Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Lab Assignment
Project on UART Implementation

Group 2

Group Members:

210169L	Fernando W.W.R.N.S.
210179R	Gammune D.J.T.
210218M	Herath H.M.S.I.

Submitted in partial fulfillment of the requirements for the module
EN2111 - Electronic Circuit Design

2024/05/07

Contents

Abstract

This report details the design and implementation of a UART transceiver on an FPGA platform, encompassing four distinct phases. The first phase involves researching the UART protocol, exploring existing Verilog implementations, and selecting a suitable design that meets the specified requirements. The second phase concentrates on developing a comprehensive testbench in Verilog to verify the functionality of the UART module under various test scenarios, including edge cases and potential error conditions. The third phase focuses on the FPGA implementation of the UART module, integrating it with the necessary logic for data transmission and reception. This phase also covers the simulation, synthesis, and programming of the FPGA board with the final design. Finally, the fourth phase involves testing the UART module's functionality without the use of 7-segment displays or oscilloscope analysis. Emphasis is placed on understanding the theoretical concepts behind the UART protocol, implementing a robust and efficient design, and verifying its functionality through rigorous testing and analysis. Throughout the project, emphasis is placed on understanding the theoretical concepts behind the UART protocol, implementing a robust and efficient design, and verifying its functionality through rigorous testing and analysis. The report provides detailed insights into the design process, implementation strategies, and the challenges encountered during each phase of the project.

1 Introduction

The Universal Asynchronous Receiver/Transmitter (UART) protocol is vital for serial data transmission between devices. This project focuses on implementing a UART transceiver on a Field Programmable Gate Array (FPGA) platform. By leveraging FPGA's flexibility, developers can tailor UART functionality to specific needs, ensuring seamless integration into diverse systems. FPGA's parallel processing capabilities enhance performance and efficiency, while its development ecosystem enables rapid prototyping and innovation. In essence, FPGA-based UART solutions combine cutting-edge communication principles with reconfigurable hardware design, driving technological progress across industries.

2 Block Diagram and Maps

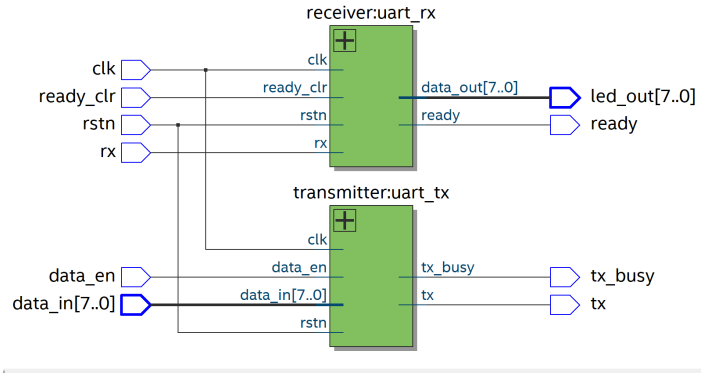

Figure 1: Block Diagram

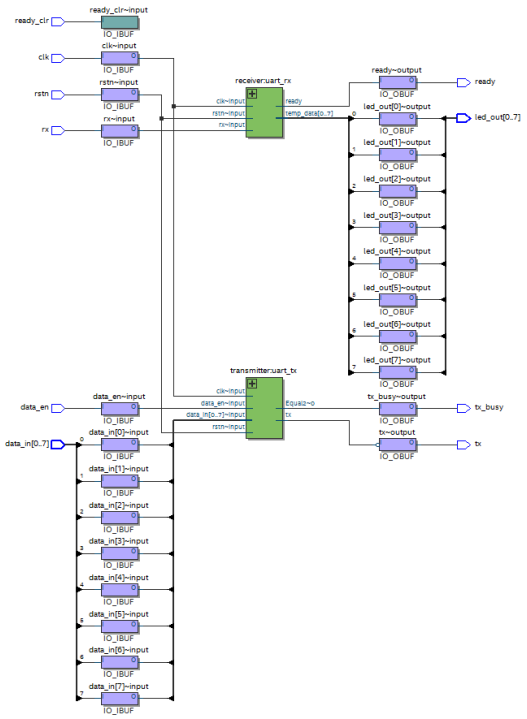

Figure 2: Technology Map

3 UART Codes

3.1 RTL Code for UART

```
module uart #(
        parameter CLOCKS_PER_PULSE = 5208
)
(
        input logic [7:0] data_in,
        input logic data_en,
        input logic clk,
        input logic rstn,
        output logic tx,
        output logic tx_busy,
        input logic ready_clr,
        input logic rx,
        output logic ready,
        output logic [7:0] led_out
);
        logic [7:0] data_input;
        logic [7:0] data_output;

        transmitter #(.CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) uart_tx (
                .data_in(data_input),
                .data_en(data_en),
                .clk(clk),
                .rstn(rstn),
                .tx(tx),
                .tx_busy(tx_busy)
        );

        receiver #(.CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) uart_rx (
                .clk(clk),
                .rstn(rstn),
                .ready_clr(ready_clr),
                .rx(rx),
                .ready(ready),
                .data_out(data_output)
        );

        assign data_input = {8'b0, data_in};
        assign led_out = data_output[7:0];


endmodule
```

3.2 Transmitter

```
module transmitter #(
        parameter CLOCKS_PER_PULSE = 16
)
(
        input logic [7:0] data_in,
        input logic data_en,
        input logic clk,
        input logic rstn,
        output logic tx,
        output logic tx_busy
);
        enum {TX_IDLE, TX_START, TX_DATA, TX_END} state;

        logic[7:0] data = 8'b0;
```

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

3.3 Receiver

```
module receiver #(
        parameter CLOCKS_PER_PULSE = 16
)
(
        input logic clk,
        input logic rstn,
        input logic ready_clr,
        input logic rx,
        output logic ready,
        output logic [7:0] data_out
);

        enum {RX_IDLE, RX_START, RX_DATA, RX_END} state;

        logic[2:0] c_bits;
        logic[$clog2(CLOCKS_PER_PULSE)-1:0] c_clocks;

        logic[7:0] temp_data;
        logic rx_sync;

        always_ff @(posedge clk or negedge rstn) begin

                if (!rstn) begin
                        c_clocks <= 0;
                        c_bits <= 0;
                        temp_data <= 8'b0;
                        //data_out <= 8'b0;
                        ready <= 0;
                        state <= RX_IDLE;

                end else begin
                        rx_sync <= rx;  // Synchronize the input signal
                           using a flip-flop

                        case (state)
                        RX_IDLE : begin
                                if (rx_sync == 0) begin
                                        state <= RX_START;
                                        c_clocks <= 0;
                                end
                        end
                        RX_START: begin
                                if (c_clocks == CLOCKS_PER_PULSE/2-1) begin
                                        state <= RX_DATA;
                                        c_clocks <= 0;
                                end else
                                        c_clocks <= c_clocks + 1;
                        end
                        RX_DATA : begin
                                if (c_clocks == CLOCKS_PER_PULSE-1) begin
                                        c_clocks <= 0;
                                        temp_data[c_bits] <= rx_sync;
                                        if (c_bits == 3'd7) begin
                                                state <= RX_END;
                                                c_bits <= 0;
                                        end else c_bits <= c_bits + 1;
                                end else c_clocks <= c_clocks + 1;
                        end
                        RX_END : begin
                                if (c_clocks == CLOCKS_PER_PULSE-1) begin
                                        //data_out <= temp_data;
```

```
                                        ready <= 1'b1;
                                        state <= RX_IDLE;
                                        c_clocks <= 0;
                                end else c_clocks <= c_clocks + 1;
                        end
                        default: state <= RX_IDLE;
                        endcase
                end
        end
        assign data_out = temp_data;
endmodule
```

3.4 Test Bench

```
`timescale 1ns/1ps

module testbench();

        localparam CLOCKS_PER_PULSE = 4;
  logic [7:0] data_in = 8'b00000001;
        logic clk = 0;
        logic rstn = 0;
        logic enable = 1;

        logic tx_busy;
        logic ready;
        logic [7:0] data_out;

        logic loopback;
        logic ready_clr = 1;

        uart #(.CLOCKS_PER_PULSE(CLOCKS_PER_PULSE))
                        test_uart(.data_in(data_in),
                                                .data_en(enable),
                                                .clk(clk),
                                                .tx(loopback),
                                                .tx_busy(tx_busy),
                                                .rx(loopback),
                                                .ready(ready),
                                                .ready_clr(ready_clr),
                                                .led_out(data_out),
                                                .rstn(rstn)
                                                );


        always begin
                #1 clk = ~clk;
        end

        initial begin
                $dumpfile("testbench.vcd");
                $dumpvars(0, testbench);
        rstn <= 1;
                enable <= 1'b0;
        #2 rstn <= 0;
                #2 rstn <= 1;
                #5 enable <= 1'b1;
        end


        always @(posedge ready) begin

```

```
                if (data_out != data_in) begin
                        $display("FAIL: rx data %x does not match tx %x",
                           data_out, data_in);
         $finish();
                end else begin
          if (data_out == 8'b11111111) begin //Check if received data is
             11111111
                                $display("SUCCESS: all bytes verified");
                                $finish();
                        end

                        #10 rstn <= 0;
                        #2 rstn <= 1;
                        data_in <= data_in + 1'b1;
                        enable <= 1'b0;
                        #2 enable <= 1'b1;
                end
        end
endmodule
```

3.4.1 Transmitter Test bench

```
`timescale 1ns/1ps

module transmitter_tb;

        localparam CLOCKS_PER_PULSE = 16, //200_000_000/9600

        CLK_PERIOD = 10;
        logic clk=0, rstn=0, data_en=1, tx, tx_busy;

        reg[7:0] data_in = 8'b10101100;

        always #1 clk <= ~clk;

        transmitter #(
        .CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) dut (.*);

        initial begin
                $dumpfile("dump.vcd"); $dumpvars;
                repeat(2) @(posedge clk) #1;
                rstn = 1;
                data_en = 0;
                repeat(5) @(posedge clk) #1;
                wait(!tx_busy);
                $finish();
        end
endmodule
```

3.4.2 Receiver Test bench

```
`timescale 1ns/1ps

module trans_rec_tb;

        localparam CLOCKS_PER_PULSE = 4; //200_000_000/9600

        localparam CLK_PERIOD = 10;
        logic clk=0, rstn=0, data_en=1, tx, tx_busy;
        logic ready, ready_clr=1;

```

```
        logic[7:0] data_in = 8'b10101100;
        logic[7:0] data_out;

        always #1 clk <= ~clk;

        transmitter #(
      .CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) trans (.*);

    receiver #(
      .CLOCKS_PER_PULSE(CLOCKS_PER_PULSE)) rec (
      .clk(clk),
      .rstn(rstn),
      .ready_clr(ready_clr),
      .rx(tx),
      .ready(ready),
      .data_out(data_out)
    );

        initial begin
                $dumpfile("dump.vcd"); $dumpvars;
                repeat(2) @(posedge clk) #1;
                rstn <= 1;
                data_en <= 0;
                repeat(5) @(posedge clk) #1;
        data_en <= 1;
        wait(ready);
        repeat(10) @(posedge clk) #1;
        data_in <= 8'b00101011;
        ready_clr <= 0;
        #1 data_en <= 0;
        repeat(5) @(posedge clk) #1;
        data_en <= 1;
        ready_clr <= 1;
        wait(ready);
        repeat(10) @(posedge clk) #1;
                $finish();
        end
endmodule
```

4 Model Sim Simulation and Timing Diagram

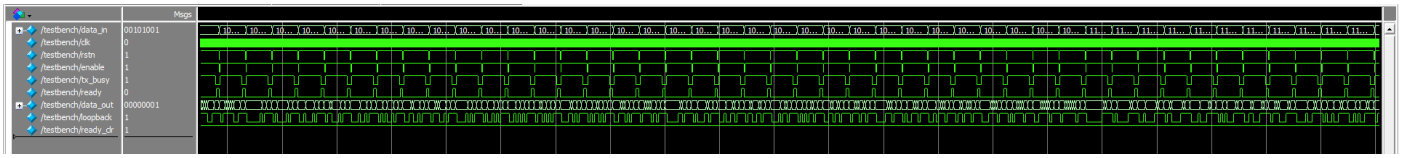

Figure 3: Timing Diagram

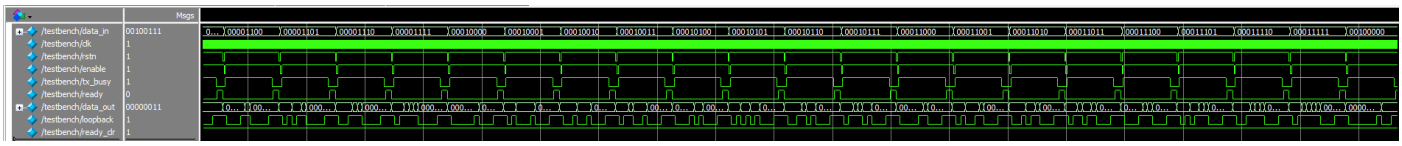

Figure 4: Zoomed Timing Diagram

5 Pin Planner

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
clk	Input	PIN_R8	3	B3_N0	PIN_R8	2.5 V		8mA (default)			
data_en	Input	PIN_E1	1	B1_N0	PIN_E1	2.5 V		8mA (default)			
data_in[7]	Input	PIN_B4	8	B8_N0	PIN_B4	2.5 V		8mA (default)			
data_in[6]	Input	PIN_B3	8	B8_N0	PIN_B3	2.5 V		8mA (default)			
data_in[5]	Input	PIN_A3	8	B8_N0	PIN_A3	2.5 V		8mA (default)			
data_in[4]	Input	PIN_A2	8	B8_N0	PIN_A2	2.5 V		8mA (default)			
data_in[3]	Input	PIN_M15	5	B5_N0	PIN_M15	2.5 V		8mA (default)			
data_in[2]	Input	PIN_B9	7	B7_N0	PIN_B9	2.5 V		8mA (default)			
data_in[1]	Input	PIN_T8	3	B3_N0	PIN_T8	2.5 V		8mA (default)			
data_in[0]	Input	PIN_M1	2	B2_N0	PIN_M1	2.5 V		8mA (default)			
led_out[7]	Output	PIN_L3	2	B2_N0	PIN_L3	2.5 V		8mA (default)	2 (default)		
led_out[6]	Output	PIN_B1	1	B1_N0	PIN_B1	2.5 V		8mA (default)	2 (default)		
led_out[5]	Output	PIN_F3	1	B1_N0	PIN_F3	2.5 V		8mA (default)	2 (default)		
led_out[4]	Output	PIN_D1	1	B1_N0	PIN_D1	2.5 V		8mA (default)	2 (default)		
led_out[3]	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		8mA (default)	2 (default)		
led_out[2]	Output	PIN_B13	7	B7_N0	PIN_B13	2.5 V		8mA (default)	2 (default)		
led_out[1]	Output	PIN_A13	7	B7_N0	PIN_A13	2.5 V		8mA (default)	2 (default)		
led_out[0]	Output	PIN_A15	7	B7_N0	PIN_A15	2.5 V		8mA (default)	2 (default)		
ready	Output	PIN_A4	8	B8_N0	PIN_A4	2.5 V		8mA (default)	2 (default)		
ready_clr	Input	PIN_A5	8	B8_N0	PIN_A5	2.5 V		8mA (default)			
rstn	Input	PIN_J15	5	B5_N0	PIN_J15	2.5 V		8mA (default)			
rx	Input	PIN_D16	6	B6_N0	PIN_D16	2.5 V		8mA (default)			
tx	Output	PIN_D15	6	B6_N0	PIN_D15	2.5 V		8mA (default)	2 (default)		
tx_busy	Output	PIN_B5	8	B8_N0	PIN_B5	2.5 V		8mA (default)	2 (default)		
<<new node>>											

Figure 5: Pin Planner

References

[1] Implementation of UART Protocol Using FPGA
https://medium.com/@namiwije/implementation-of-uart-protocol-using-fpga-a04d0b3b5bcb

[2] FPGA Basics
https://www.intel.com/content/www/us/en/support/programmable/support-resources/fpga-training/getting-started.html

[3] UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/-Transmitter
https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[4] UART-Implementation-Using-FPGA
https://github.com/namiwijeuom/UART-Implementation-Using-FPGA/tree/main

[5] DE0 Nano User Manual
https://www.terasic.com.tw/attachment/archive/941/DE0-Nano-SoC_User_manual_rev.D0.pdf