**Sprawozdanie**

**MKWSC**

Lab2

**Serial Asynchronous Data Transmission**

Michał Kapuściński

Katowice 06.11.2024

# Design a synthesizable asynchronous receiver and transmitter for 8-bit characters and 1 stop-bit.

Tx, Rx receivers and clock generator were implemented:

Rx module:

`timescale 10ns / 100ps

module uart\_rx (

    input wire clk,

    input wire reset,

    input wire rx,

    output reg [7:0] rx\_data,

    output reg rx\_ready,

    output reg error

);

    reg [3:0] bit\_cnt;

    reg [9:0] shift\_reg;

    reg [3:0] state;

    localparam IDLE = 4'd0,

               START = 4'd1,

               DATA = 4'd2,

               STOP = 4'd3;

    initial begin

        state <= IDLE;

        rx\_ready <= 1'b0;

        bit\_cnt <= 4'd0;

        shift\_reg <= 10'd0;

        error <= 1'b0;

        rx\_data <= 8'd0;

    end

    always @(posedge clk or posedge reset) begin

        if (reset) begin

            state <= IDLE;

            rx\_ready <= 1'b0;

            bit\_cnt <= 4'd0;

            shift\_reg <= 10'd0;

            error <= 1'b0;

        end else begin

            case (state)

                IDLE: begin

                    rx\_ready <= 1'b0;

                    if (~rx) state <= START; // Start bit detected

                end

                START: begin

                    state <= DATA;

                    bit\_cnt <= 4'd0;

                end

                DATA: begin

                    shift\_reg <= {rx, shift\_reg[9:1]};

                    bit\_cnt <= bit\_cnt + 1;

                    if (bit\_cnt == 4'd7) state <= STOP;

                end

                STOP: begin

                    if (rx) begin

                        error <= 1'b0; // Stop bit detected

                        rx\_data <= shift\_reg[8:1]; // Extract data bits

                        rx\_ready <= 1'b1;

                    end

                    else begin

                        error <= 1'b1; // Error: Stop bit not detected

                    end

                    state <= IDLE;

                end

            endcase

        end

    end

endmodule

Tx module:

`timescale 10ns / 100ps

module uart\_tx (

    input wire clk,

    input wire reset,

    input wire tx\_start,

    input wire [7:0] tx\_data,

    output reg tx,

    output reg tx\_busy

);

    reg [3:0] bit\_cnt;

    reg [9:0] shift\_reg;

    reg [3:0] state;

    localparam IDLE = 4'd0,

               START = 4'd1,

               DATA = 4'd2,

               STOP = 4'd3;

    initial begin

        state <= IDLE;

        tx <= 1'b1;

        tx\_busy <= 1'b0;

        bit\_cnt <= 4'd0;

        shift\_reg <= 10'd0;

    end

    always @(posedge clk or posedge reset) begin

        if (reset) begin

            state <= IDLE;

            tx <= 1'b1;

            tx\_busy <= 1'b0;

            bit\_cnt <= 4'd0;

            shift\_reg <= 10'd0;

        end else begin

            case (state)

                IDLE: begin

                    tx <= 1'b1;

                    tx\_busy <= 1'b0;

                    if (tx\_start) begin

                        shift\_reg <= {1'b1, tx\_data, 1'b0}; // Stop bit, data, start bit

                        state <= START;

                        tx\_busy <= 1'b1;

                    end

                end

                START: begin

                    tx <= shift\_reg[0];

                    shift\_reg <= shift\_reg >> 1;

                    state <= DATA;

                    bit\_cnt <= 4'd0;

                end

                DATA: begin

                    tx <= shift\_reg[0];

                    shift\_reg <= shift\_reg >> 1;

                    bit\_cnt <= bit\_cnt + 1;

                    if (bit\_cnt == 4'd7) state <= STOP;

                end

                STOP: begin

                    tx <= 1'b1;

                    state <= IDLE;

                    tx\_busy <= 1'b0;

                end

            endcase

        end

    end

endmodule

Clock generator module:

`timescale 10ns / 100ps

module ClockDivider #(parameter DIVISOR = 250) // Default

(

    input wire clk\_50MHz,    // 50MHz input clock

    input wire reset,        // Reset signal

    output reg clk\_uart     // 9600 baud rate clock output

);

    // Calculate the number of clock cycles needed for the desired baud rate

    //localparam integer DIVISOR = 250;

    // Counter to keep track of clock cycles

    integer counter = 0;

    always @(posedge clk\_50MHz or posedge reset) begin

        if (reset) begin

            counter <= 0;

            clk\_uart <= 0;

        end else begin

            if (counter == (DIVISOR - 1)) begin

                counter <= 0;

                clk\_uart <= ~clk\_uart;  // Toggle the output clock

            end else begin

                counter <= counter + 1;

            end

        end

    end

    initial begin

        clk\_uart <= 1'b0;

    end

endmodule

Successfully transferred data (0xAE) from tx to rx module (simulation):

Obraz zawierający zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne, wyświetlacz

Opis wygenerowany automatycznie

## Check using simulation the maximal frequency error for the designed receiver that allows for the correct reception of characters.

Receiver allows correct data reception up to ~9% frequency variation. Tested with tx driven at 1MHz and rx driven at 910kHz.

### Implementation

After simulation phase the design was transferred to DE1-SOC board project