Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

UART Assignment - Group 10

A.A.W.L.R.Amarasinghe	210031H
Y.W.S.P.Amarasinghe	210035A
D.N.Amarathunga	210037G

Submitted in partial fulfillment of the requirements for the module
EN2111 - Electronic Circuit Design

7th of May 2024

Contents

1 Introduction

The DE0-nano FPGA board can be used to implement a UART transceiver. UART is a communication protocol for serial data exchange between two devices. In UART, the transmitter converts parallel data to serial form and transmits it, while the receiver converts the received serial data back to parallel form.

UART communication is asynchronous, meaning there is no clock signal for synchronization. Start and stop bits are used to mark the beginning and end of data packets. The receiver reads incoming bits at a specific baud rate, which determines the data transfer speed. Matching baud rates between the transmitter and receiver is crucial for successful data transfer.

To implement UART on the DE0-nano FPGA board, you need to design the UART module in Verilog, write a testbench for simulationDE0-nano board to the corresponding pins of the device you want to communicate with.

2 Verilog RTL Code

2.1 Transmitter

Date: May 06, 2024 transmitter.v Project: uart_tx_rx

```
1  module transmitter(  input wire [7:0] data_in, //input data as an 8-bit regsiter/vector
2                       input wire wr_en, //e nable 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 00
20 reg [1:0] state = TX_STATE_IDLE; //state is a 2 bit register/vector,initially equal to 00
21
22 always @(posedge clk_50m) begin
23    case (state) //Let us consider the 4 states of the transmitter
24    TX_STATE_IDLE: begin //We define the conditions for idle  or NOT-BUSY state
25       if (~wr_en) begin
26          state <= TX_STATE_START; //assign the start signal to state
27          data <= data_in; //we assign input data vector to the current data
28          bit_pos <= 3'h0; //we assign the bit position to zero
29       end
30    end
31    TX_STATE_START: begin //We define the conditions for the transmission start state
32       if (clken) begin
33          Tx <= 1'b0; //set Tx = 0 indicating transmission has started
34          state <= TX_STATE_DATA;
35       end
36    end
37    TX_STATE_DATA: begin
38       if (clken) begin
39          if (bit_pos == 3'h7) //we keep assigning Tx with the data until all bits have
   been transmitted from 0 to 7
40             state <= TX_STATE_STOP; // when bit position has finally reached 7, assign
   state to stop transmission
41          else
42             bit_pos <= bit_pos + 3'h1; //increment the bit position by 001
43          Tx <= data[bit_pos]; //Set Tx to the data value of the bit position ranging from
   0-7
44       end
45    end
46    TX_STATE_STOP: begin
47       if (clken) begin
48          Tx <= 1'b1; //set Tx = 1 after transmission has ended
49          state <= TX_STATE_IDLE; //Move to IDLE state once a transmission has been
   completed
50       end
51    end
52    default: begin
53       Tx <= 1'b1; // always begin with Tx = 1 and state assigned to IDLE
54       state <= TX_STATE_IDLE;
55    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
```

Figure 1: Transmitter Verilog code

2.2 Receiver

Date: May 06, 2024 receiver.v Project: uart_tx_rx

```
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 0
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
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
```

Figure 2: Receiver Verilog code

2.3 Testbench

Date: May 06, 2024 uart_TB.v Project: uart_tx_rx

```
1  //This is a simple testbench for UART Tx and Rx.
2  //The Tx and Rx pins have been connected together creating a serial loopback.
3  //We check if we receive what we have transmitted by sending incremeting data bytes.
4
5  //It sends out byte 0xAB over the transmitter
6  //It then exercises the receive by receiving byte 0x3F
7  // include "uart.v"
8
9  module uart_TB();
10
11 reg [7:0] data = 0;
12 reg clk = 0;
13 reg enable = 0;
14
15 wire Tx_busy;
16 wire rdy;
17 wire [7:0] Rx_data;
18
19 wire loopback;
20 reg ready_clr = 0;
21
22 uart test_uart(.data_in(data),
23                .wr_en(enable),
24                .clk_50m(clk),
25                .Tx(loopback),
26                .Tx_busy(Tx_busy),
27                .Rx(loopback),
28                .ready(ready),
29                .ready_clr(ready_clr),
30                .data_out(Rx_data)
31                );
32 initial begin
33    $dumpfile("uart.vcd");
34    $dumpvars(0, uart_TB);
35    enable <= 1'b1;
36    #2 enable <= 1'b0;
37 end
38 always begin
39    #1 clk = ~clk;
40 end
41 always @(posedge ready) begin
42    #2 ready_clr <= 1;
43    #2 ready_clr <= 0;
44    if (Rx_data != data) begin
45       $display("FAIL: rx data %x does not match tx %x", Rx_data, data);
46       $finish;
47    end
48    else begin
49       if (Rx_data == 8'h2) begin //Check if received data is 11111111
50          $display("SUCCESS: all bytes verified");
51          $finish;
52       end
53       data <= data + 1'b1;
54       enable <= 1'b1;
55       #2 enable <= 1'b0;
56    end
57 end
58 endmodule
59
```

Figure 3: Testbench Verilog code

2.4 Baudrate generator

Date: May 06, 2024 baudrate.v Project: uart_tx_rx

```
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
```

Figure 4: Baudrate generator Verilog code

2.5 Top level module

Date: May 06, 2024 uart.v Project: uart_tx_rx

```
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
```

Figure 5: Top Level module Verilog code

3 Simulation Results

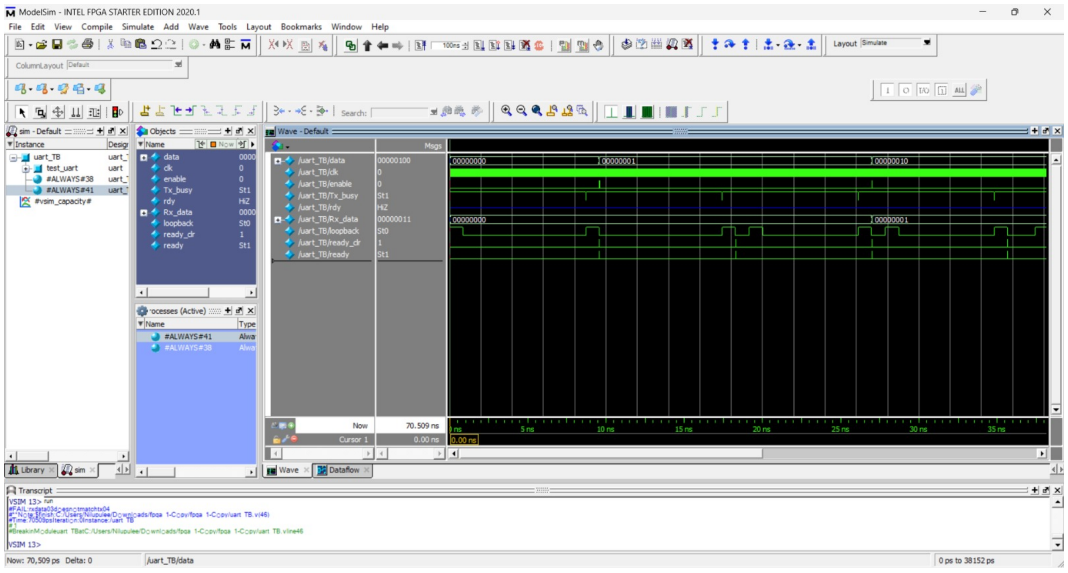

Figure 6: Simulation Results

4 FPGA Implementation

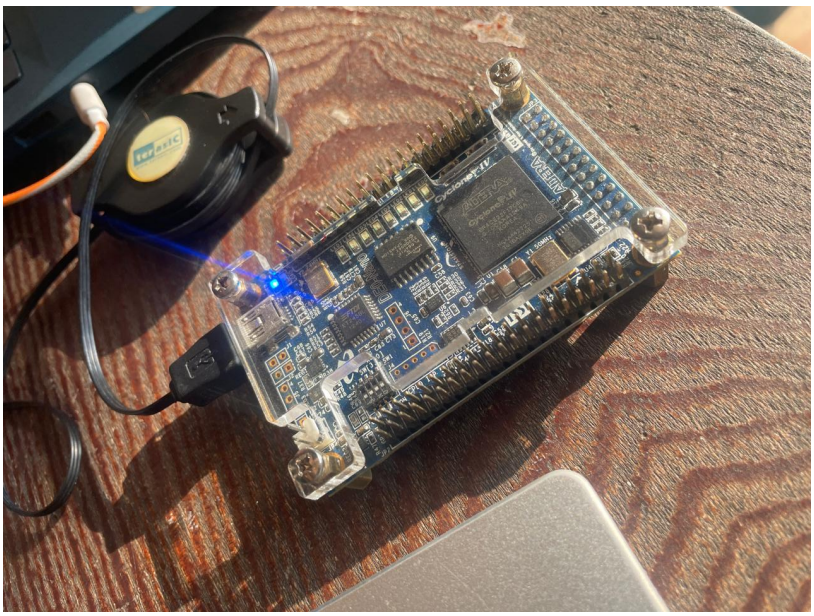

Figure 7: FPGA Implementation

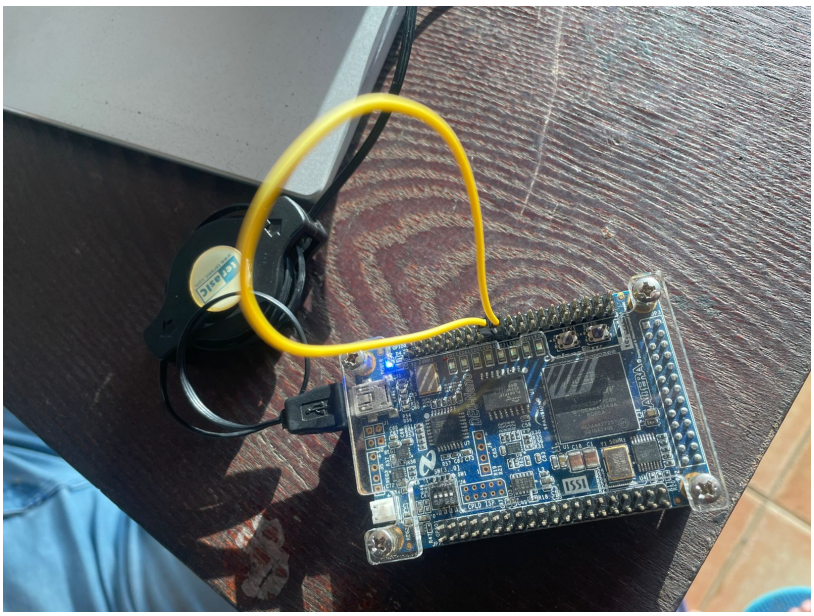

Figure 8: FPGA Implementation