Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN2111 - Electronic Circuit Design

UART Assignment

PASIRA I.P.M.	210446J
PATHIRANA R.P.S.	210451U
PEIRIS D.L.C.J.	210454G

Date - 2024.05.07

1 UART main

```
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
```

2 Transmitter

```
module transmitter(
    input wire [7:0] data_in, // Input data as an 8-bit register/vector
    input wire wr_en,         // Enable wire to start transmission
    input wire clk_50m,       // 50 MHz clock signal
    input wire clken,         // Clock signal for the transmitter
```

```
    output reg Tx,              // Single bit register variable to hold transmitting bit
    output wire Tx_busy         // Transmitter busy signal
);

initial begin
    Tx = 1'b1; // Initialize Tx to 1 to begin 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;      // 8-bit register/vector for data, initially 00000000
reg [2:0] bit_pos = 3'h0;    // 3-bit register/vector for bit position, initially 000
reg [1:0] state = TX_STATE_IDLE; // 2-bit register/vector for state, initially 00

always @(posedge clk_50m) begin
    case (state)
        TX_STATE_IDLE: begin
            if (~wr_en) begin
                state <= TX_STATE_START; // Assign start signal to state
                data <= data_in;         // Assign input data vector to current data
                bit_pos <= 3'h0;         // Set bit position to zero
            end
        end
        TX_STATE_START: begin
            if (clken) begin
                Tx <= 1'b0;              // Set Tx = 0 indicating transmission has sta
                state <= TX_STATE_DATA;
            end
        end
        TX_STATE_DATA: begin
            if (clken) begin
                if (bit_pos == 3'h7)     // Keep assigning Tx with data until all bits
                    state <= TX_STATE_STOP; // When bit position reaches 7, assign sta
                else
                    bit_pos <= bit_pos + 3'h1; // Increment bit position by 001
                Tx <= data[bit_pos];       // Set Tx to data value of the current bit
            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 transmission comp
            end
        end
        default: begin
            Tx <= 1'b1;  // Begin with Tx = 1 and state assigned to IDLE
            state <= TX_STATE_IDLE;
        end
    endcase
end
```

```
assign Tx_busy = (state != TX_STATE_IDLE); // Assign BUSY signal when transmitter is n

endmodule
```

3 Receiver

```
module receiver (
    input wire Rx,
    output reg ready,        // Default 1-bit register
    input wire ready_clr,
    input wire clk_50m,
    input wire clken,
    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 3 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;  // 2-bit register/vector for state, initially 00
reg [3:0] sample = 0;               // 4-bit register for sample
reg [3:0] bit_pos = 0;              // 4-bit register/vector for bit position, initial
reg [7:0] scratch = 8'b0;           // 8-bit register assigned to 00000000

always @(posedge clk_50m) begin
    if (ready_clr)
        ready <= 1'b0;  // Reset ready to 0

    if (clken) begin
        case (state) // Consider the 3 states of the receiver
            RX_STATE_START: begin // Define conditions 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 // Define conditions for starting data collecting
                sample <= sample + 4'b1; // Increment by 0001
                if (sample == 4'h8) begin // Keep assigning Rx data until all bits hav
                    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
                    state <= RX_STATE_STOP; // 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 think that
                 * we're at least halfway 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
            end
            default: begin
                state <= RX_STATE_START; // Always begin with state assigned to START
            end
        endcase
    end
end
endmodule
```

endmodule

4 Baudrate

```
// 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
```

5 Test Bench

```
// This is a simple testbench for UART Tx and Rx.
// The Tx and Rx pins have been connected together creating a serial loopback.
// We check if we receive what we have transmitted by sending incrementing data bytes.

module uart_TB();

reg [7:0] data = 0;
reg clk = 0;
reg enable = 0;

wire Tx_busy;
wire ready;
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
```

1 Timing Diagram

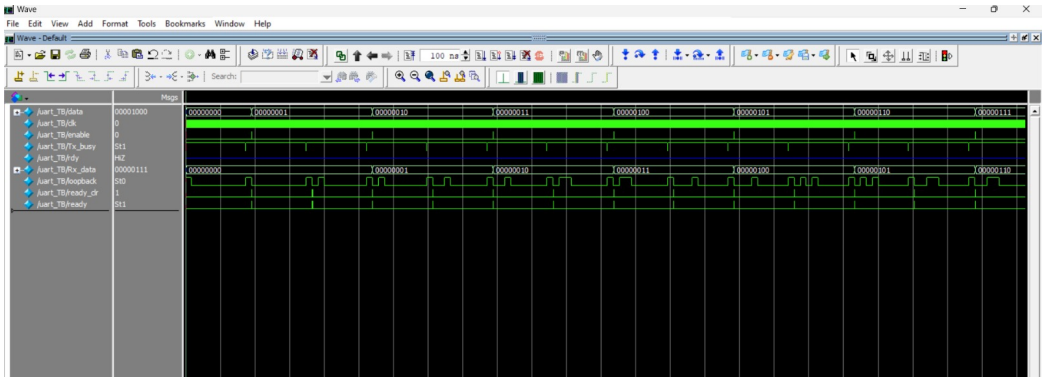

Figure 1: Timing Diagram

2 RTL Diagram

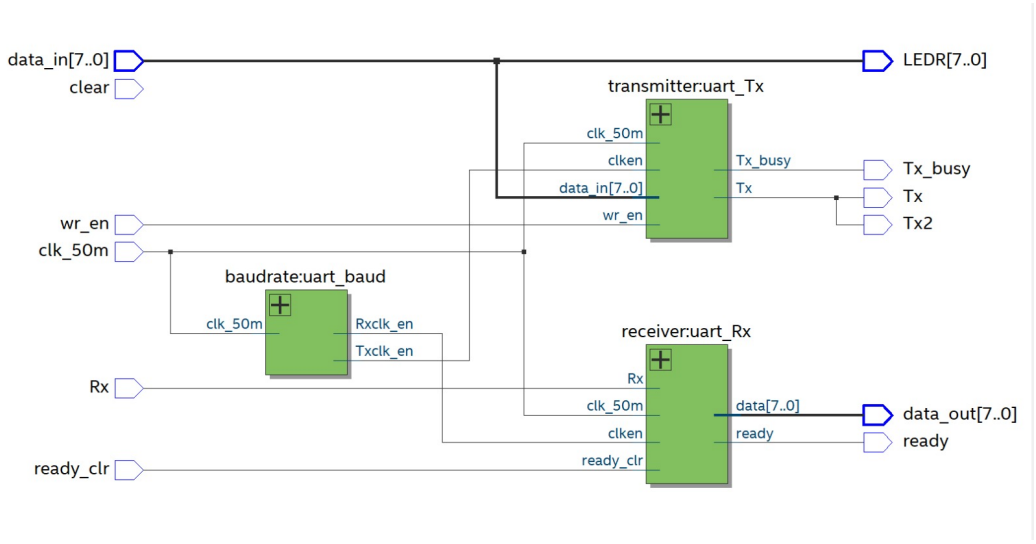

Figure 2: RTL Diagram