Department of Electronic & Telecommunication Engineering

University of Moratuwa

EN2111 - Electronic Circuit Design

UART Implementation in FPGA

Group 09

Name: *Alahakoon U.M.Y.B (210027C)*
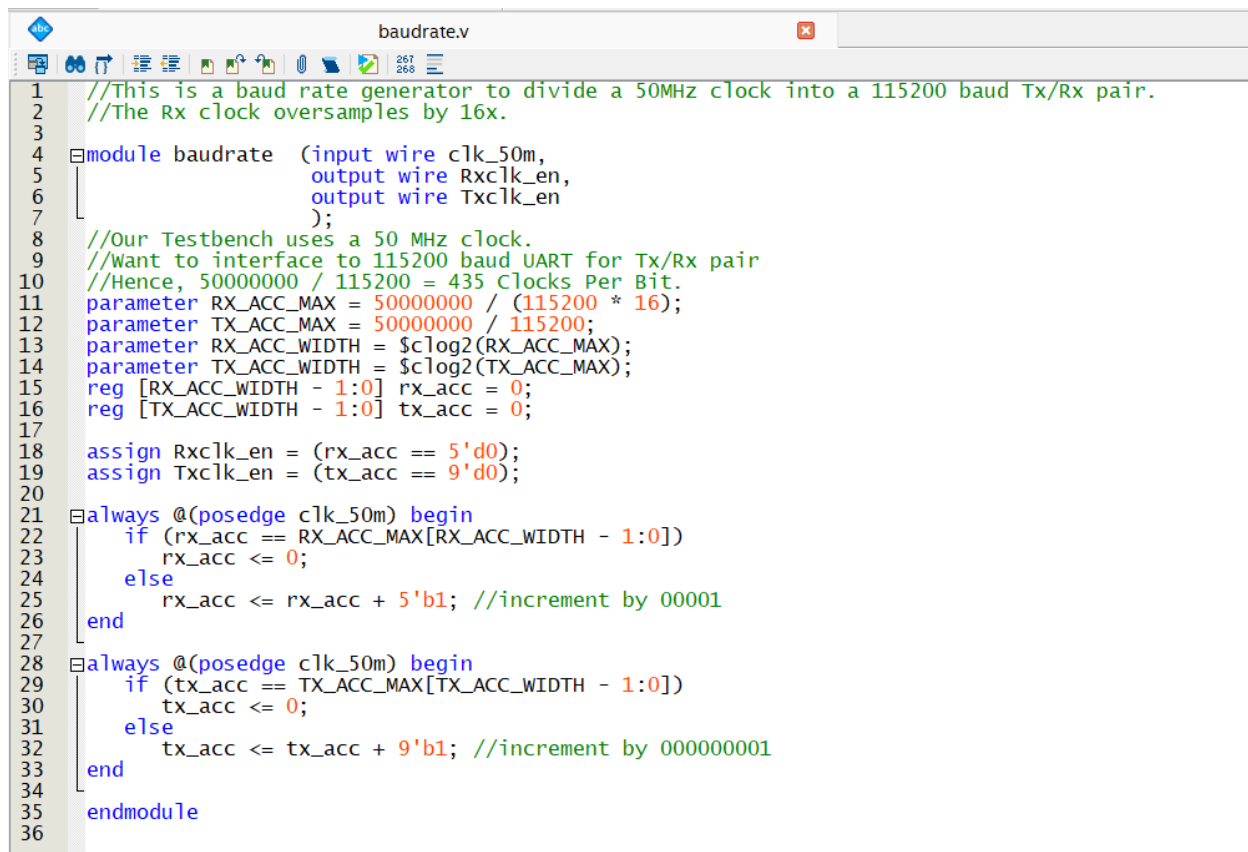
Ahameth M.J (210024N)

Date: *08/05/2024*

Introduction

This report is about setting up a UART transceiver and receiver on the DE0-Nano FPGA board. There are three main parts which are the simulation timing diagram, the test benches that go with it, and the RTL codes. UART is a way for devices to talk to each other without needing a clock signal. It uses start and stop bits to show where data begins and ends. The RTL code explains in detail how the UART module handles data packets, acting as a model for the module's working. The test benches are carefully made to check that the module works well in different situations. Also, the timing diagram from the simulation helps us see when data is sent and received, guiding us on how to make communication better. Once the FPGA is set up with this, the Tx and Rx pins allow devices to talk to each other smoothly.

.

01. RTL code for UART

- Baud Rate Code

```verilog
//This is a baud rate generator to divide a 50MHz clock into a 115200 baud Tx/Rx pair.
//The Rx clock oversamples by 16x.

module baudrate  (input wire clk_50m,
                  output wire Rxclk_en,
                  output wire Txclk_en
                  );
//Our Testbench uses a 50 MHz clock.
//Want to interface to 115200 baud UART for Tx/Rx pair
//Hence, 50000000 / 115200 = 435 Clocks Per Bit.
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 = 0;
reg [TX_ACC_WIDTH - 1:0] tx_acc = 0;

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; //increment by 00001
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; //increment by 000000001
end

endmodule
```

• Transmitter Code

```verilog
1    module transmitter(  input wire [7:0] data_in, //input data as an 8-bit regsiter/vector
2                         input wire wr_en, //enable wire to start
3                         input wire clk_50m,
4                         input wire clken, //clock signal for the transmitter
5                         output reg Tx, //a single 1-bit register variable to hold transmitting
     bit
6                         output wire Tx_busy //transmitter is busy signal
7                         );
8
9    initial begin
10       Tx = 1'b1; //initialize Tx = 1 to begin the transmission
11   end
12   //Define the 4 states using 00,01,10,11 signals
13   parameter TX_STATE_IDLE = 2'b00;
14   parameter TX_STATE_START  = 2'b01;
15   parameter TX_STATE_DATA  = 2'b10;
16   parameter TX_STATE_STOP = 2'b11;
17
18   reg [7:0] data = 8'h00; //set an 8-bit register/vector as data,initially equal to 00000000
19   reg [2:0] bit_pos = 3'h0; //bit position is a 3-bit register/vector, initially equal to 000
20   reg [1:0] state = TX_STATE_IDLE; //state is a 2 bit register/vector,initially equal to 00
21
22   assign Tx_busy = (state != TX_STATE_IDLE); //We assign the BUSY signal when the transmitter
     is not idle
23
24   always @(posedge clk_50m) begin
25       case (state) //Let us consider the 4 states of the transmitter
26       TX_STATE_IDLE: begin //We define the conditions for idle  or NOT-BUSY state
27           if (wr_en) begin
28               state <= TX_STATE_START; //assign the start signal to state
29               data <= data_in; //we assign input data vector to the current data
30               bit_pos <= 3'h0; //we assign the bit position to zero
31           end
32       end
33       TX_STATE_START: begin //We define the conditions for the transmission start state
34           if (clken) begin
35               Tx <= 1'b0; //set Tx = 0 indicating transmission has started
36               state <= TX_STATE_DATA;
37           end
38       end
39       TX_STATE_DATA: begin
40           if (clken) begin
41               if (bit_pos == 3'h7) //we keep assigning Tx with the data until all bits have been
     transmitted from 0 to 7
42                   state <= TX_STATE_STOP; // when bit position has finally reached 7, assign
     state to stop transmission
43               else
44                   bit_pos <= bit_pos + 3'h1; //increment the bit position by 001
45               Tx <= data[bit_pos]; //Set Tx to the data value of the bit position ranging from 0-7
46           end
47       end
48       TX_STATE_STOP: begin
49           if (clken) begin
50               Tx <= 1'b1; //set Tx = 1 after transmission has ended
51               state <= TX_STATE_IDLE; //Move to IDLE state once a transmission has been completed
52           end
53       end
54       default: begin
55           Tx <= 1'b1; // always begin with Tx = 1 and state assigned to IDLE
56           state <= TX_STATE_IDLE;
57       end
58       endcase
59   end
60
61
62   endmodule
63
```

Receiver Code

```verilog
module receiver (input wire Rx,
                output reg ready,          // default 1 bit reg
                input wire ready_clr,
                input wire clk_50m,
                input wire clken,
                input wire RX_en,
                input wire reset,
                output reg [7:0] data    // 8 bit register
                );

initial begin
    ready = 1'b0; // initialize ready = 0
    data = 8'b0; // initialize data as 00000000
end
// Define the 4 states using 00,01,10 signals
parameter RX_STATE_START   = 2'b00;
parameter RX_STATE_DATA    = 2'b01;
parameter RX_STATE_STOP    = 2'b10;

reg [1:0] state = RX_STATE_START; // state is a 2-bit register/vector,initially equal to 00
reg [3:0] sample = 0; // This is a 4-bit register
reg [3:0] bit_pos = 0; // bit position is a 4-bit register/vector, initially equal to 000
reg [7:0] scratch = 8'b0; // An 8-bit register assigned to 00000000

always @(posedge clk_50m) begin
    if (reset) begin
        data = 0;
        scratch = 0;
    end

    if (RX_en) begin

        if (ready_clr)
            ready <= 1'b0; // This resets ready to 0

        if (clken) begin
            case (state) // Let us consider the 3 states of the receiver
            RX_STATE_START: begin // We define condtions for starting the receiver
                if (!Rx || sample != 0) // start counting from the first low sample
                    sample <= sample + 4'b1; // increment by 0001
                if (sample == 15) begin // once a full bit has been sampled
                    state <= RX_STATE_DATA; // start collecting data bits
                    bit_pos <= 0;
                    sample <= 0;
                    scratch <= 0;
                end
            end
            RX_STATE_DATA: begin // We define conditions for starting the data colleting
                sample <= sample + 4'b1; // increment by 0001
                if (sample == 4'h8) begin // we keep assigning Rx data until all bits have 01
to 7
                    scratch[bit_pos[2:0]] <= Rx;
                    bit_pos <= bit_pos + 4'b1; // increment by 0001
                end
                if (bit_pos == 8 && sample == 15) // when a full bit has been sampled and
                    state <= RX_STATE_STOP; // bit position has finally reached 7, assign state
to stop
            end
            RX_STATE_STOP: begin
                /*
                 * Our baud clock may not be running at exactly the
                 * same rate as the transmitter.  If we thing that
                 * we're at least half way into the stop bit, allow
                 * transition into handling the next start bit.
                 */
                if (sample == 15 || (sample >= 8 && !Rx)) begin
                    state <= RX_STATE_START;
                    data <= scratch;
                    ready <= 1'b1;
                    sample <= 0;
                end
                else begin
                    sample <= sample + 4'b1;
                end
```

```verilog
            end
            default: begin
                state <= RX_STATE_START; // always begin with state assigned to START
            end
            endcase
        end
    end
end

endmodule
```

UART Code

```verilog
module uart(output wire [7:0] data_out,
            output wire data_valid,
            output wire TX,
            input wire [7:0] data_in,
            input wire clk,
            input wire TX_en,
            input wire RX_en,
            input wire RX,
            input wire reset
            );


reg ready_clr = 0;
wire Txclk_en, Rxclk_en;

baudrate uart_baud(   .clk_50m(clk), //50MHz Clock
                      .Rxclk_en(Rxclk_en), // Reciever sampling clock 16 samples per bit
                      .Txclk_en(Txclk_en) // Transmitter clock 115,200 bps
                      );

transmitter uart_Tx( .data_in(data_in),
                     .wr_en(TX_en),
                     .clk_50m(clk),
                     .clken(Txclk_en),
                     .Tx(TX)
                     );

receiver uart_Rx( .Rx(RX),
                  .ready(data_valid),
                  .ready_clr(ready_clr),
                  .clk_50m(clk),
                  .clken(Rxclk_en),
                  .data(data_out),
                  .RX_en(RX_en),
                  .reset(reset)
                  );

endmodule
```

Project Navigator — Files
- Files
  - transmitter_tb.v
  - uart.v
  - transmitter.v
  - receiver_tb.v
  - receiver.v
  - baudrate_tb.v
  - baudrate.v
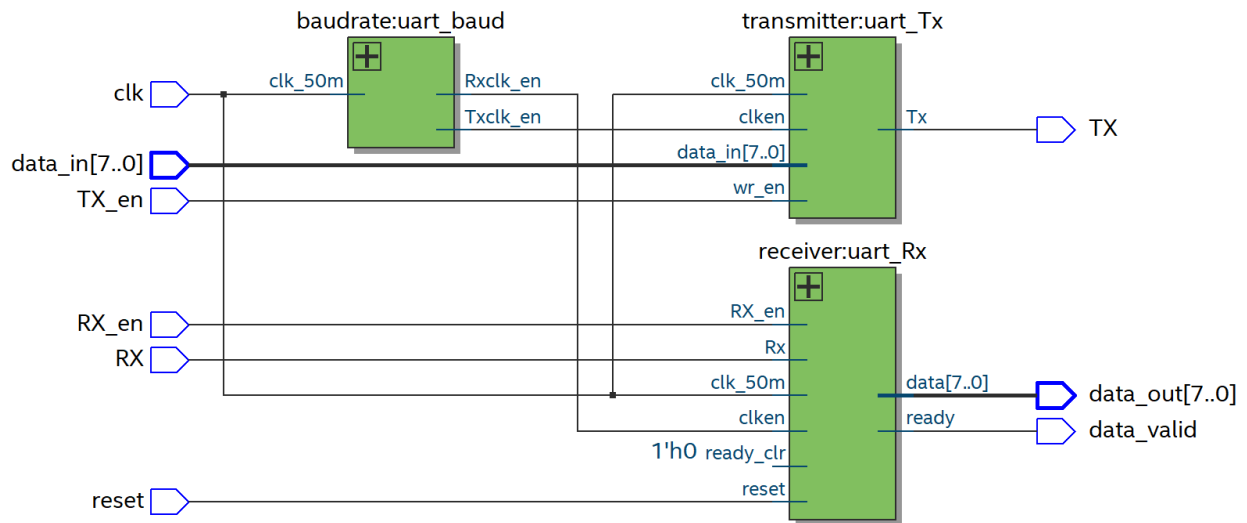
Tasks — Compilation
- Task
  - View Report
  - Analysis & Elaboration
    - Partition Merge
    - Netlist Viewers
    - Design Assistant (Post-Mapp...
    - I/O Assignment Analysis
  - Fitter (Place & Route)
    - Assembler (Generate programm...
    - TimeQuest Timing Analysis
    - EDA Netlist Writer

Compilation Report - UART
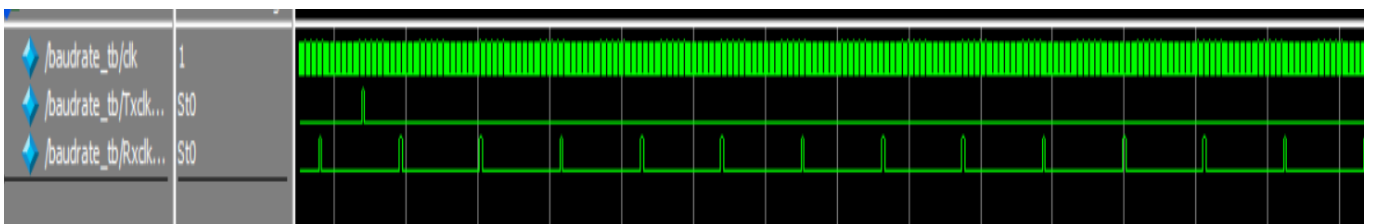
02. <u>Testbenches</u>

- ## Baud Rate Tb


```
 1    // `timescale 10ns/1ps
 2    module baudrate_tb;
 3
 4    reg clk = 0;
 5    wire Txclk_en,Rxclk_en;
 6
 7    baudrate br1(.clk_50m(clk),
 8                 .Rxclk_en(Rxclk_en)
 9                 .Txclk_en(Txclk_en)
10                 );
11
12    always
13       begin
14          #1 clk = ~clk;
15       end
16
17    endmodule
```
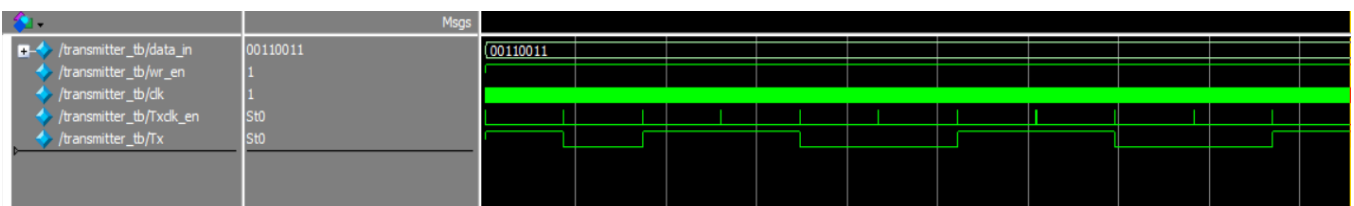
Transmitter Tb


```verilog
1   module transmitter_tb;
2
3       reg [7:0] data_in = 8'b00110011 ;
4       reg wr_en = 1;
5       reg clk = 0;
6       wire Txclk_en,Rxclk_en;
7       wire Tx;
8       wire Tx_busy;
9
10      baudrate br1(.clk_50m(clk),
11                   .Rxclk_en(Rxclk_en),
12                   .Txclk_en(Txclk_en)
13                   );
14
15      transmitter tr1(.data_in(data_in),
16                   .wr_en(wr_en),
17                   .clk_50m(clk),
18                   .clken(Txclk_en),
19                   .Tx(Tx),
20                   .Tx_busy(Tx_busy)
21                   );
22
23      initial #5 wr_en = 1;
24
25      always
26          begin
27              #1 clk = ~clk;
28          end
29
30      endmodule
```


Receiver Tb

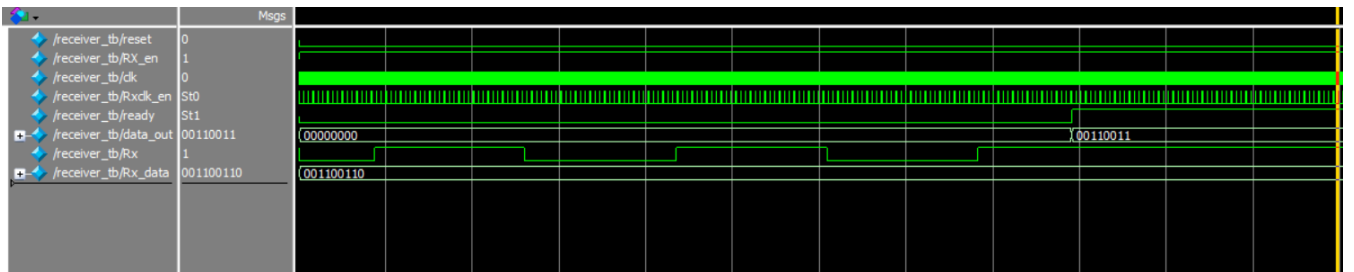Date: May 07, 2024   receiver_tb.v   Project: UART

```verilog
1   module receiver_tb;
2
3       reg reset = 0;
4       reg RX_en = 1;
5       reg clk = 0;
6       wire Txclk_en,Rxclk_en;
7       reg ready_clr = 0;
8       wire ready;
9       wire [7:0] data_out;
10      reg Rx = 1'b1;
11      reg [8:0] Rx_data = 9'b001100110;
12      integer i = 0;
13
14      baudrate br1(.clk_50m(clk),
15                   .Rxclk_en(Rxclk_en),
16                   .Txclk_en(Txclk_en)
17                   );
18
19      receiver rx1(.Rx(Rx),
20                   .ready(ready),
21                   .ready_clr(ready_clr),
22                   .clk_50m(clk),
23                   .clken(Rxclk_en), //We assign Tx clock to enable clock
24                   .data(data_out),
25                   .RX_en(RX_en),
26                   .reset(reset)
27                   );
28
29
30      always
31          begin
32              #1 clk = ~clk;
33          end
34
35      always @(posedge Txclk_en)
36          begin
37              if (i!=9) begin
38                  Rx = Rx_data[i];
39                  i = i+1;
40              end
41              else Rx = 1;
42          end
43
44
45      endmodule
```

- ## UART Tb


```
                                              uart_tb.v
  1    module uart_tb;
  2
  3    wire [7:0] data_out;
  4    wire data_valid;
  5    reg [7:0] data_in = 8'b00110011;
  6    reg clk = 0;
  7    reg TX_en = 1;
  8    reg RX_en = 1;
  9    reg reset = 0;
 10    wire loopback;
 11
 12    uart u1(.data_out(data_out),
 13            .data_valid(data_valid),
 14            .TX(loopback),
 15            .data_in(data_in),
 16            .clk(clk),
 17            .TX_en(TX_en),
 18            .RX_en(RX_en),
 19            .RX(loopback),
 20            .reset(reset)
 21            );
 22
 23    initial #5 TX_en = 1;
 24
 25    always
 26       begin
 27          #1 clk = ~clk;
 28       end
 29
 30
 31    endmodule
```
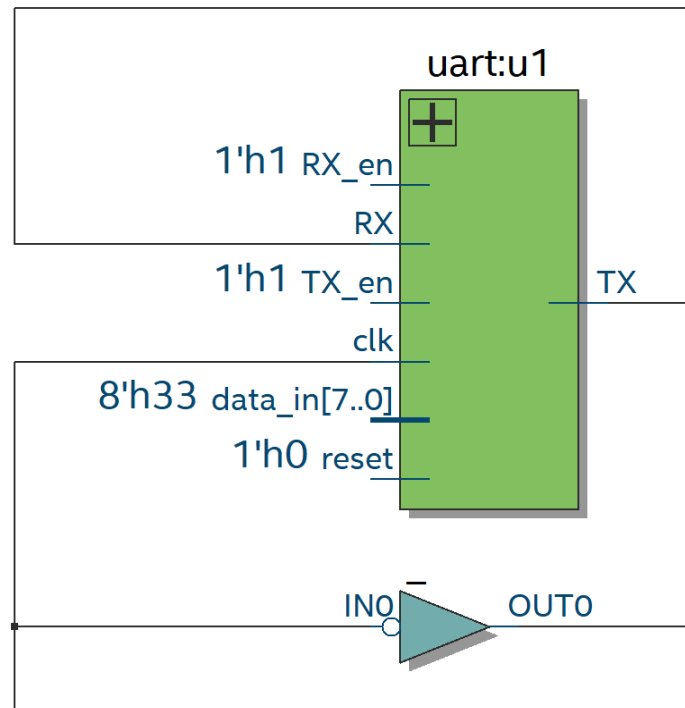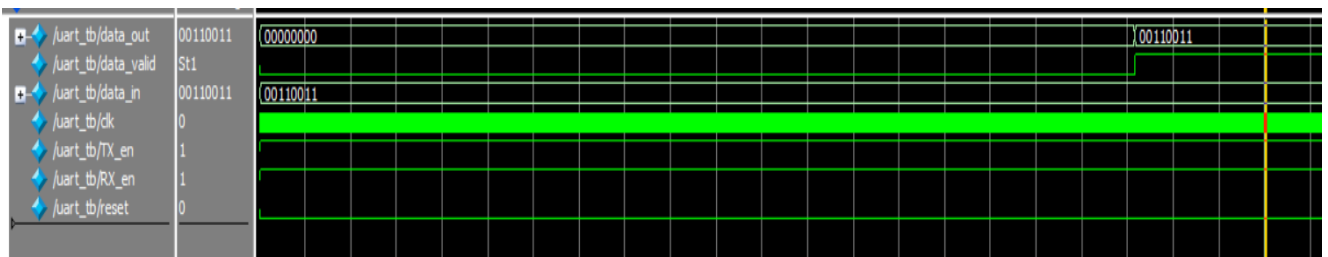
03.   Timing Analysis

****