Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN2111 Electronic Circuit Design

UART Implementation on FPGA

Cooray P.L.L.K. - 210088L

Dabare A.D.T. - 210089P

Dassanayake D.M.T.K. - 210095F

Group No. : **12**

Date : 05/08/2024

Content

1. Introduction

UART, a popular serial communication protocol, enables devices to exchange data efficiently. Unlike bulky parallel connections, UART transmits data one bit at a time over a single wire. This simplicity comes at a cost - there's no constant clock signal for synchronization. To overcome this, UART utilizes start and stop bits to mark the beginning and end of data packets, ensuring the receiving device interprets information correctly. Additionally, both devices must agree on a specific transmission speed (baud rate) for error-free communication. Implementing UART on a DE0-Nano FPGA board involves creating a Verilog program that defines the data conversion process and incorporates start/stop bits. This program is then tested and uploaded to the board. Finally, the board's designated pins (Tx for transmit and Rx for receive) are connected to the other device, allowing them to seamlessly exchange data.

2. Verilog HDL Code

2.1 Transmitter

```
module transmitter(  input wire [7:0] data_in, //input data as an 8-bit regsiter/vector
                     input wire wr_en, //enable wire to start
                     input wire clk_50m,
                     input wire clken, //clock signal for the transmitter
                     output reg Tx, //a single 1-bit register variable to hold transmitting bit
                     output wire Tx_busy //transmitter is busy signal
                     );

initial begin
    Tx = 1'b1; //initialize Tx = 1 to begin the transmission
end
//Define the 4 states using 00,01,10,11 signals
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 = 8'h00; //set an 8-bit register/vector as data,initially equal to 00000000
reg [2:0] bit_pos = 3'h0; //bit position is a 3-bit register/vector, initially equal to 000
reg [1:0] state = TX_STATE_IDLE; //state is a 2 bit register/vector,initially equal to 00

always @(posedge clk_50m) begin
   case (state) //Let us consider the 4 states of the transmitter
   TX_STATE_IDLE: begin //We define the conditions for idle  or NOT-BUSY state
      if (~wr_en) begin
         state <= TX_STATE_START; //assign the start signal to state
         data <= data_in; //we assign input data vector to the current data
         bit_pos <= 3'h0; //we assign the bit position to zero
      end
   end
   TX_STATE_START: begin //We define the conditions for the transmission start state
      if (clken) begin
         Tx <= 1'b0; //set Tx = 0 indicating transmission has started
         state <= TX_STATE_DATA;
      end
   end
   TX_STATE_DATA: begin
      if (clken) begin
         if (bit_pos == 3'h7) //we keep assigning Tx with the data until all bits have been transmitted from 0 to 7
            state <= TX_STATE_STOP; // when bit position has finally reached 7, assign state to stop transmission
         else
            bit_pos <= bit_pos + 3'h1; //increment the bit position by 001
         Tx <= data[bit_pos]; //Set Tx to the data value of the bit position ranging from 0-7
      end
   end
   TX_STATE_STOP: begin
      if (clken) begin
         Tx <= 1'b1; //set Tx = 1 after transmission has ended
         state <= TX_STATE_IDLE; //Move to IDLE state once a transmission has been completed
      end
   end
   default: begin
      Tx <= 1'b1; // always begin with Tx = 1 and state assigned to IDLE
      state <= TX_STATE_IDLE;
   end
```

```
56      endcase
57    end
58
59    assign Tx_busy = (state != TX_STATE_IDLE); //We assign the BUSY signal when the transmitter is not idle
60
61    endmodule
62
```

2.2 Receiver

```
 1    module receiver  (input wire Rx,
 2                      output reg ready,       // default 1 bit reg
 3                      input wire ready_clr,
 4                      input wire clk_50m,
 5                      input wire clken,
 6                      output reg [7:0] data   // 8 bit register
 7                      );
 8    initial begin
 9       ready = 1'b0; // initialize ready = 0
10       data = 8'b0; // initialize data as 00000000
11    end
12    // Define the 4 states using 00,01,10 signals
13    parameter RX_STATE_START   = 2'b00;
14    parameter RX_STATE_DATA    = 2'b01;
15    parameter RX_STATE_STOP    = 2'b10;
16
17    reg [1:0] state = RX_STATE_START; // state is a 2-bit register/vector,initially equal to 00
18    reg [3:0] sample = 0; // This is a 4-bit register
19    reg [3:0] bit_pos = 0; // bit position is a 4-bit register/vector, initially equal to 000
20    reg [7:0] scratch = 8'b0; // An 8-bit register assigned to 00000000
21
22    always @(posedge clk_50m) begin
23       if (ready_clr)
24          ready <= 1'b0; // This resets ready to 0
25
26       if (clken) begin
27          case (state) // Let us consider the 3 states of the receiver
28          RX_STATE_START: begin // We define condtions for starting the receiver
29             if (!Rx || sample != 0) // start counting from the first low sample
30                sample <= sample + 4'b1; // increment by 0001
31             if (sample == 15) begin // once a full bit has been sampled
32                state <= RX_STATE_DATA; // start collecting data bits
33                bit_pos <= 0;
34                sample <= 0;
35                scratch <= 0;
36             end
37          end
38          RX_STATE_DATA: begin // We define conditions for starting the data colleting
39             sample <= sample + 4'b1; // increment by 0001
40             if (sample == 4'h8) begin // we keep assigning Rx data until all bits have 01 to 7
41                scratch[bit_pos[2:0]] <= Rx;
42                bit_pos <= bit_pos + 4'b1; // increment by 0001
43             end
44             if (bit_pos == 8 && sample == 15) // when a full bit has been sampled and
45                state <= RX_STATE_STOP; // bit position has finally reached 7, assign state to
    stop
46          end
```

```
47          RX_STATE_STOP: begin
48             /*
49              * Our baud clock may not be running at exactly the
50              * same rate as the transmitter.  If we thing that
51              * we're at least half way into the stop bit, allow
52              * transition into handling the next start bit.
53              */
54             if (sample == 15 || (sample >= 8 && !Rx)) begin
55                state <= RX_STATE_START;
56                data <= scratch;
57                ready <= 1'b1;
58                sample <= 0;
59             end
60             else begin
61                sample <= sample + 4'b1;
62             end
63          end
64          default: begin
65             state <= RX_STATE_START; // always begin with state assigned to START
66          end
67          endcase
68       end
69    end
70
71    endmodule
72
```

2.3 Baud Rate

```
 1   //This is a baud rate generator to divide a 50MHz clock into a 115200 baud Tx/Rx pair.
 2   //The Rx clock oversamples by 16x.
 3
 4   module baudrate  (input wire clk_50m,
 5                     output wire Rxclk_en,
 6                     output wire Txclk_en
 7                     );
 8   //Our Testbench uses a 50 MHz clock.
 9   //Want to interface to 115200 baud UART for Tx/Rx pair
10   //Hence, 50000000 / 115200 = 435 Clocks Per Bit.
11   parameter RX_ACC_MAX = 50000000 / (115200 * 16);
12   parameter TX_ACC_MAX = 50000000 / 115200;
13   parameter RX_ACC_WIDTH = $clog2(RX_ACC_MAX);
14   parameter TX_ACC_WIDTH = $clog2(TX_ACC_MAX);
15   reg [RX_ACC_WIDTH - 1:0] rx_acc = 0;
16   reg [TX_ACC_WIDTH - 1:0] tx_acc = 0;
17
18   assign Rxclk_en = (rx_acc == 5'd0);
19   assign Txclk_en = (tx_acc == 9'd0);
20
21   always @(posedge clk_50m) begin
22      if (rx_acc == RX_ACC_MAX[RX_ACC_WIDTH - 1:0])
23         rx_acc <= 0;
24      else
25         rx_acc <= rx_acc + 5'b1; //increment by 00001
26   end
27
28   always @(posedge clk_50m) begin
29      if (tx_acc == TX_ACC_MAX[TX_ACC_WIDTH - 1:0])
30         tx_acc <= 0;
31      else
32         tx_acc <= tx_acc + 9'b1; //increment by 000000001
33   end
34
35   endmodule
36
```

2.4 Top - Level Entity

```
 1   module uart(input wire [7:0] data_in, //input data
 2               input wire wr_en,
 3               input wire clear,
 4               input wire clk_50m,
 5               output wire Tx,
 6               output wire Tx_busy,
 7               input wire Rx,
 8               output wire ready,
 9               input wire ready_clr,
10               output wire [7:0] data_out,
11               output [7:0] LEDR,
12               output wire Tx2//output data
13               );
14   assign LEDR = data_in;
15   assign Tx2 = Tx;
16   wire Txclk_en, Rxclk_en;
17   baudrate uart_baud(  .clk_50m(clk_50m),
18                        .Rxclk_en(Rxclk_en),
19                        .Txclk_en(Txclk_en)
20                        );
21   transmitter uart_Tx( .data_in(data_in),
22                        .wr_en(wr_en),
23                        .clk_50m(clk_50m),
24                        .clken(Txclk_en), //We assign Tx clock to enable clock
25                        .Tx(Tx),
26                        .Tx_busy(Tx_busy)
27                        );
28   receiver uart_Rx( .Rx(Rx),
29                     .ready(ready),
30                     .ready_clr(ready_clr),
31                     .clk_50m(clk_50m),
32                     .clken(Rxclk_en), //We assign Tx clock to enable clock
33                     .data(data_out)
34                     );
35
36   endmodule
37
```

2.5 Test Bench

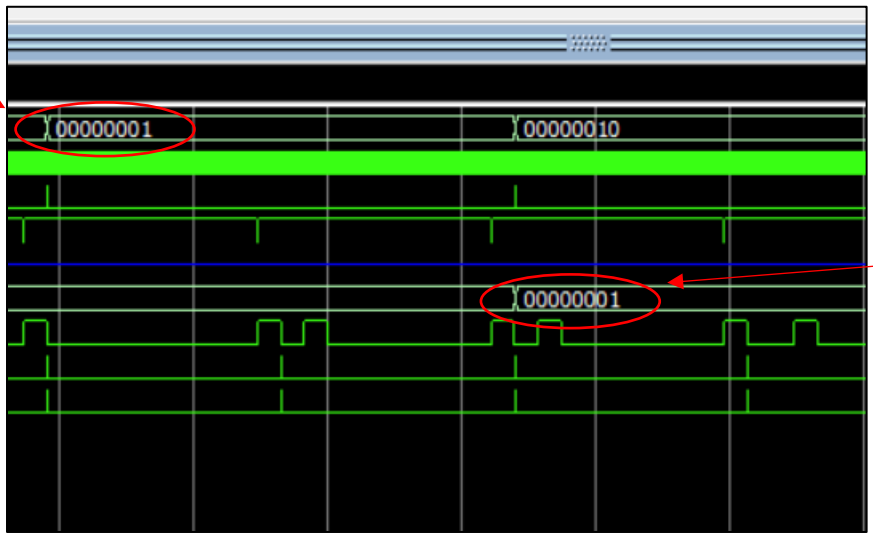
Date: May 08, 2024 uart_TB.v Project: uart_tx_rx

```
1   module uart_TB();
2
3   reg [7:0] data = 0;                 // Initialize data to 0
4   reg clk = 0;
5   reg enable = 0;
6
7   wire Tx_busy;
8   wire ready;
9   wire [7:0] Rx_data;
10
11  wire loopback;
12  reg ready_clr = 0;
13
14  // Instantiation of uart module
15  uart test_uart(.data_in(data),
16                 .wr_en(enable),
17                 .clk_50m(clk),
18                 .Tx(loopback),
19                 .Tx_busy(Tx_busy),
20                 .Rx(loopback),
21                 .ready(ready),
22                 .ready_clr(ready_clr),
23                 .data_out(Rx_data)
24                 );
25  initial begin
26     $dumpfile("uart.vcd");
27     $dumpvars(0, uart_TB);
28     enable <= 1'b1;
29     #2 enable <= 1'b0;
30  end
31
32
33  // Behavior upon reception of ready signal from UART module
34  always @(posedge ready) begin
35     #2 ready_clr <= 1;
36     #2 ready_clr <= 0;
37     if (Rx_data != data)             // Check if received data matches transmitted data
38     begin
39        $display("FAIL: rx data %x does not match tx %x", Rx_data, data);
40        $finish;
41     end
42     else begin
43        if (Rx_data == 8'h2) begin    // Check if received data is 11111111
44           $display("SUCCESS: all bytes verified");
45           $finish;
46        end
47        enable <= 1'b1;               // Enable UART module
48        data <= data + 1'b1;          // Increment transmitted data
49        #2 enable <= 1'b0;            // Disable UART module after 2 time units
50     end
51  end
52
53  // Clock generation
54  always begin
55     #1 clk = ~clk;
56  end
57  endmodule
58
```

3. Simulation Results

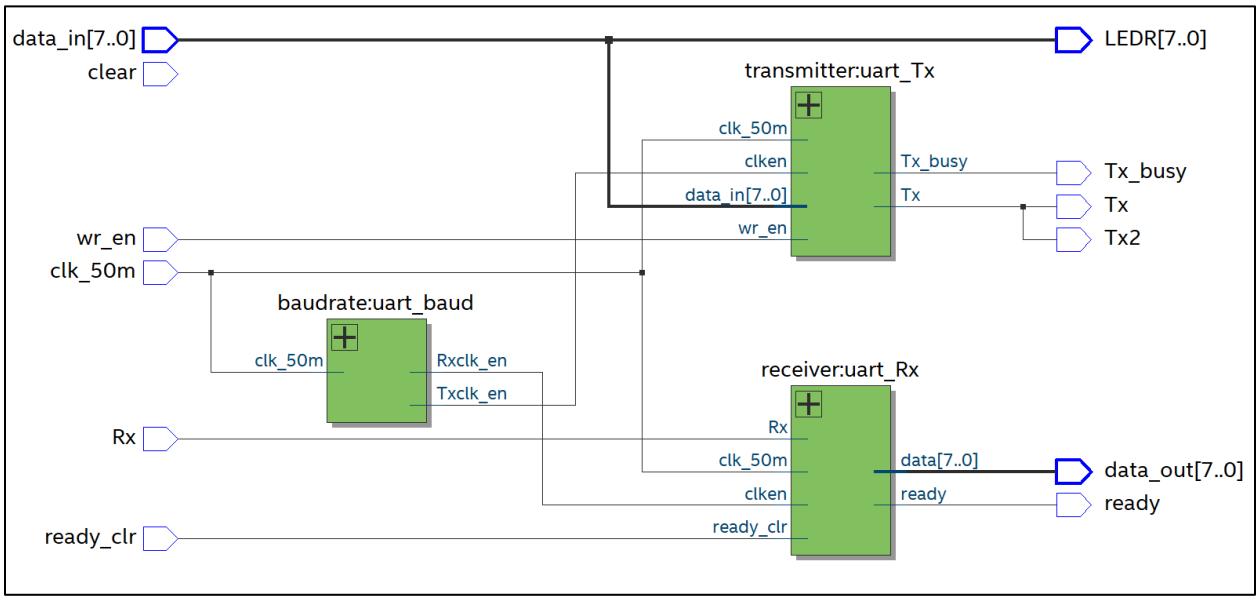
Testbench results

Transmitted bits

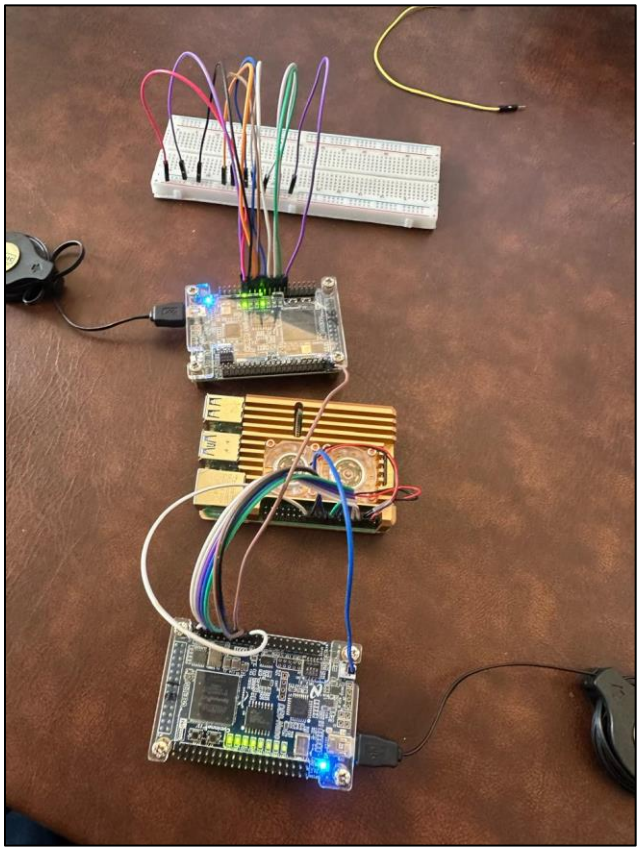


Received bits

RTL Viewer



FPGA Implementation



References

[1] M. Maged, “MuhammadMajiid/UART,” *GitHub*, Apr. 17, 2024. https://github.com/MuhammadMajiid/UART (accessed May 08, 2024).

[2]“UART & FPGA Bluetooth connection | Road to FPGAs #104,” www.youtube.com. https://youtu.be/QlscDcbKUV4?si=bgv5kwBHtRHo2gAz (accessed May 08, 2024).