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

Lab Assignment

UART Implementation in FPGA

By

Group - 5

Basnayake B.M.N.S - 220072R
Budvin M.P.L - 220077L
De Alwis W.M.R - 220097X

EN 2111 - Electronic Control Design

10th May 2025

Abstract

This report presents the design and implementation of a UART transceiver using Verilog HDL on an FPGA. The Universal Asynchronous Receiver/Transmitter (UART) is a communication protocol used for serial data exchange between two devices. The project includes the Verilog implementation of transmitter and receiver modules, baud rate generator, and a top-level module integrating all components. Simulation results and practical testing on a DE2-115 FPGA are also presented.

Contents

1 Introduction

The DE2-115 FPGA board can be used to implement a UART transceiver. UART is a communication protocol for serial data exchange between two devices. In UART, the transmitter converts parallel data to serial form and transmits it, while the receiver converts the received serial data back to parallel form.

UART communication is asynchronous, meaning there is no clock signal for synchronization. Start and stop bits are used to mark the beginning and end of data packets. The receiver reads incoming bits at a specific baud rate, which determines the data transfer speed. Matching baud rates between the transmitter and receiver is crucial for successful data transfer.

To implement UART on the DE2-115 FPGA board, you need to design the UART module in Verilog, write a testbench for simulation, and program the FPGA with the design. Connect the Tx and Rx pins of the DE2-115 board to the corresponding pins of the device you want to communicate with.

2 UART Code

2.1 Code Explanation

The receiving part of the module consists of a simple state machine with two states: 'IDLE' and 'READ'.

In the 'IDLE' state, the receiver signal is kept at high level (3.3 V). When data is being received, it starts with a low state pulse. When the receiver detects this start bit it changes its state to 'READ'. From then onwards, 8-bits are read from the receiver. Apart from the clock, another pulse stream called the 'tick' is used to read these bits. The 'tick' stream has a frequency which is 16 times that of the baud rate of the transmission. This allows reading each bit at the middle of the pulse. When the starting bit is detected (negative edge from the 'IDLE' state):

1. The counter counts 8 ticks to read the start bit.
2. Then the counter counts 16 ticks to read each of the 8 data bits.
3. Finally, the counter counts 16 ticks to read the stop bit (which should be high).
4. After reading these 8-bits, the receiver changes its state to 'IDLE' again.

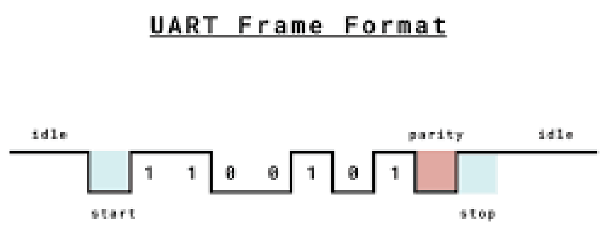

Figure 1: UART Data Frame

2.2 Code

2.2.1 Transmitter Module

```
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,
```

```verilog
  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
```

Listing 1: Transmitter Module Code

2.2.2 Receiver Module

```
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;
          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
```

Listing 2: Receiver Module Code

2.2.3 Top 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
  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),
    .status(stat),
    .tx_ctr(tx_ctr)
  );

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

```

```
endmodule
```

Listing 3: Top Level Entity Code

2.3 Testbench

```
‘timescale 10ns/1ns

module UART_tb();

  reg clk;
  reg rst;
  reg txrst;
  wire tx_line;
  wire [7:0]tx_data;
  wire [7:0] received_data;

  UART_Module uut (
    .clk(clk),
    .rst(rst),
    .Rx(tx_line),
    .Tx(tx_line),
    .tx_data(tx_data),
    .data(received_data)
  );

  initial begin
    clk = 0;
    rst =1;
    #10;
    rst =0;
    #2;
    rst =1;
  end

  always begin
    #1 clk = ~clk;
  end

  initial begin
    #1000000;
  end

endmodule
```

Listing 4: Testbench Code

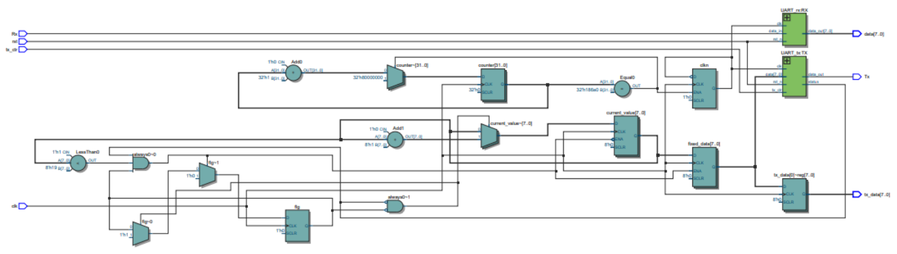

Figure 2: RTL Viewer of the System

3 Simulation Results

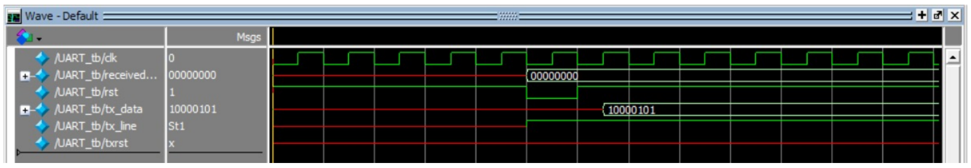

Figure 3: Simulation Waveforms

4 Testing

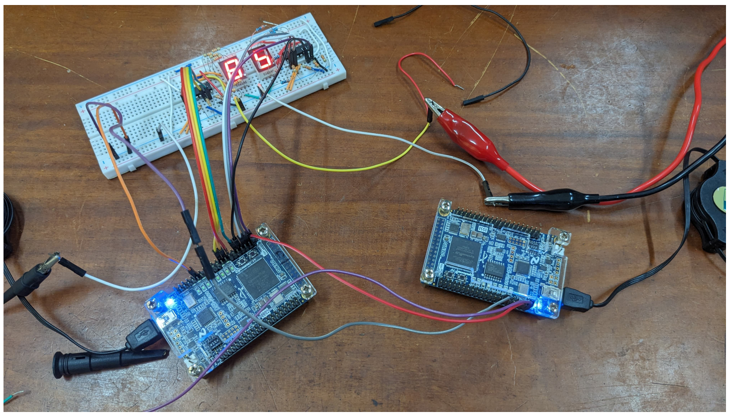

Figure 4: Transferring Data Between Two FPGAs

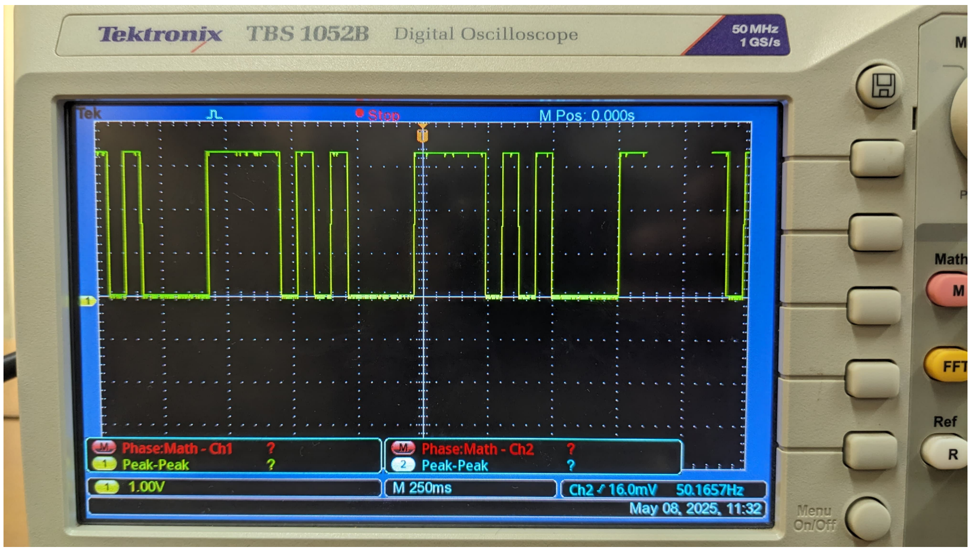

Figure 5: Transmitted Waveform