University of Moratuwa

Department of Electronic & Telecommunication Engineering

UART implementation in FPGA

GAMIDU A.G.D. 200179H

GUNAWARDENA M.N. 200201V

HAPUTHANTHRI H.H.A.M. 200207U

RTL code for UART

```
module uart(                input wire [7:0] data_in,   //input data
                            input wire wr_en,
                            input wire clear,
                            input wire clk_50m,
                            output wire Tx,
                            output wire Tx_busy,
                            input wire Rx,
                            output wire ready,
                            input wire ready_clr,
                            output wire [7:0] data_out,
                            output [7:0] LEDR,
                            output wire Tx2//output data
                            );
assign LEDR = data_in;
assign Tx2 = Tx;
wire Txclk_en, Rxclk_en;
baudrate uart_baud(.clk_50m(clk_50m),
                   .Rxclk_en(Rxclk_en),
                   .Txclk_en(Txclk_en)
);

transmitter uart_Tx( .data_in(data_in),
                   .wr_en(wr_en),
                   .clk_50m(clk_50m),
                   .clken(Txclk_en), //We assign Tx clock to enable clock
                   .Tx(Tx),
                   .Tx_busy(Tx_busy)
                   );
```

```
receiver uart_Rx(     .Rx(Rx),
                      .ready(ready),
                      .ready_clr(ready_clr),
                      .clk_50m(clk_50m),
                      .clken(Rxclk_en), //We assign Tx clock to enable clock
                      .data(data_out)
                      );
endmodule

module baudrate  (input wire clk_50m,
                  output wire Rxclk_en,
                  output wire Txclk_en
                   );
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

module receiver  (input wire Rx,
                                        output reg ready,
                                        input wire ready_clr,
                                        input wire clk_50m,
                                        input wire clken,
                                        output reg [7:0] data
                                        );
initial begin
        ready = 1'b0;
        data = 8'b0;
```

```
end

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;
reg [3:0] sample = 0;
reg [3:0] bit_pos = 0;
reg [7:0] scratch = 8'b0;
always @(posedge clk_50m) begin
        if (ready_clr)
                ready <= 1'b0;

        if (clken) begin
                case (state)
                RX_STATE_START: begin
                        if (!Rx || sample != 0)
                                sample <= sample + 4'b1;
                        if (sample == 15) begin
                                state <= RX_STATE_DATA;
                                bit_pos <= 0;
                                sample <= 0;
                                scratch <= 0;
                        end
                end
                RX_STATE_DATA: begin
                        sample <= sample + 4'b1;
                        if (sample == 4'h8) begin
                                scratch[bit_pos[2:0]] <= Rx;
                                bit_pos <= bit_pos + 4'b1;
                        end
                        if (bit_pos == 8 && sample == 15)
                                state <= RX_STATE_STOP;
                end
                RX_STATE_STOP: begin
                        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
                end
```

```
                default: begin
                        state <= RX_STATE_START;
                end
                endcase
        end
end
endmodule

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 after transmission has started
                        state <= TX_STATE_DATA;
```

```
                end
        end
        TX_STATE_DATA: begin
                if (clken) begin
                        if (bit_pos == 3'h7)
                                state <= TX_STATE_STOP;
                        else
                                bit_pos <= bit_pos + 3'h1; //increment the bit position by 001
                        Tx <= data[bit_pos];
                end
        end
        TX_STATE_STOP: begin
                if (clken) begin
                        Tx <= 1'b1; //set Tx = 1 after transmission has ended
                        state <= TX_STATE_IDLE;
                end
        end
        default: begin
                Tx <= 1'b1; // always begin with Tx = 1 and state assigned to IDLE
                state <= TX_STATE_IDLE;
        end
        endcase
end

assign Tx_busy = (state != TX_STATE_IDLE);

endmodule
```

Testbench

```
module uart_TB();

reg [7:0] data = 2'b11;
reg clk = 0;
reg enable = 0;

wire Tx_busy;
wire rdy;
wire [7:0] Rx_data;

wire loopback;
reg ready_clr = 0;

uart test_uart(.data_in(data),
```

```
                                        .wr_en(enable),
                                        .clk_50m(clk),
                                        .Tx(loopback),
                                        .Tx_busy(Tx_busy),
                                        .Rx(loopback),
                                        .ready(ready),
                                        .ready_clr(ready_clr),
                                        .data_out(Rx_data)
                                        );
initial begin
        $dumpfile("uart.vcd");
        $dumpvars(0, uart_TB);
        enable <= 1'b1;
        #2 enable <= 1'b0;
end
always begin
        #1 clk = ~clk;
end
always @(posedge ready) begin
        #2 ready_clr <= 1;
        #2 ready_clr <= 0;
        if (Rx_data != data) begin
                $display("FAIL: rx data %x does not match tx %x", Rx_data, data);
                $finish;
        end
        else begin
                if (Rx_data == 8'h2) begin //Check if received data is 11111111
                        $display("SUCCESS: all bytes verified");
                        $finish;
                end
                data <= data + 1'b1;
                enable <= 1'b1;
                #2 enable <= 1'b0;
        end
end
endmodule
```

FPGA Implementation