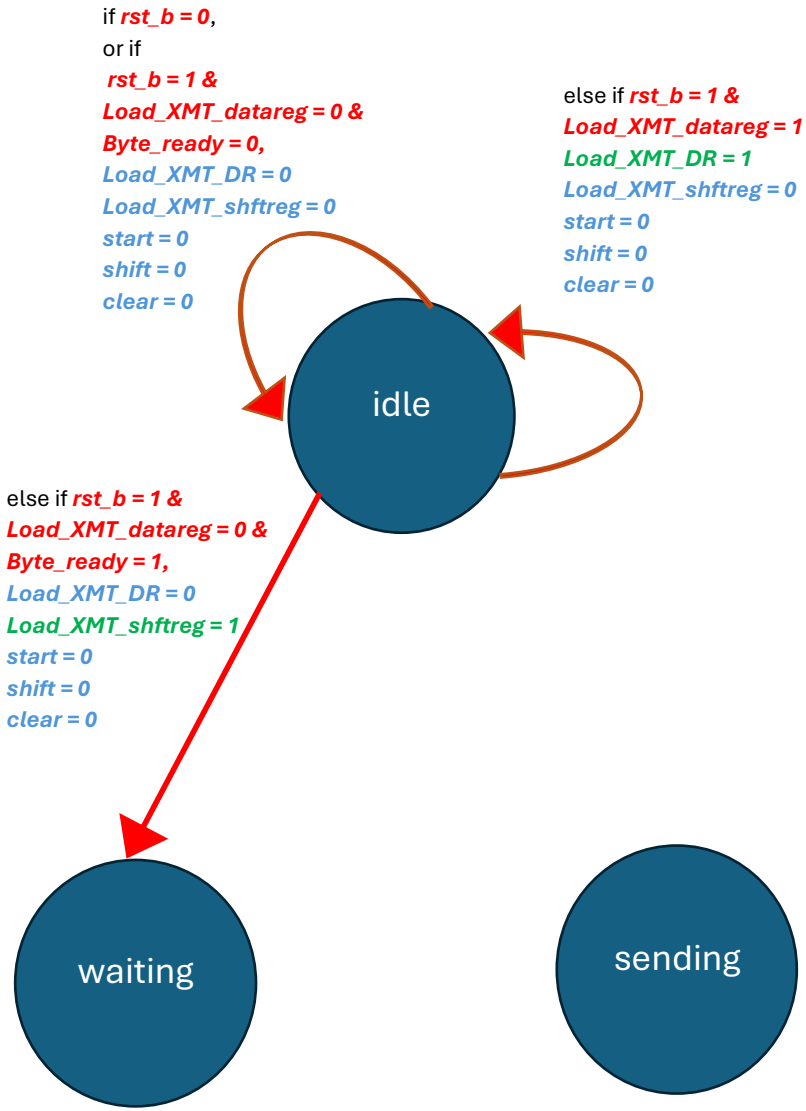
When the controller output (acting as an input to the DataPath) is raised to a logic 1, the XMT_datareg will be loaded by the Data_Bus value. See how the line is connected between the assignment of the XMT_datareg and the controller output box.

When the shift output is 1, the XMT_shftreg will have its content shifted to the right and the MSB filled with a 1. Also with an active shift the bit_count register in the DataPath will increment by 1.

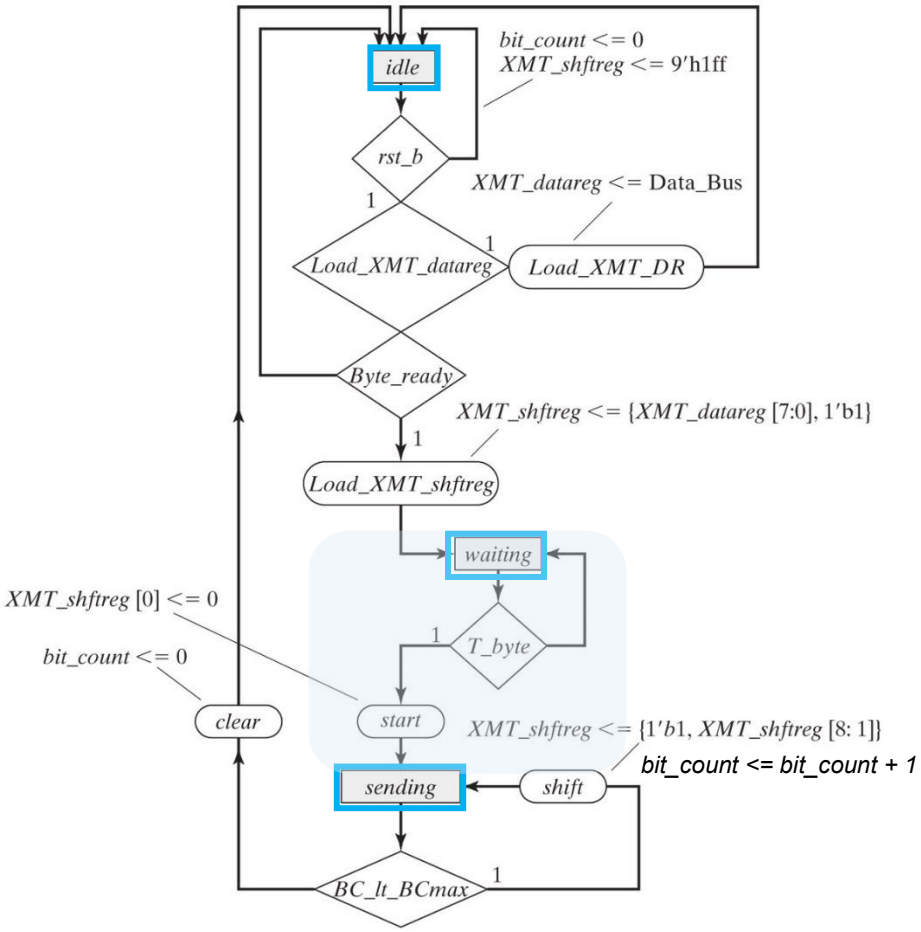
Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Generating the FSM



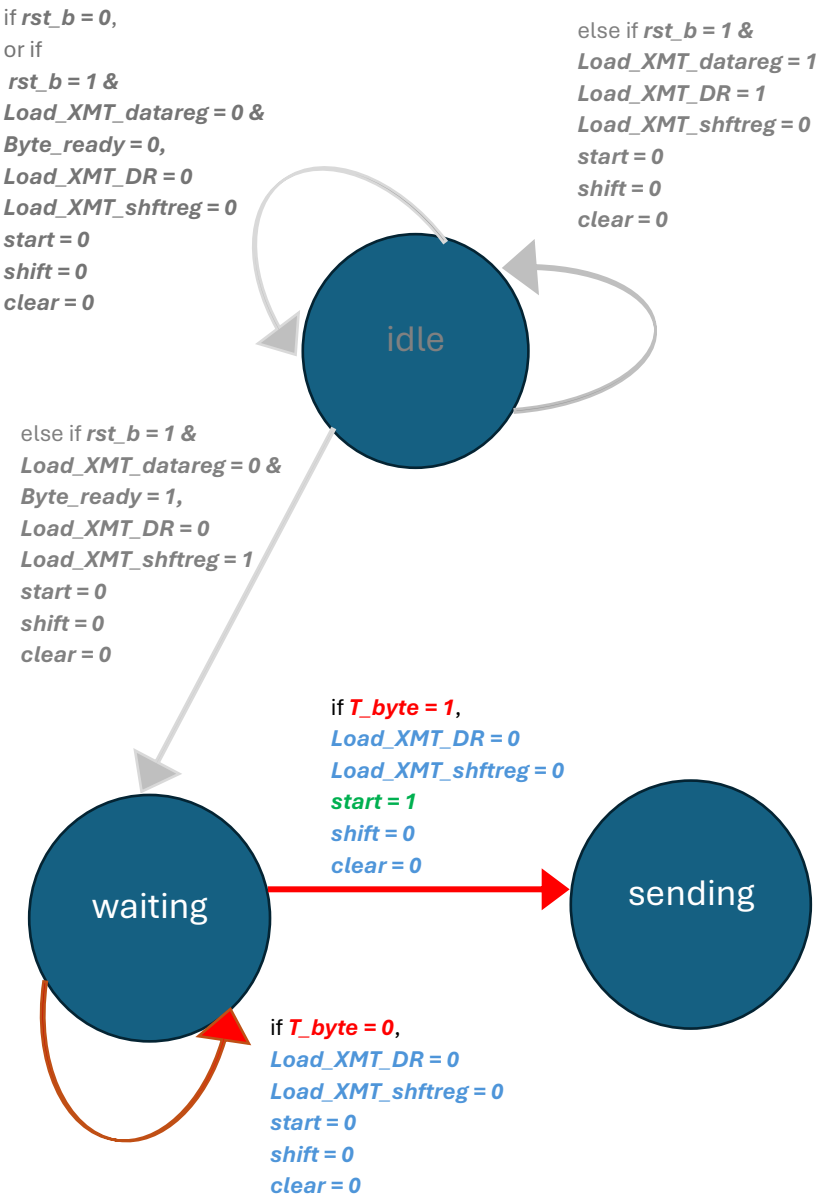
Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.



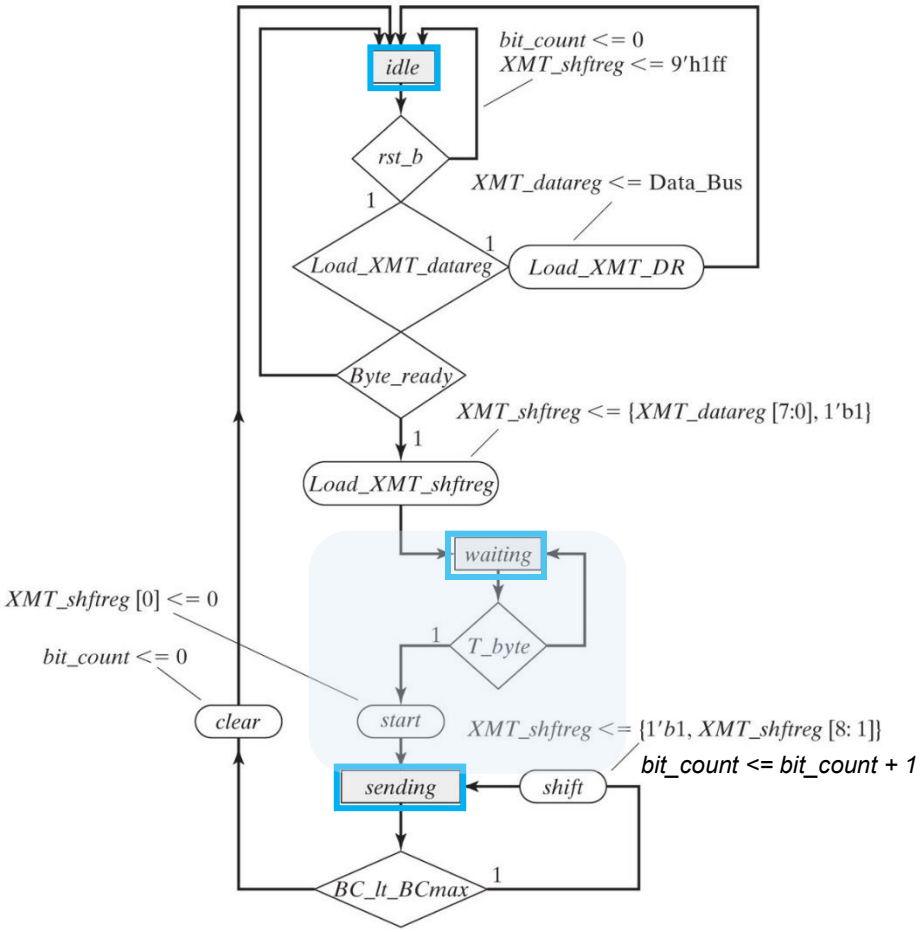
Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Generating the FSM



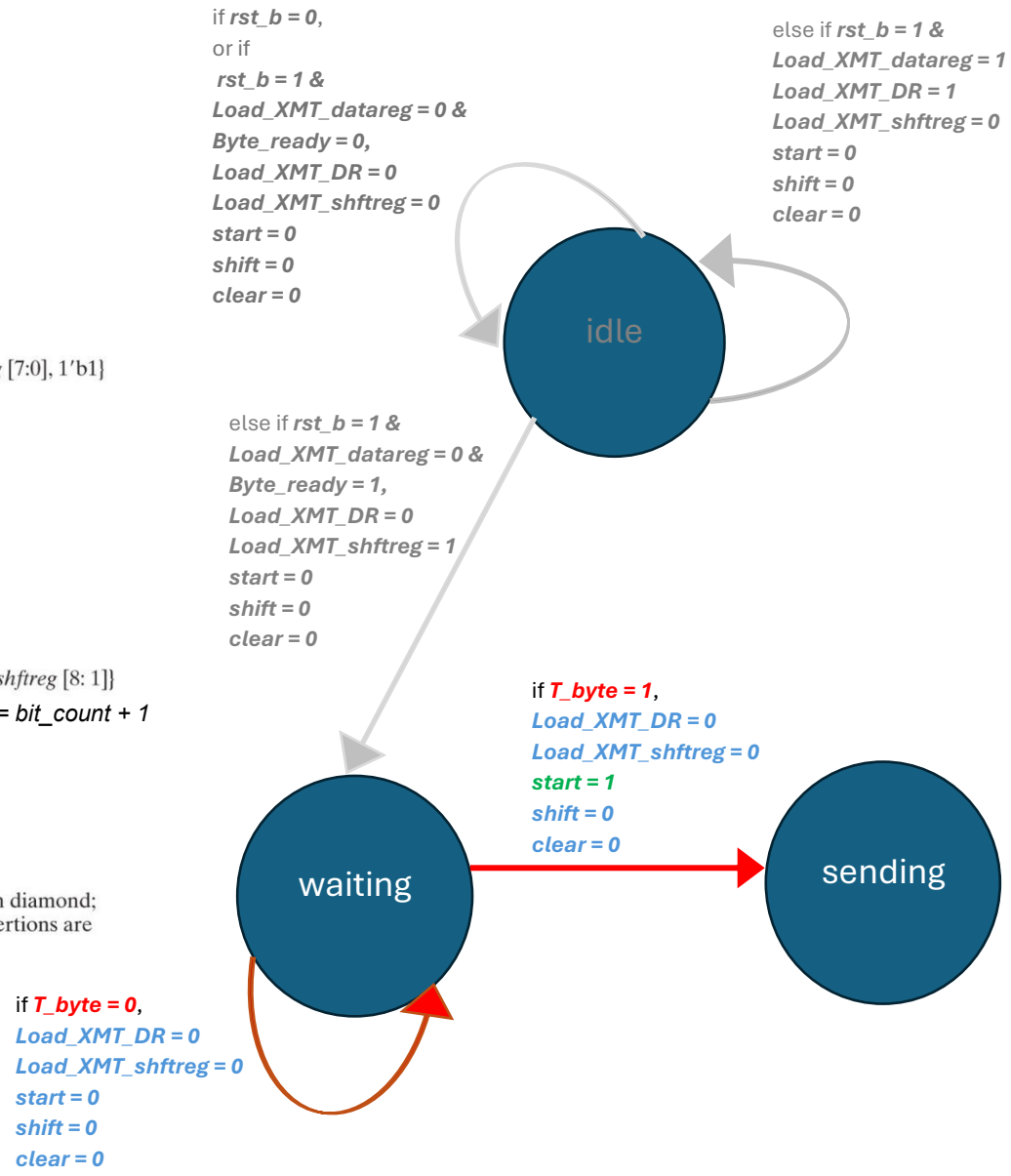
Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.



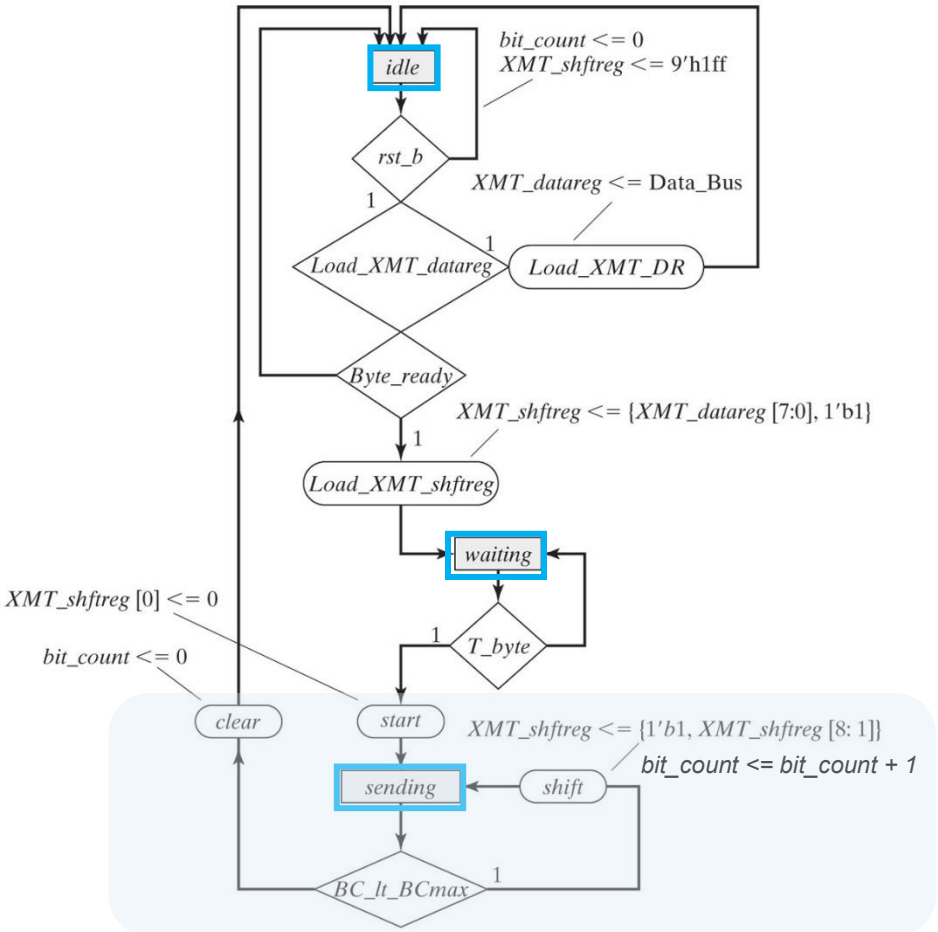
Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Generating the FSM



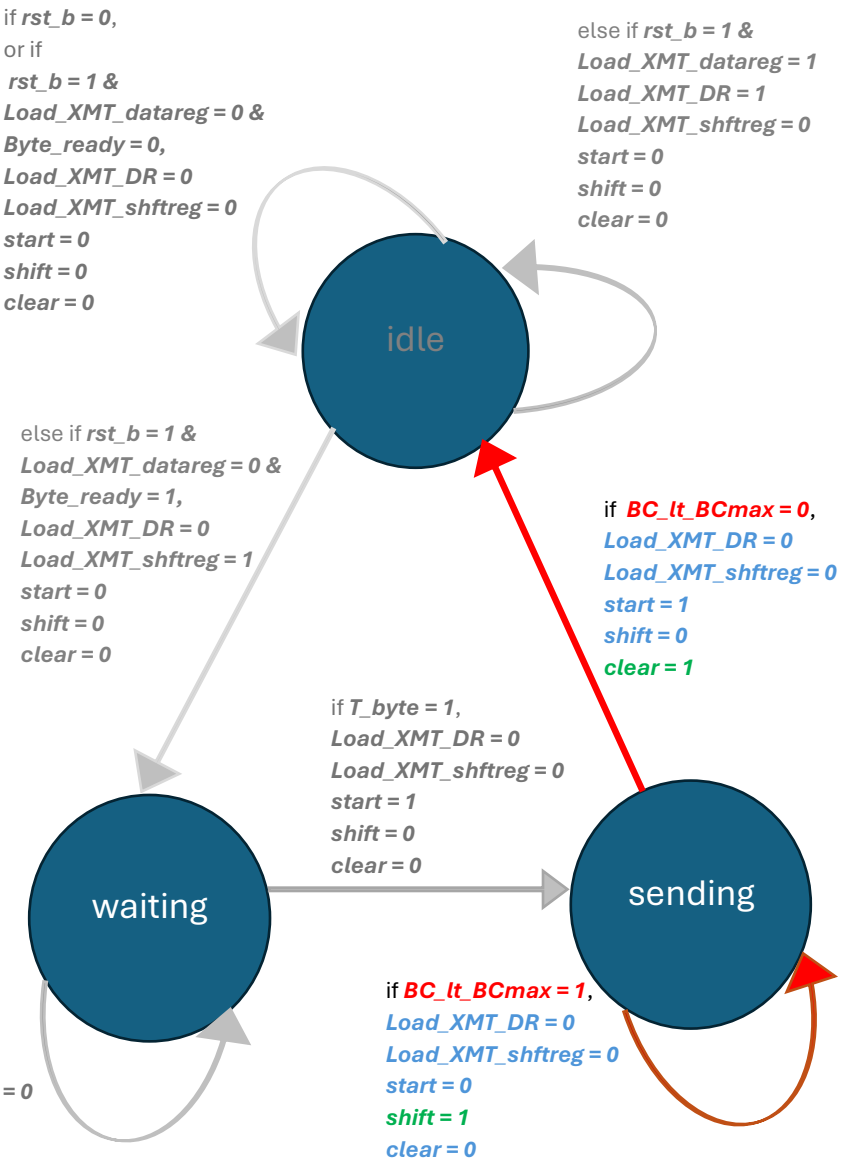
Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.



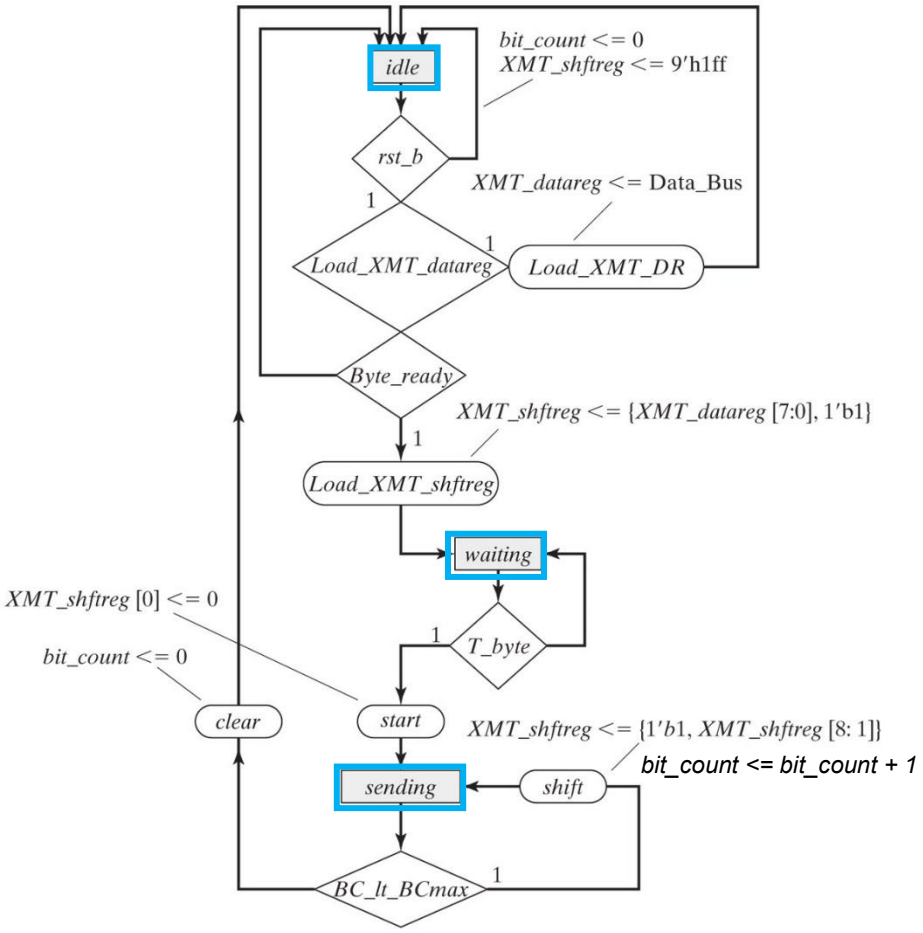
Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Generating the FSM



Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.



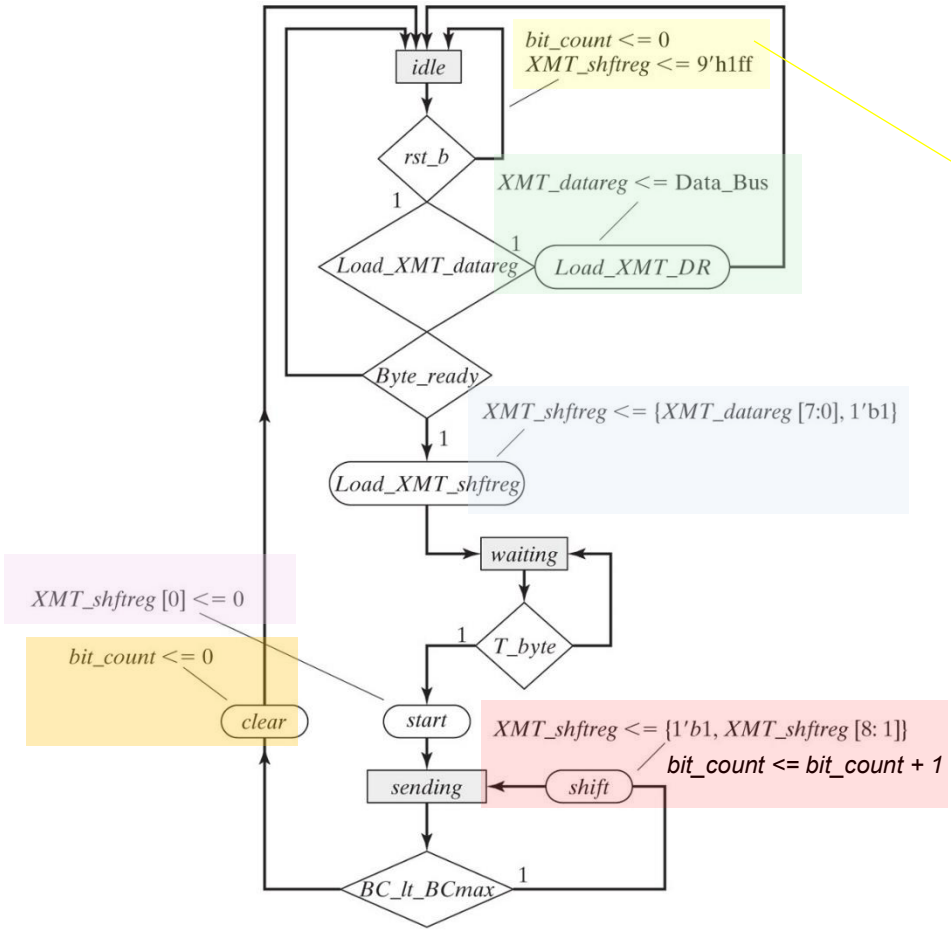
Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: Controller Verilog code



Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.


```
always @ (posedge clock, negedge rst_b)
    begin: State_transition
        if (rst_b == 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_XMT_DR = 0;            Load_XMT_shftreg  = 0;
            start     = 0;                  shift     = 0;
            clear     = 0;                next_state = idle;

         case (state)
            idle:     if (Load_Tx_datareg == 1'b1) begin
                                Load_XMT_DR = 1;
                                next_state = idle;      end
                      else if (Byte_ready == 1'b1) begin
                                Load_XMT_shftreg = 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
```

Explaining the ASMD (Algorithmic State Machine and DataPath) for the UART transmitter: DataPath Verilog code



Note: Only the branch corresponding to a true decision is annotated at a decision diamond; signals that are de-asserted are not shown explicitly de-asserted. Conditional assertions are indicated by the name of the asserted signal.
Note: *BC_lt_BCmax* asserts if *Bit_count* < *word_size* +1.


```
assign Serial_out      = XMT_shftreg[0];
assign BC_lt_BCmax = (bit_count < word_size+1);


always @ (posedge clock, negedge rst_b)

    if (rst_b == 0) begin
            XMT_shftreg <= 9'h1FF;
            bit_count     <= 0;
                end

        else  begin: Register_Transfers
            if (Load_XMT_DR == 1'b1)
               XMT_datareg <= Data_Bus;      // get the data word from data bus

            if (Load_Tx_SR == 1'b1)
               XMT_shftreg <= {XMT_datareg, 1'b1};   // load shftreg & insert start bit

            if (start == 1'b1)
               XMT_shftreg [0] <= 0;              // signal start of transmission

            if (clear == 1'b1)
                bit_count <= 0;

            if (shift == 1'b1) begin      // shift Shift_Register right and fill with 1's
               XMT_shftreg <= {1'b1, XMT_shftreg [word_size : 1]};
                bit_count      <= bit_count + 1;        end

            end
```