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

Assignment Report
UART Implementation in FPGA

Group 38
Wickramasinghe S.D. - 220700T
Pirathishanth A. - 220480P

Submitted in partial fulfillment of the requirements for the module
EN 2111 Electronic Circuit Design

5/23/2025

Contents

1 Introduction

Universal Asynchronous Receiver/Transmitter (UART) is a widely used protocol for asynchronous serial communication in embedded and digital systems. It enables reliable data exchange between devices using minimal wiring, typically requiring a transmit (TX) and receive (RX) line. This project involves the design and implementation of a UART communication system using SystemVerilog, including transmitter and receiver modules developed at the RTL (Register Transfer Level). A testbench was created to simulate and validate functionality, and the complete system was deployed on an FPGA for real-time verification. The project also includes capturing timing diagrams through simulation tools, confirming proper bit-level operation of the UART protocol.

2 RTL Code for UART

2.1 Module

```
// Top-level UART module for serial communication
module UART_Module(
  input         clk,
  input         tx_ctr,
  input         rst,
  input         Rx,        // Serial input (receive)
  output        Tx,        // D12 Serial output (transmit)
  output [7:0] data,       // Received data output
  output reg [7:0] tx_data
);
  // Internal signals        // Unused TX debug signal
  reg         clkn=0;         // Divided clock for TX/RX
  reg [7:0] fixed_data; // Fixed data to transmit
  reg [31:0] counter = 0;  // Clock divider counter
  reg [31:0] countbyte=0;
  reg [63:0] buff =64'h85; //   00010010 00110100 01010110 00010001
  wire stat;
  reg flg=1;

  // Clock divider to generate baud rate clock
  always @(posedge clk) begin
    counter <= counter + 1;
         tx_data=fixed_data;
    if (counter == 100000) begin // 325Incorrect: Should be CLKS_PER_BIT/2 (
        ↪ e.g., 5208/2)
      counter <= 1;
                countbyte<=countbyte+1;
      clkn <= ~clkn;
    end
         if (stat&& flg)
                  begin
                         fixed_data<=buff[7:0];
                         flg<=0;
                  end
         else if (~flg && ~stat) begin
                //buff<= {8'h00,buff[63:8]};
                flg<=1;
         end

         end

```

```
  // Instantiate TX module EP4CE22F17C6
  UART_tx TX (
    .clk(clkn),
    .data(fixed_data),
    .data_out(Tx),
    .rst_n(rst),    // Hardcoded: Should be input   // Hardcoded: Should be
       ↪ controlled
    .status(stat),
         .tx_ctr(tx_ctr)
  );

  // Instantiate RX module
  UART_rx RX (
    .clk(clkn),
    .data_in(Rx),
    .data_out(data),           // Unconnected
    .rst_n(rst)

         // Unconnected
  );

endmodule
//PIN_A11
//PIN_B13
//PIN_A13
//PIN_A15
```

2.2 Transmitter

```
module UART_tx (
  input         clk,
  input         rst_n,     // Active-low reset
  input [7:0]   data,      // Data to transmit
  output reg    data_out,  // Serial output
  output reg    status,
  input         tx_ctr
);
  parameter IDLE  = 2'b00, START = 2'b01, DATA = 2'b10, STOP = 2'b11;
  parameter CLKS_PER_BIT = 16;
  parameter CLKSidel = 50;
  reg [7:0]  data_buff=0;     // Data buffer for transmission
  reg        curr_stat;       // Tracks start status
  reg [19:0] clk_counter;     // Counts clock cycles per bit
  reg [1:0]  STATE = IDLE;    // State machine register
  reg [3:0]  bit_index = 0;   // Tracks transmitted bits


  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      data_out    <= 1;
      data_buff   <= data;
      clk_counter <= 0;
                status      <= 1;
    end else begin
      case (STATE)
        IDLE: begin
          if (clk_counter < CLKSidel ) begin
```

```
            data_out    <= 1;
            data_buff   <= data;
                                clk_counter <= clk_counter +1;
                                status          <=1;
          end else begin
            STATE               <= START;
            status              <= 0;
                                clk_counter <= 0;
          end
        end
        START: begin
          if (clk_counter < CLKS_PER_BIT-1) begin
            data_out    <= 0;
                                data_buff   <= data;
            clk_counter <= clk_counter + 1;
          end else begin
            clk_counter <= 0;
            STATE               <= DATA;
            bit_index   <= 0;
          end
        end
        DATA: begin
          if (bit_index < 8) begin
            STATE <= DATA;
            if (clk_counter < CLKS_PER_BIT-1) begin
              data_out          <= data_buff[0];   // Send LSB
              clk_counter  <= clk_counter + 1;
            end else begin
              data_buff         <= data_buff >> 1; // Shift right
              clk_counter  <= 0;
              bit_index         <= bit_index + 1;
            end
          end else begin
            STATE <= STOP;
            clk_counter <= 0;
          end
        end
        STOP: begin
          if (clk_counter < CLKS_PER_BIT-1) begin
            data_out    <= 1;     // Send stop bit
            STATE               <= STOP;
            clk_counter <= clk_counter + 1;
          end else begin
            data_out <= 1;
            STATE       <= IDLE;
                                status  <=1;
          end
        end
        default: STATE <= IDLE;
      endcase
    end
  end

endmodule
```

2.3 Receiver

```
module UART_rx #(
  parameter CLKS_PER_BIT = 16
) (
   input            clk,
   input            rst_n,
   input            data_in,
   output reg [7:0] data_out
);
  parameter IDLE  = 2'b00, START = 2'b01, DATA = 2'b10, STOP = 2'b11;
  reg [7:0]  data_val;
  reg [3:0]  count;
  reg [15:0] clk_counter;
  reg [3:0]  filtercount;
  reg [64:0] data_buffrx;
  reg [1:0]  STATE = IDLE;
  reg [3:0]  bitcount;
  reg        flag = 1;
  reg        statflag = 1;

  always @(posedge clk or negedge rst_n) begin

    if (~rst_n) begin
      count <= 0;
      bitcount <= 0;
      statflag <= 0;
        data_out <= 0;
    end else begin
      case (STATE)
        IDLE: begin
          if (data_in==0) begin
            STATE <= START; // Detect start bit
                clk_counter<=0;
          end
        end
        START: begin
             clk_counter <= clk_counter + 1;
          if (data_in == 0 && clk_counter == CLKS_PER_BIT/2 -1) begin
            STATE <= DATA;
            bitcount <= 0;
                //data_val =8'h00;
                clk_counter<=0;
                data_val<=8'h00;
             end
             else if (data_in == 1 && clk_counter == CLKS_PER_BIT/2 -1)
                 ↪ begin
                STATE <= IDLE;
             end
        end
        DATA: begin
          clk_counter <= clk_counter + 1;
          if (data_in && clk_counter == CLKS_PER_BIT) begin
            data_val <= {1'b1, data_val[7:1]};
                clk_counter<=0;
                bitcount <= bitcount + 1;
          end
             else if (data_in==0 && clk_counter == CLKS_PER_BIT)begin
                data_val <= {1'b0, data_val[7:1]};
```

```
                clk_counter<=0;
                bitcount <= bitcount + 1;
          end
          if (bitcount > 7) begin

            STATE <= STOP;
            clk_counter <= 0;
          end

        end
        STOP: begin
          if (data_in && clk_counter == CLKS_PER_BIT) begin
                  data_out<=data_val;
                  STATE <= IDLE;
             end
             else if (data_in==0 && clk_counter == CLKS_PER_BIT) begin
                  STATE <= IDLE;
             end
            clk_counter <= clk_counter + 1;
          end
        default: STATE <= IDLE;
      endcase
    end
  end

endmodule
```

3 Test Bench

```
`timescale 10ns/1ns

module UART_tb();

  reg clk;
  reg rst;
  reg [7:0] fixed_data;
  wire tx_line;
  wire [7:0] received_data;
  wire [7:0] tx_data;
  wire status;
  wire [3:0] tx_ctr;

  // Clock generation
  initial clk = 0;
  always #5 clk = ~clk;  // 100 MHz clock (10ns period)

  // Reset logic
  initial begin
    rst = 1;
    #20;
    rst = 0;
    #20;
    rst = 1;
  end

  // Instantiate Transmitter
  UART_tx TX (
```

```
    .clk(clk),
    .data(fixed_data),
    .data_out(tx_line),
    .rst_n(rst),
    .status(status),
    .tx_ctr(tx_ctr)
  );

  // Instantiate Receiver
  UART_rx RX (
    .clk(clk),
    .data_in(tx_line),
    .data_out(received_data),
    .rst_n(rst)
  );

  // Test scenario
  initial begin
    fixed_data = 8'b00100100; // ASCII 'A'
    #100000; // Wait for UART to transmit and receive
    $display("Received Data = %h", received_data);
    $finish;
  end

endmodule
```

4 Timing Diagram Captured on Simulation

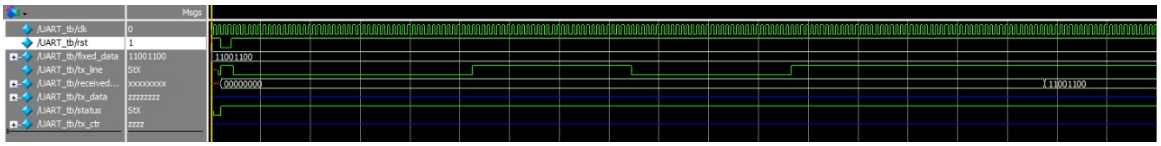

Figure 1: Test Bench Simulation

5 FPGA Implementation

We used a DE0-Nano board to implement this communication system.

The pin planning we used for the board is as follows:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
Rx	Input	PIN_D3	8	B8_N0	PIN_D3	2.5 V		8mA (default)	
Tx	Output	PIN_B12	7	B7_N0	PIN_B12	2.5 V		8mA (default)	2 (default)
clk	Input	PIN_R8	3	B3_N0	PIN_R8	2.5 V		8mA (default)	
data[7]	Output	PIN_L3	2	B2_N0	PIN_L3	2.5 V		8mA (default)	2 (default)
data[6]	Output	PIN_B1	1	B1_N0	PIN_B1	2.5 V		8mA (default)	2 (default)
data[5]	Output	PIN_F3	1	B1_N0	PIN_F3	2.5 V		8mA (default)	2 (default)
data[4]	Output	PIN_D1	1	B1_N0	PIN_D1	2.5 V		8mA (default)	2 (default)
data[3]	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		8mA (default)	2 (default)
data[2]	Output	PIN_B13	7	B7_N0	PIN_B13	2.5 V		8mA (default)	2 (default)
data[1]	Output	PIN_A13	7	B7_N0	PIN_A13	2.5 V		8mA (default)	2 (default)
data[0]	Output	PIN_A15	7	B7_N0	PIN_A15	2.5 V		8mA (default)	2 (default)
rst	Input	PIN_J15	5	B5_N0	PIN_J15	2.5 V		8mA (default)	

Figure 2: Pin Planner for the DE0-Nano Board

The UART transmitter and receiver were successfully designed and tested using the DE0-Nano

FPGA development board. The transmission and reception of serial data were verified by analyzing the UART waveforms on a digital oscilloscope, which confirmed accurate timing and signal integrity. As part of the testing process, we collaborated with another group to validate inter-board communication. Both groups configured their UART transmitters to send predefined 8-bit values. These values were successfully received by the corresponding receiver modules. To further demonstrate functionality, our board was programmed to light up its LEDs in parallel with the received data, multiply the data by 2, and then transmit it back to the other group's board. The other group's receiver module accurately decoded the data and displayed it on their DE0-Nano FPGA development board's LEDs, confirming successful two-way communication and processing.

6 Conclusion

The UART communication system was successfully designed, simulated, and implemented using SystemVerilog. The transmitter and receiver modules functioned as intended, enabling reliable asynchronous serial data transmission. Simulation results and timing diagrams confirmed accurate protocol behavior, including start, data, and stop bit handling. The system was further validated through FPGA deployment and hardware testing using a 7-segment display and an oscilloscope. This project demonstrated the complete design flow—from RTL coding and simulation to hardware implementation, highlighting the importance of UART in digital communication systems and reinforcing practical skills in SystemVerilog and FPGA development.