transmitter and receiver modules. The transmitter module will be basically a shift register that loads parallel data and shifts it in a specific rate through TX pin of the device. The receiver module will convert the received serial data through RX pin into parallel form to be processed by the receiver. Before dealing with HDL descriptions, let's first focus on the working principles of UART.

12.1.1 Working Principles of UART

To use a UART, we should understand how it works. Therefore, we introduce data format, timing, transmission, and reception operations in this section. These will help us forming HDL descriptions in the following section.

12.1.1.1 Data Format

Data is transmitted in terms of packages in the UART. Data framing of a UART package begins with a start bit, followed by seven to eight data bits optionally attached by a parity bit (explained in Sec. 8.6), and concluded by one or two stop bits. This setup can be seen in Fig. 12.1.

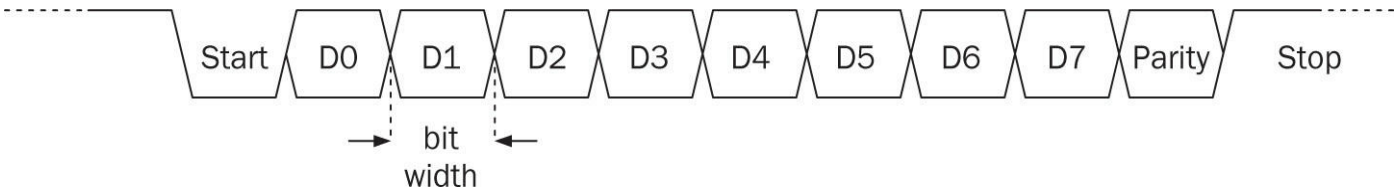

FIGURE 12.1 Data framing of a UART package with eight-bit data.

12.1.1.2 Timing

Although the UART works in asynchronous manner, the transmitter and receiver should have same timing values to transmit and receive data. In other words, the data can be transmitted in asynchronous manner. However, as the transmission starts, the receiver should know the duration of each pulse in the UART package. This is set by the baud rate which determines the timing. The baud rate is denoted by bits per second (bps). For example, a 2400-bps indicates a 416-µs bit width (or period) in the UART transmission.

12.1.1.3 Transmission Operation

We can explain the transmission operation in the UART as a state machine. We will explain this state machine in detail in Sec. 12.1.2. Here, let's briefly summarize it. The TX pin should be at logic level 1 when the transmitter is in idle mode. Once transmission starts, a falling edge is created on the data transmit line which wakes up the receiver. Afterward, the clock is set according to the baud rate and all bits are sent one by one in every clock cycle in the transmitter side. The receiver should have the same baud rate for receiving transmitted bits sequentially. As the transmit operation finalizes, the TX pin should be set to logic level 1 for one or two bit widths to inform

the receiver that the transmission is done. These are also called stop bit(s). The number of stop bits and usage of parity bit should also be predetermined so that the transmitter and receiver have same settings.

12.1.1.4 Reception Operation

We can explain the reception operation in the UART as another state machine. Although we will explain this state machine in Sec. 12.1.2, let's briefly summarize it here. The receiver will be in ready state initially. When a falling edge signal (start bit) comes to `RX` pin, it starts receiving data bits sequentially. To do so, the receiver should have an internal timer with the predetermined baud rate as in the transmitter. After receiving start bit, the timer waits for a certain time to sample the first data bit. This offset allows starting the sampling process in the middle of the first data pulse. Note that although data is sent as logic levels 1 and 0 by the transmitter, these are converted to analog pulse signals. Hence, the sampling operation converts the received analog signal to logic level 0 or 1 again. Afterward, we perform the sampling operation at each successive time period to recover data bits. As all bits are received this way, the receiver checks the parity bit within the received data (if the protocol consists one). When stop bit(s) is received, the receiver turns back to ready state waiting to receive the next data packet.

12.1.2 UART in Verilog

We can describe the transmit and receive operations as two separate modules in Verilog. Let's start with the transmitter module.

12.1.2.1 The Transmitter Module

The Verilog description of the transmitter module is presented in Listing 12.1. This module has three inputs as `send`, `data`, and `clk`. `send` is used to trigger starting a transmit operation. `data` carries data to be transmitted. `clk` is used to enter the 100-MHz clock of the FPGA board (Basys3 or Arty) to the module. The transmitter module has two outputs as `ready` and `tx`. When `ready` is at logic level 1, this indicates that the module is ready to transmit data. Output `tx` should be directly connected to `TX` pin of the device.

The working principles of the transmitter module (as a state machine) are as follows. Within the module, the baud rate is defined as a parameter and set to 9600 bps by default. Here, `baud_timer` calculates the number of clock cycles needed for a particular baud rate by dividing the main clock frequency to the baud rate. The transmitter module has three states as `RDY`, `LOAD_BIT`, and `SEND_BIT`. `RDY` state indicates that the module is ready to send next data package. When in the `LOAD_BIT` state, the data is loaded to `tx` output. Finally, `SEND_BIT` state indicates that the data is being transmitted. Initial state of the module is set as `RDY`. Hence, it waits for the `send` trigger. When `send` is set to logic level 1, the module loads `data` with a leading zero and a trailing one to `txData`. Afterward, the module switches to `LOAD_BIT` state. Here,

the first bit to be transmitted (LSB in our configuration) is loaded to txBit. Then, the module waits for bit_index_max clock cycles in SEND_BIT state. Then, it switches back to LOAD_BIT state to load the next bit to be transmitted. This operation is repeated until the last stop bit is transmitted. At the end of the transmission operation, the state is set as RDY. Hence, the transmitter module starts waiting for the next send trigger. In this module, txBit is wired to tx and ready is set as a conditional assignment such that when state equals to RDY, it is at logic level 1, otherwise 0.

12.1.2.2 The Receiver Module

The Verilog description of the receiver module is presented in Listing 12.2. The module has two inputs as clk and rx. clk is used to enter the 100-MHz clock of the FPGA board (Basys3 or Arty) to the module as in the transmitter module. rx should be directly connected to RX pin of the device. Through it, the receiver module listens for a possible incoming package. The receiver module has four outputs as data, parity, ready, and error. data represents the received data. parity shows the received parity bit. ready indicates that the receive operation is complete. Finally, error shows if the data package is received with or without error.

The working principles of the receiver module (as a state machine) are as follows. Within the module, the baud rate is defined as a parameter and set to 9600 bps similar to the transmitter module. As in the transmitter module, baud_timer calculates the number of clock cycles needed for a particular baud rate by dividing the main clock frequency to baud rate. The receiver module has five states as RDY, START, RECEIVE, WAIT, and CHECK. The state machine starts initially at RDY state, which indicates that the module is ready to receive the next data package. Hence, it listens to the RX pin through rx at every rising edge of the clock. When rx goes to logic level 0, the state machine goes to START. There, it waits for half of the baud_timer period where it ends up in the middle of the start signal. First data bit will be ready to be read after waiting for baud_timer period. WAIT state acts as a delay station in which the receiver waits for baud_timer period. Then, it returns to RECEIVE state unless ready is at logic level 1. In RECEIVE state, the incoming data is sampled. Then, bitIndex is incremented by one and checked whether it has reached the maximum value (eight for our case).

Listing 12.1 Verilog Description of the UART Transmitter Module

```
module UART_tx_ctrl(ready,uart_tx,send,data,clk);

input send, clk;
input [7:0] data;
output ready, uart_tx;

parameter baud = 9600;
parameter bit_index_max = 10;

localparam [31:0] baud_timer = 100000000/baud;

localparam RDY = 2'b00, LOAD_BIT = 2'b01, SEND_BIT = 2'b10;

reg [1:0] state = RDY;
reg [31:0] timer = 0;
reg [9:0] txData;
reg [3:0] bitIndex;
reg txBit=1'b1;

always @ (posedge clk)

case (state)
RDY:
        begin
        if (send)
        begin
        txData <= {1'b1,data,1'b0};
        state <= LOAD_BIT;
        end
        timer <= 14'b0;
        bitIndex <= 0;
        txBit <= 1'b1;
        end
LOAD_BIT:
        begin
        state <= SEND_BIT;
        bitIndex <= bitIndex + 1'b1;
        txBit <= txData[bitIndex];
        end
SEND_BIT:
        if (timer == baud_timer)
        begin
        timer <= 14'b0;
        if (bitIndex == bit_index_max)
        state <= RDY;
        else state <= LOAD_BIT;
        end
        else timer <= timer + 1'b1;
default:
        state <= RDY;
endcase

assign uart_tx = txBit;
assign ready = (state == RDY);
endmodule
```

Listing 12.2 Verilog Description of the UART Receiver Module

```
module UART_rx_ctrl(clk,rx,data,parity,ready,error);

input clk, rx;
output reg [7:0] data;
output reg parity;
output reg ready=0;
output reg error=0;

parameter baud=9600;

localparam RDY=3'b000, START=3'b001, RECEIVE=3'b010, WAIT=3'b011,
    CHECK=3'b100;

reg [2:0] state = RDY;

localparam [31:0] baud_timer = 100000000/baud;

reg [31:0] timer = 32'b0;
reg [3:0] bitIndex = 3'b0;
reg [8:0] rxdata;

always @ (posedge clk)

case (state)
RDY:
        if (rx == 1'b0)
        begin
        state <= START;
        bitIndex <= 3'b0;
        end
START:
        if (timer == baud_timer/2)
        begin
        state <= WAIT;
        timer <= 14'b0;
        error <= 1'b0;
        ready <= 1'b0;
        end
        else timer <= timer + 1'b1;
WAIT:
        if (timer == baud_timer)
        begin
        timer <= 14'b0;
        if (ready) state <= RDY;
        else state <= RECEIVE;
        end
        else timer <= timer + 1'b1;
RECEIVE:
        begin
        rxdata[bitIndex] <= rx;
        bitIndex <= bitIndex + 1'b1;
        if (bitIndex == 4'd8) state <= CHECK;
        else state <= WAIT;
        end
```

```
CHECK:
        if (^rxdata[7:0] == rxdata[8])
        begin
        ready <= 1'b1;
        state <= WAIT;
        data <= rxdata[7:0];
        parity <= rxdata[8];
        end
        else
        begin
        ready <= 1'b1;
        data[7:0] <= 8'bx;
        error <= 1'b1;
        state <= RDY;
        end
endcase
endmodule
```

Since we have eight data bits and a parity bit, the state machine has to switch to `CHECK` state after all bits are received. Even parity check is performed in `CHECK` state. If the received data package is consistent with parity bit, then `ready` is set to logic level 1 and the next state is set to `WAIT`. Received data and parity values in `rxdata` are written to `data` and `parity` outputs. If the parity check fails, then `error` and `ready` go to logic level 1. `data` is filled with logic level 1. Then, the reception operation ends. The receiver turns back to `RDY` state waiting to receive the next data package.

12.1.3 UART in VHDL

As in Verilog, we can describe the transmit and receive operations in two separate modules in VHDL. Let's start with the transmitter module.

12.1.3.1 The Transmitter Module

The VHDL description of the transmitter module is presented in Listing 12.3. In this description, we tried to keep the input, output definitions, and state names the same as in Listing 12.1. Hence, the reader can associate the working principles of the corresponding Verilog description with the VHDL description here.

12.1.3.2 The Receiver Module

We provide the VHDL description of the receiver module in Listing 12.4. Again, this module has the same working principles as its Verilog version in Listing 12.1.

12.1.4 UART Applications

The UART needs an RS-232 port for communication. Unfortunately, Basys3 and Arty boards do not have such a port. However, they share the micro USB port for the UART communication as mentioned in Chap. 3. To run UART applications in this