

University of Moratuwa
Department of Electronic & Telecommunication



EN2111 - Electronic Circuit Design
UART implementation in FPGA
Group no - 18

MADHUSHAN R.M.S.	200363R
MADUSAN A.K.C.S.	200366E
MALANBAN K.	200373X

Codes

Transmitter

```
1 module receiver #(
2     parameter CLOCKS_PER_PULSE = 16
3 )
4 (
5     input logic clk,
6     input logic rstn,
7     input logic ready_clr,
8     input logic rx,
9     output logic ready,
10    output logic [7:0] data_out
11 );
12
13    enum {RX_IDLE, RX_START, RX_DATA, RX_END} state;
14
15    logic[2:0] c_bits;
16    logic[$clog2(CLOCKS_PER_PULSE)-1:0] c_clocks;
17
18    logic[7:0] temp_data;|
19    logic rx_sync;
20
21    always_ff @(posedge clk or negedge rstn) begin
22
23        if (!rstn) begin
24            c_clocks <= 0;
25            c_bits <= 0;
26            temp_data <= 8'b0;
27            //data_out <= 8'b0;
28            ready <= 0;
29            state <= RX_IDLE;
30
31        end else begin
32            rx_sync <= rx; // Synchronize the input signal using a
33            flip-flop
34
35            case (state)
36            RX_IDLE : begin
37                if (rx_sync == 0) begin
38                    state <= RX_START;
39                    c_clocks <= 0;
40                end
41            end
42            RX_START: begin
43                if (c_clocks == CLOCKS_PER_PULSE/2-1) begin
44                    state <= RX_DATA;
45                    c_clocks <= 0;
46                end else
47                    c_clocks <= c_clocks + 1;
48            end
49            RX_DATA : begin
50                if (c_clocks == CLOCKS_PER_PULSE-1) begin
51                    c_clocks <= 0;
52                    temp_data[c_bits] <= rx_sync;
53                    if (c_bits == 3'd7) begin
54                        state <= RX_END;
55                        c_bits <= 0;
56                    end else c_bits <= c_bits + 1;
57                end else c_clocks <= c_clocks + 1;
58            end
59            RX_END : begin
60                if (c_clocks == CLOCKS_PER_PULSE-1) begin
61                    //data_out <= temp_data;
62                    ready <= 1'b1;
63                    state <= RX_IDLE;
64                    c_clocks <= 0;
65                end else c_clocks <= c_clocks + 1;
66            end
67            default: state <= RX_IDLE;
68            endcase
69        end
70    end
71    assign data_out = temp_data;
72 endmodule
73
```

Receiver

```
1 module transmitter #(
2     parameter CLOCKS_PER_PULSE = 16
3 )
4 (
5     input logic [7:0] data_in,
6     input logic data_en,
7     input logic clk,
8     input logic rstn,
9     output logic tx,
10    output logic tx_busy
11 );
12
13    enum {TX_IDLE, TX_START, TX_DATA, TX_END} state;
14
15    logic[7:0] data = 8'b0;
16    logic[2:0] c_bits = 3'b0;
17    logic[$clog2(CLOCKS_PER_PULSE)-1:0] c_clocks = 0;
18
19    always_ff @(posedge clk or negedge rstn) begin
20        if (!rstn) begin
21            c_clocks <= 0;
22            c_bits <= 0;
23            data <= 0;
24            tx <= 1'b1;
25            state <= TX_IDLE;
26        end else begin
27            case (state)
28            TX_IDLE: begin
29                if (~data_en) begin
30                    state <= TX_START;
31                    data <= data_in;
32                    c_bits <= 3'b0;
33                    c_clocks <= 0;
34                end else tx <= 1'b1;
35            end
36            TX_START: begin
37                if (c_clocks == CLOCKS_PER_PULSE-1) begin
38                    state <= TX_DATA;
39                    c_clocks <= 0;
40                end else begin
41                    tx <= 1'b0;
42                    c_clocks <= c_clocks + 1;
43                end
44            end
45            TX_DATA: begin
46                if (c_clocks == CLOCKS_PER_PULSE-1) begin
47                    c_clocks <= 0;
48                    if (c_bits == 3'd7) begin
49                        state <= TX_END;
50                    end else begin
51                        c_bits <= c_bits + 1;
52                        tx <= data[c_bits];
53                    end
54                end else begin
55                    tx <= data[c_bits];
56                    c_clocks <= c_clocks + 1;
57                end
58            end
59            TX_END: begin
60                if (c_clocks == CLOCKS_PER_PULSE-1) begin
61                    state <= TX_IDLE;
62                    c_clocks <= 0;
63                end else begin
64                    tx <= 1'b1;
65                    c_clocks <= c_clocks + 1;
66                end
67            end
68            default: state <= TX_IDLE;
69            endcase
70        end
71    end
72    assign tx_busy = (state != TX_IDLE);
73 endmodule
74
```

Binary to 7-segment converter

```

1 module binary_to_7seg (
2     input logic [3:0] data_in,
3     output logic [6:0] data_out
4 );
5 // Make a LUT to convert digits to 7 segment output
6 // Input - 4 bits, output - 7 bits
7 logic [15:0][6:0] lut_7seg;
8
9 // Output is gfedcba
10 assign lut_7seg[0] = 7'b0111111;
11 assign lut_7seg[1] = 7'b0000110;
12 assign lut_7seg[2] = 7'b1011011;
13 assign lut_7seg[3] = 7'b1001111;
14 assign lut_7seg[4] = 7'b1100110;
15 assign lut_7seg[5] = 7'b1101101;
16 assign lut_7seg[6] = 7'b1111101;
17 assign lut_7seg[7] = 7'b0000111;
18 assign lut_7seg[8] = 7'b1111111;
19 assign lut_7seg[9] = 7'b1101111;
20 assign lut_7seg[15:10] = 7'b0; // unused
21
22 assign data_out = ~lut_7seg[data_in];
23
24 endmodule
25
26 /*
27 Seven segment display
28 a
29 f b
30 g
31 e c
32 d
33 */
34 */

```

UART

```

1 module uart #(
2     parameter CLOCKS_PER_PULSE = 5208
3 )
4 (
5     input logic [3:0] data_in,
6     input logic data_en,
7     input logic clk,
8     input logic rstn,
9     output logic tx,
10    output logic tx_busy,
11    input logic ready_clr,
12    input logic rx,
13    output logic ready,
14    output logic [3:0] led_out,
15    output logic [6:0] display_out
16 );
17 logic [7:0] data_input;
18 logic [7:0] data_output;
19
20 transmitter #(.CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) uart_tx (
21     .data_in(data_input),
22     .data_en(data_en),
23     .clk(clk),
24     .rstn(rstn),
25     .tx(tx),
26     .tx_busy(tx_busy)
27 );
28
29 receiver #(.CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) uart_rx (
30     .clk(clk),
31     .rstn(rstn),
32     .ready_clr(ready_clr),
33     .rx(rx),
34     .ready(ready),
35     .data_out(data_output)
36 );
37
38 binary_to_7seg converter (
39     .data_in(data_output[3:0]),
40     .data_out(display_out)
41 );
42
43 assign data_input = {4'b0, data_in};
44 assign led_out = data_output[3:0];
45
46 endmodule
47

```

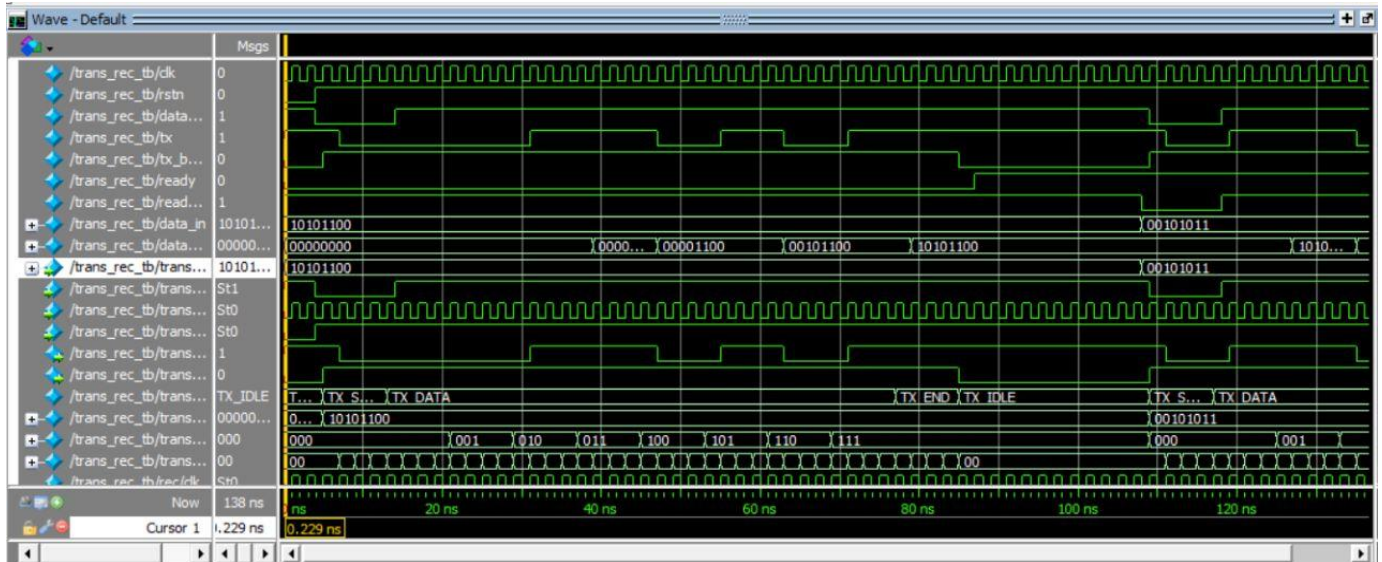
Test bench

```

1 `timescale 1ns/1ps
2
3 module testbench();
4
5     localparam CLOCKS_PER_PULSE = 4;
6     logic [3:0] data_in = 4'b0001;
7     logic clk = 0;
8     logic rstn = 0;
9     logic enable = 1;
10
11     logic tx_busy;
12     logic ready;
13     logic [3:0] data_out;
14     logic [7:0] display_out;
15
16     logic loopback;
17     logic ready_clr = 1;
18
19     uart #(.CLOCKS_PER_PULSE(CLOCKS_PER_PULSE))
20         test_uart(.data_in(data_in),
21                 .data_en(enable),
22                 .clk(clk),
23                 .tx(loopback),
24                 .tx_busy(tx_busy),
25                 .rx(loopback),
26                 .ready(ready),
27                 .ready_clr(ready_clr),
28                 .led_out(data_out),
29                 .display_out(display_out),
30                 .rstn(rstn)
31             );
32
33
34     always begin
35         #1 clk = ~clk;
36     end
37
38     initial begin
39         $dumpfile("testbench.vcd");
40         $dumpvars(0, testbench);
41         rstn <= 1;
42         enable <= 1'b0;
43         #2 rstn <= 0;
44         #2 rstn <= 1;
45         #5 enable <= 1'b1;
46     end
47
48     always @(posedge ready) begin
49
50         if (data_out != data_in) begin
51             $display("FAIL: rx data %x does not match tx %x",
52                 data_out, data_in);
53             $finish();
54         end else begin
55             if (data_out == 4'b1111) begin //Check if received d
56                 $display("SUCCESS: all bytes verified");
57                 $finish();
58             end
59
60             #10 rstn <= 0;
61             #2 rstn <= 1;
62             data_in <= data_in + 1'b1;
63             enable <= 1'b0;
64             #2 enable <= 1'b1;
65         end
66     end
67 endmodule

```

Timing diagram captured on simulation



FPGA with oscilloscope showing the waveform

