Department of Electronic and
Telecommunication Engineering
University of Moratuwa Faculty of Engineering

EN2111- Electronic Circuit Design

UART Assignment

Group members:
210490L – PRABODHA KPKA
210469G – PERERA PDP
210463H – PERERA LCS
Group no : 29

Introduction

The UART protocol is essential for serial data communication between devices, offering a direct method without requiring a centralized clock for synchronization like synchronous protocols do. Instead, UART uses a start bit and stop bit to delineate the beginning and end of each data transmission. In this setup, a transmitting device (transmitter) converts parallel data into a serial format before sending it to the receiving device (receiver), which then converts the serial data back into parallel form upon reception. This two-way data flow is managed through Tx (transmit) and Rx (receive) pins on the devices. One crucial parameter in UART communication is the baud rate, measured in bits per second (bps), which dictates the speed of data transfer. To ensure accurate data exchange, both the transmitter and receiver must operate at the same baud rate.

This project aims to implement a UART transceiver using Verilog, a hardware description language, and simulate it using Quartus Software. The next step involves practical testing on a DEO Nano FPGA. Accompanying the Verilog code will be a comprehensive test bench for simulation purposes, enabling thorough verification of the transceiver's functionality. By developing and validating this UART transceiver, valuable insights will be gained into its performance and suitability for various applications requiring serial data communication. Additionally, this project serves as a hands-on demonstration of hardware description language programming and FPGA-based prototyping, showcasing their role in communication system design.

RTL Code for UART

Transmitter

```
module transmitter(
    input wire [7:0] data_in,
    input wire wr_en,
    input wire clk_50m,
    input wire clken,
    output reg Tx,
    output wire Tx_busy
);

    parameter TX_STATE_IDLE = 2'b00;
    parameter TX_STATE_START = 2'b01;
    parameter TX_STATE_DATA = 2'b10;
    parameter TX_STATE_STOP = 2'b11;

    reg [7:0] data;
    reg [2:0] bit_pos;
    reg [1:0] state;

    always @(posedge clk_50m) begin
        case (state)
            TX_STATE_IDLE: begin
                if (~wr_en) begin
                    state <= TX_STATE_START;
                    data <= data_in;
                    bit_pos <= 3'h0;
                end
            end
            TX_STATE_START: begin
                if (clken) begin
                    Tx <= 1'b0;
                    state <= TX_STATE_DATA;
                end
            end
            TX_STATE_DATA: begin
                if (clken) begin
                    if (bit_pos == 3'h7)
                        state <= TX_STATE_STOP;
                    else
                        bit_pos <= bit_pos + 3'h1;
                    Tx <= data[bit_pos];
                end
            end
            TX_STATE_STOP: begin
                if (clken) begin
                    Tx <= 1'b1;
                    state <= TX_STATE_IDLE;
                end
            end
```

```
            default: begin
                Tx <= 1'b1;
                state <= TX_STATE_IDLE;
            end
        endcase
    end

    assign Tx_busy = (state != TX_STATE_IDLE);

endmodule
```

Receiver

```
module receiver(
    input wire Rx,
    output reg ready,
    input wire ready_clr,
    input wire clk_50m,
    input wire clken,
    output reg [7:0] data
);

    parameter RX_STATE_START = 2'b00;
    parameter RX_STATE_DATA = 2'b01;
    parameter RX_STATE_STOP = 2'b10;

    reg [1:0] state;
    reg [3:0] sample;
    reg [3:0] bit_pos;
    reg [7:0] scratch;

    always @(posedge clk_50m) begin
        if (ready_clr)
            ready <= 1'b0;

        if (clken) begin
            case (state)
                RX_STATE_START: begin
                    if (!Rx || sample != 0)
                        sample <= sample + 4'b1;
                    if (sample == 15) begin
                        state <= RX_STATE_DATA;
                        bit_pos <= 0;
                        sample <= 0;
                        scratch <= 0;
                    end
                end
                RX_STATE_DATA: begin
                    sample <= sample + 4'b1;
                    if (sample == 4'h8) begin
                        scratch[bit_pos[2:0]] <= Rx;
```

```
                    bit_pos <= bit_pos + 4'b1;
                end
                if (bit_pos == 8 && sample == 15)
                    state <= RX_STATE_STOP;
            end
            RX_STATE_STOP: begin
                if (sample == 15 || (sample >= 8 && !Rx)) begin
                    state <= RX_STATE_START;
                    data <= ~scratch;
                    ready <= 1'b1;
                    sample <= 0;
                end
                else
                    sample <= sample + 4'b1;
            end
            default: begin
                state <= RX_STATE_START;
            end
        endcase
    end
end

endmodule
```

Baud Code

```
module baudrate(
    input wire clk_50m,
    output wire Rxclk_en,
    output wire Txclk_en
);

    parameter RX_ACC_MAX = 50000000 / (115200 * 16);
    parameter TX_ACC_MAX = 50000000 / 115200;
    parameter RX_ACC_WIDTH = $clog2(RX_ACC_MAX);
    parameter TX_ACC_WIDTH = $clog2(TX_ACC_MAX);

    reg [RX_ACC_WIDTH - 1:0] rx_acc;
    reg [TX_ACC_WIDTH - 1:0] tx_acc;

    assign Rxclk_en = (rx_acc == 5'd0);
    assign Txclk_en = (tx_acc == 9'd0);

    always @(posedge clk_50m) begin
        if (rx_acc == RX_ACC_MAX[RX_ACC_WIDTH - 1:0])
            rx_acc <= 0;
        else
            rx_acc <= rx_acc + 5'b1;
    end

    always @(posedge clk_50m) begin
```

```
        if (tx_acc == TX_ACC_MAX[TX_ACC_WIDTH - 1:0])
            tx_acc <= 0;
        else
            tx_acc <= tx_acc + 9'b1;
    end

endmodule
```

UART

```
module uart(
    input wire [7:0] data_in,
    input wire wr_en,
    input wire clear,
    input wire clk_50m,
    output wire Tx,
    output wire Tx_busy,
    input wire Rx,
    output wire ready,
    input wire ready_clr,
    output wire [7:0] data_out,
    output [7:0] LEDR,
    output wire Tx2
);

    wire Txclk_en, Rxclk_en;

    baudrate uart_baud(
        .clk_50m(clk_50m),
        .Rxclk_en(Rxclk_en),
        .Txclk_en(Txclk_en)
    );

    transmitter uart_Tx(
        .data_in(data_in),
        .wr_en(wr_en),
        .clk_50m(clk_50m),
        .clken(Txclk_en),
        .Tx(Tx),
        .Tx_busy(Tx_busy)
    );

    receiver uart_Rx(
        .Rx(Rx),
        .ready(ready),
        .ready_clr(ready_clr),
        .clk_50m(clk_50m),
        .clken(Rxclk_en),
        .data(data_out)
    );
```

```
    assign LEDR = data_in;
    assign Tx2 = Tx;

endmodule
```

RTL Viewer

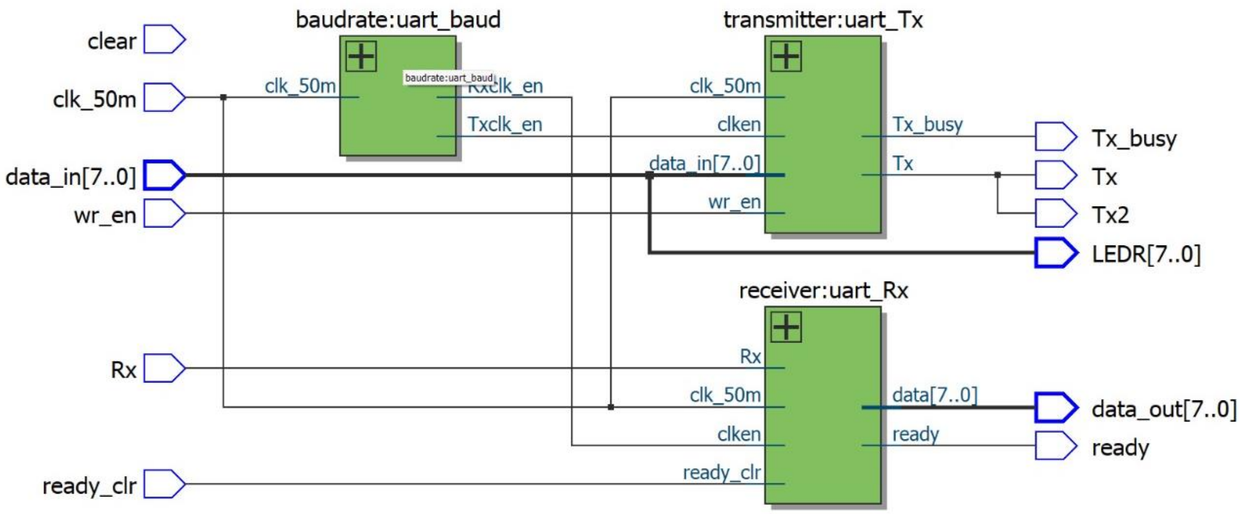

Figure 1 : RTL Viewer

Test Bench

The test bench is made to send 8 bit data from transmitter to receiver. It'll start from 00000000 and increment by 1 bit at a time.

```
module uart_tb;

    // Inputs
    reg [7:0] data_in;
    reg wr_en;
    reg clear;
    reg clk_50m;
    reg Rx;
    reg ready_clr;

    // Outputs
    wire Tx;
    wire Tx_busy;
    wire ready;
    wire [7:0] data_out;
    wire [7:0] LEDR;
    wire Tx2;

    // Instantiate the UART module
```

```
    uart uart_inst (
        .data_in(data_in),
        .wr_en(wr_en),
        .clear(clear),
        .clk_50m(clk_50m),
        .Rx(Rx),
        .ready(ready),
        .ready_clr(ready_clr),
        .data_out(data_out),
        .LEDR(LEDR),
        .Tx(Tx),
        .Tx_busy(Tx_busy),
        .Tx2(Tx2)
    );

    // Clock generation
    always #10 clk_50m = ~clk_50m;

    // Initial values
    initial begin
        // Initialize inputs
        data_in = 8'hFF;
        wr_en = 1'b0;
        clear = 1'b0;
        Rx = 1'b0;
        ready_clr = 1'b0;

        // Apply reset
        clear = 1'b1;
        #100;
        clear = 1'b0;

        // Testbench stimulus
        #100;
        wr_en = 1'b1;
        data_in = 8'hAA; // Set test data
        #100;
        wr_en = 1'b0;
        #100;
        ready_clr = 1'b1; // Clear ready
        #100;
        ready_clr = 1'b0;
        #100;
        data_in = 8'h55; // Set another test data
        wr_en = 1'b1;
        #100;
        wr_en = 1'b0;
        #100;
        $finish;
    end

endmodule
```

Timing Diagram Captured on Simulation

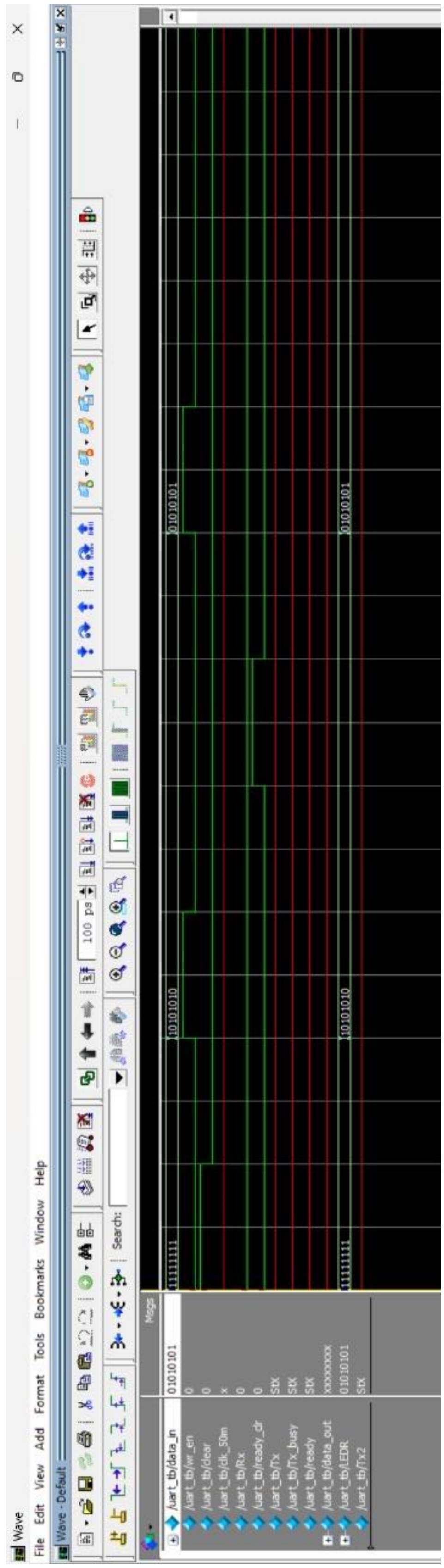

Figure 2 : Timing Diagram Captured on Simulation