University of Moratuwa, Sri Lanka

Department of Electronic & Telecommunication Engineering

FPGA-Based UART Transceiver Design

EN2111 - Electronic Circuit Design

Group 35

Team Members

220074B Boralugoda M.S
220029T Ananthakumar.T
220089B Cooray M.S.T

Contents

1 Abstract

This report presents the complete process of designing and implementing a UART transceiver using an FPGA, structured across four key phases. The initial phase focuses on gaining a solid understanding of the UART communication protocol, reviewing existing Verilog designs, and selecting an appropriate implementation that aligns with the project's requirements. In the second phase, a thorough Verilog testbench is developed to validate the UART module's performance across a variety of scenarios, including edge cases and potential faults. The third phase covers the actual integration of the UART design onto the FPGA, incorporating the required logic for sending and receiving data, and includes simulation, synthesis, and deployment on the hardware.Throughout the project, a strong focus is maintained on understanding UART at a conceptual level, crafting a reliable and optimized design, and ensuring its correctness through detailed verification and testing. The report elaborates on the methodologies adopted, implementation techniques used, and challenges faced throughout each development stage.

2 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.

3 Block Diagram and Maps

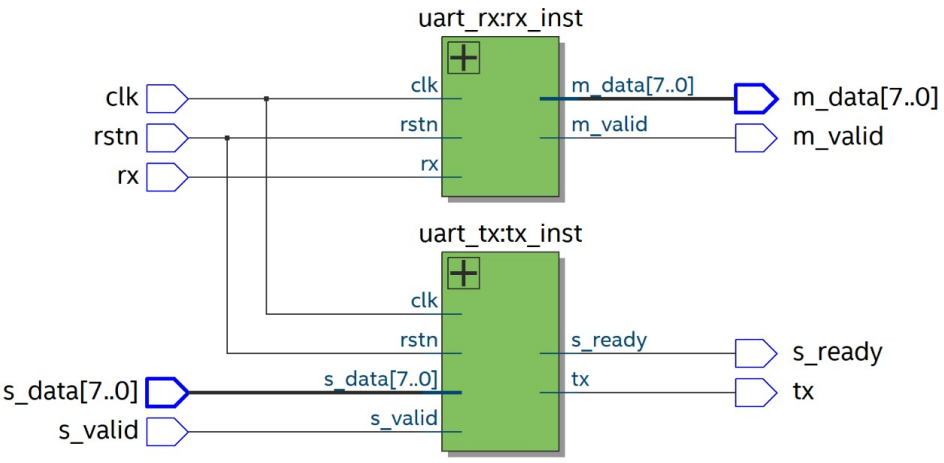

Figure 1: Block Diagram

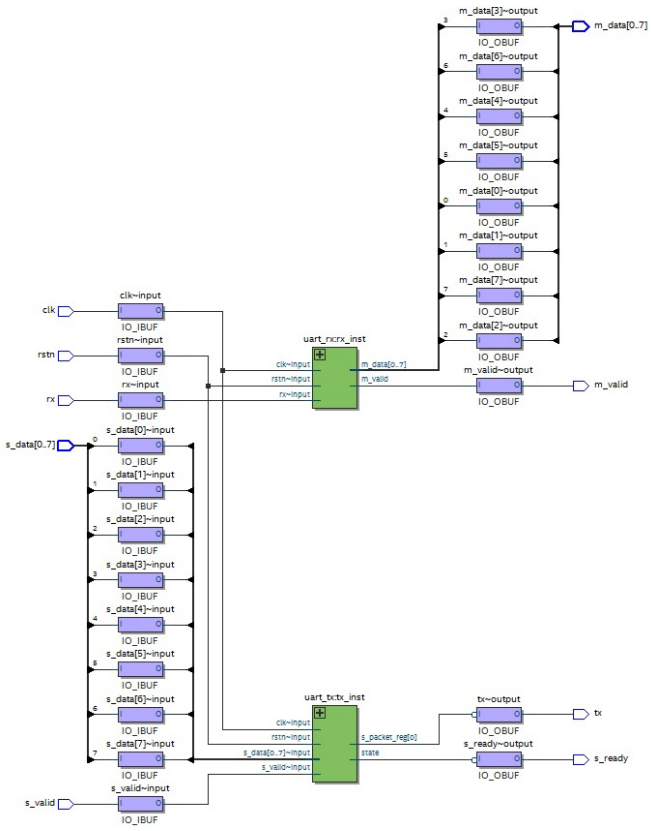

Figure 2: Technology Map

4 UART Design Details

4.1 RTL Code for UART (Top level module)

```
module uart #(
  parameter CLOCKS_PER_PULSE = 5208,  // For 9600 baud at 50 MHz
            BITS_PER_WORD    = 8
)(
  input  logic clk,
  input  logic rstn,

  // TX interface
  input  logic s_valid,                                                    //from data
    source  , input control to the transmitter
  input  logic [BITS_PER_WORD-1:0] s_data,      //from data source  , input data to
    the transmitter
  output logic s_ready,                                                    //to
    the data source
  output logic tx,

  // RX interface
  input  logic rx,
  output logic m_valid,                                                    //data
     output validity
  output logic [BITS_PER_WORD-1:0] m_data               //output data
);

  // Instantiate TX module
  uart_tx #(
    .CLOCKS_PER_PULSE(CLOCKS_PER_PULSE),
    .BITS_PER_WORD(BITS_PER_WORD)
  ) tx_inst (
    .clk(clk),
    .rstn(rstn),
    .s_valid(s_valid),
    .s_data(s_data),
    .tx(tx),
    .s_ready(s_ready)
  );

  // Instantiate RX module
  uart_rx #(
    .CLOCKS_PER_PULSE(CLOCKS_PER_PULSE),
    .BITS_PER_WORD(BITS_PER_WORD)
  ) rx_inst (
    .clk(clk),
    .rstn(rstn),
    .rx(rx),
    .m_valid(m_valid),
    .m_data(m_data)
  );

endmodule
```

4.2 Transmitter

```
module uart_tx #(
  parameter CLOCKS_PER_PULSE = 5208,                // 200 MHz / 9600 baud
            BITS_PER_WORD    = 8,
            PACKET_SIZE      = BITS_PER_WORD + 5     // 1 start + 8 data + 2 stop + (
    optional parity)
)(
  input  logic clk,
  input  logic rstn,
  input  logic s_valid,
  input  logic [BITS_PER_WORD-1:0] s_data,
  output logic tx,
  output logic s_ready
);

  // Constants
  localparam END_BITS = PACKET_SIZE - BITS_PER_WORD - 1; // Number of stop bits (
    assuming 2)

  // FSM States
  typedef enum logic {IDLE, SEND} state_t;
  state_t state;

  // Internal registers
  logic [PACKET_SIZE-1:0] s_packet;
  logic [PACKET_SIZE-1:0] s_packet_reg;
  logic [$clog2(PACKET_SIZE)-1:0] c_bits;
  logic [$clog2(CLOCKS_PER_PULSE)-1:0] c_clocks;

  // Format packet: stop bits (1's), data, start bit (0)
  always_comb begin
    s_packet = { {END_BITS{1'b1}}, s_data, 1'b0 }; // LSB first
  end

  // Output is the LSB of the shift register
  assign tx = s_packet_reg[0];

  // FSM and logic
  always_ff @(posedge clk or negedge rstn) begin
    if (!rstn) begin
      state         <= IDLE;
      s_packet_reg  <= '1; // Idle line is high
      c_bits        <= 0;
      c_clocks      <= 0;
    end else begin
      case (state)
        IDLE: begin
          if (s_valid) begin
            s_packet_reg <= s_packet;
            c_bits       <= 0;
            c_clocks     <= 0;
            state        <= SEND;
          end
        end

        SEND: begin
          if (c_clocks == CLOCKS_PER_PULSE - 1) begin
            c_clocks <= 0;
            if (c_bits == PACKET_SIZE - 1) begin
              state        <= IDLE;
```

```
            s_packet_reg <= '1; // Reset line to idle (high)
          end else begin
            c_bits       <= c_bits + 1;
            s_packet_reg <= s_packet_reg >> 1; // Shift next bit out
          end
        end else begin
          c_clocks <= c_clocks + 1;
        end
      end

      default: state <= IDLE;
    endcase
  end
end

// s_ready indicates the module is ready for new data
assign s_ready = (state == IDLE);

endmodule
```

4.3 Receiver

```
module uart_rx #(
  parameter CLOCKS_PER_PULSE = 5208, // For 9600 baud at 200 MHz
            BITS_PER_WORD    = 8
)(
  input  logic clk,
  input  logic rstn,
  input  logic rx,
  output logic m_valid,
  output logic [BITS_PER_WORD-1:0] m_data
);

  localparam PACKET_SIZE = BITS_PER_WORD + 2;  // 1 start + 8 data + 1 stop
  typedef enum logic [1:0] {IDLE, START, DATA, STOP} state_t;

  state_t state;

  // Internal registers
  logic [$clog2(CLOCKS_PER_PULSE)-1:0] c_clocks;
  logic [$clog2(BITS_PER_WORD)-1:0]    c_bits;
  logic [BITS_PER_WORD-1:0]            shift_reg;

  always_ff @(posedge clk or negedge rstn) begin
    if (!rstn) begin
      state      <= IDLE;
      c_clocks   <= 0;
      c_bits     <= 0;
      shift_reg  <= 0;
      m_data     <= 0;
      m_valid    <= 0;
    end else begin
      m_valid <= 0;  // default

      case (state)
        // Wait for falling edge of start bit
        IDLE: if (rx == 0) begin
          state    <= START;
          c_clocks <= 0;
        end
```

```
        // Wait until middle of start bit
        START: if (c_clocks == CLOCKS_PER_PULSE / 2 - 1) begin
          c_clocks <= 0;
          c_bits   <= 0;
          state    <= DATA;
        end else begin
          c_clocks <= c_clocks + 1;
        end

        // Sample 8 data bits
        DATA: if (c_clocks == CLOCKS_PER_PULSE - 1) begin
          c_clocks <= 0;
          shift_reg <= {rx, shift_reg[BITS_PER_WORD-1:1]};  // LSB first

          if (c_bits == BITS_PER_WORD - 1) begin
            state <= STOP;
          end else begin
            c_bits <= c_bits + 1;
          end
        end else begin
          c_clocks <= c_clocks + 1;
        end

        // Wait one stop bit
        STOP: if (c_clocks == CLOCKS_PER_PULSE - 1) begin
          c_clocks <= 0;
          m_data   <= shift_reg;
          m_valid  <= 1;
          state    <= IDLE;
        end else begin
          c_clocks <= c_clocks + 1;
        end
      endcase
    end
  end

endmodule
```

4.4 Test Bench

```
`timescale 1ns / 1ps

module uart_tb;

  // Parameters
  parameter CLOCKS_PER_PULSE = 10;  // Small for simulation speed
  parameter BITS_PER_WORD    = 8;

  // Signals
  logic clk = 0, rstn = 0;
  logic s_valid, s_ready;
  logic [BITS_PER_WORD-1:0] s_data;
  logic tx, rx;
  logic m_valid;
  logic [BITS_PER_WORD-1:0] m_data;

  // Clock generation
  always #5 clk = ~clk;  // 100 MHz
```

```
  // Instantiate UART module (which includes TX and RX)
  uart #(
    .CLOCKS_PER_PULSE(CLOCKS_PER_PULSE),
    .BITS_PER_WORD(BITS_PER_WORD)
  ) uart_inst (
    .clk(clk),
    .rstn(rstn),
    .s_valid(s_valid),
    .s_data(s_data),
    .tx(tx),
    .s_ready(s_ready),
    .rx(rx),
    .m_valid(m_valid),
    .m_data(m_data)
  );

  // Connect tx to rx for loopback
  assign rx = tx;

  // Stimulus
  initial begin
    // Test vector
    logic [BITS_PER_WORD-1:0] test_data [0:2];
    int i;

    test_data[0] = 8'hA5;
    test_data[1] = 8'h3C;
    test_data[2] = 8'hF0;

    // Initial reset and setup
    $display("Starting simulation...");
    rstn = 0;
    s_valid = 0;
    #50;
    rstn = 1;

    // Test loop to send multiple bytes
    for (i = 0; i < 3; i++) begin
      // Wait until TX is ready for a new byte
      @(posedge clk);
      wait (s_ready);
      s_data = test_data[i];
      s_valid = 1;
      @(posedge clk);
      s_valid = 0;  // Pulse s_valid for one clock cycle

      // Wait for RX to receive data and assert m_valid
      wait (m_valid);
      $display("TX: %02X -> RX: %02X %s", test_data[i], m_data,
               (m_data == test_data[i]) ? "PASS" : "FAIL");

      // Add some delay between transmissions
      repeat (5) @(posedge clk);
    end

    $display("Simulation complete.");
    #50;
    $finish;
  end

endmodule
```

5 ModelSim Simulation and Timing Diagram

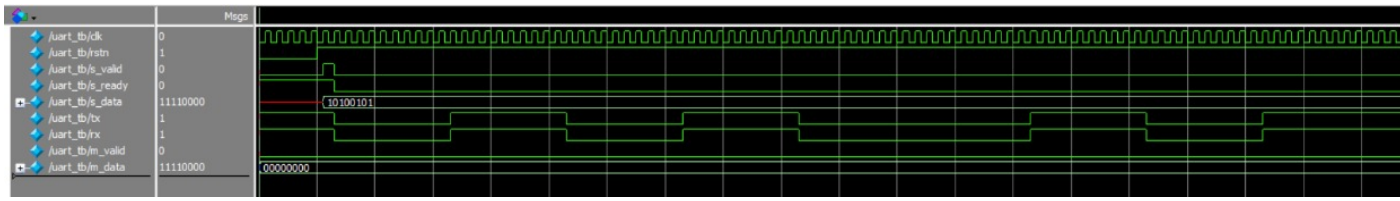

Figure 3: Waveform upon initiating transmission

6 Hardware Testing

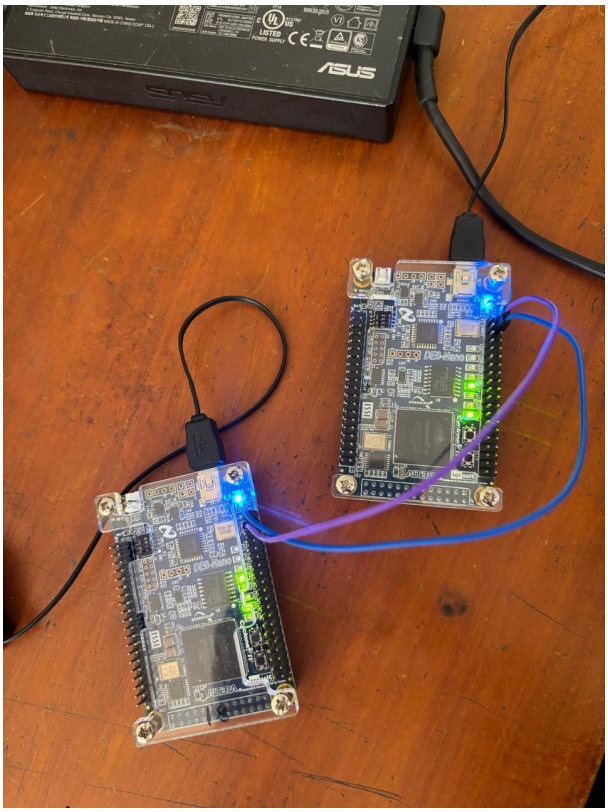

Figure 4: Cross-verification of UART communication between two FPGA boards

7 Pin Configuration

Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatior	I/O Standard	Reserved	urrent Streng	Slew Rate	ifferential Pai	ict Preservati
clk	Input	PIN_R8	3	B3_N0	PIN_R8	2.5 V		8mA (default)			
m_data[7]	Output				PIN_F13	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[6]	Output				PIN_J13	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[5]	Output				PIN_G15	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[4]	Output				PIN_F15	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[3]	Output				PIN_B16	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[2]	Output				PIN_F14	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[1]	Output				PIN_G16	2.5 V ...fault)		8mA (default)	2 (default)		
m_data[0]	Output				PIN_J14	2.5 V ...fault)		8mA (default)	2 (default)		
m_valid	Output				PIN_J16	2.5 V ...fault)		8mA (default)	2 (default)		
rstn	Input	PIN_J15	5	B5_N0	PIN_J15	2.5 V		8mA (default)			
rx	Input	PIN_C3	8	B8_N0	PIN_C3	2.5 V		8mA (default)			
s_data[7]	Input				PIN_E1	2.5 V ...fault)		8mA (default)			
s_data[6]	Input				PIN_L1	2.5 V ...fault)		8mA (default)			
s_data[5]	Input				PIN_J2	2.5 V ...fault)		8mA (default)			
s_data[4]	Input				PIN_J1	2.5 V ...fault)		8mA (default)			
s_data[3]	Input				PIN_L2	2.5 V ...fault)		8mA (default)			
s_data[2]	Input				PIN_L3	2.5 V ...fault)		8mA (default)			
s_data[1]	Input				PIN_K5	2.5 V ...fault)		8mA (default)			
s_data[0]	Input				PIN_N2	2.5 V ...fault)		8mA (default)			
s_ready	Output				PIN_K2	2.5 V ...fault)		8mA (default)	2 (default)		
s_valid	Input				PIN_K1	2.5 V ...fault)		8mA (default)			
tx	Output	PIN_D3	8	B8_N0	PIN_D3	2.5 V		8mA (default)	2 (default)		

Figure 5: FPGA Pin Assignments

8 Conclusion

We successfully implemented a UART transceiver on the DE0-Nano FPGA using SystemVerilog. The design incorporated both transmitter and receiver modules, which were integrated into a top-level module to enable bidirectional communication. The complete system was synthesized, simulated, and validated through both software simulation and physical hardware testing