Digital Logic Laboratory

Report On LAB #4 - UART

Ana Maghradze

30th November 2021

Contents

1 Description of The Assignment

For this assignment, we should implement UART protocol. The requirements are the following:

When `KEY[0]` is pressed first time, we should write 4-bit value of `SWITCH` into `DATA[3:0]` and when `KEY[0]` is pressed second time, we should write 4-bit value of `SWITCH` into `DATA[7:4]`.

After this, when `KEY[1]` is pressed, the data is sent to transmitter (TX, connected to `GPIO_0[0] pin`) with the baud rate of 1000. Then transmission package is filled with START_BIT, ODD_PARITY_BIT, STOP_BIT and DATA received from switches and then this package should be sent to Receiver (RX, connected to `GPIO_0[1] pin`) and if parity is correct, received data should appear on `LED`s, otherwise `LED`s should display `8'b11111111`. TX and RX should work in full-duplex mode.

2 Description of The Solution

In my code, I have three modules, one for transmitter, another for receiver, and third one as top module where I have instances of transmitter and receiver, provide inputs for them and connect output to `LED`s.

In top module, I have signal `START_TX` that becomes 1 when `KEY[1]` is pressed. This signal tells transmitter to start transmission package creation. Also, I indicate baud rate in this module and then use it as input for transmitter and receiver. This baud rate is then used to slow down or speed up the clock whose default frequency is 50 million Hz, so I define it in Tx and Rx.

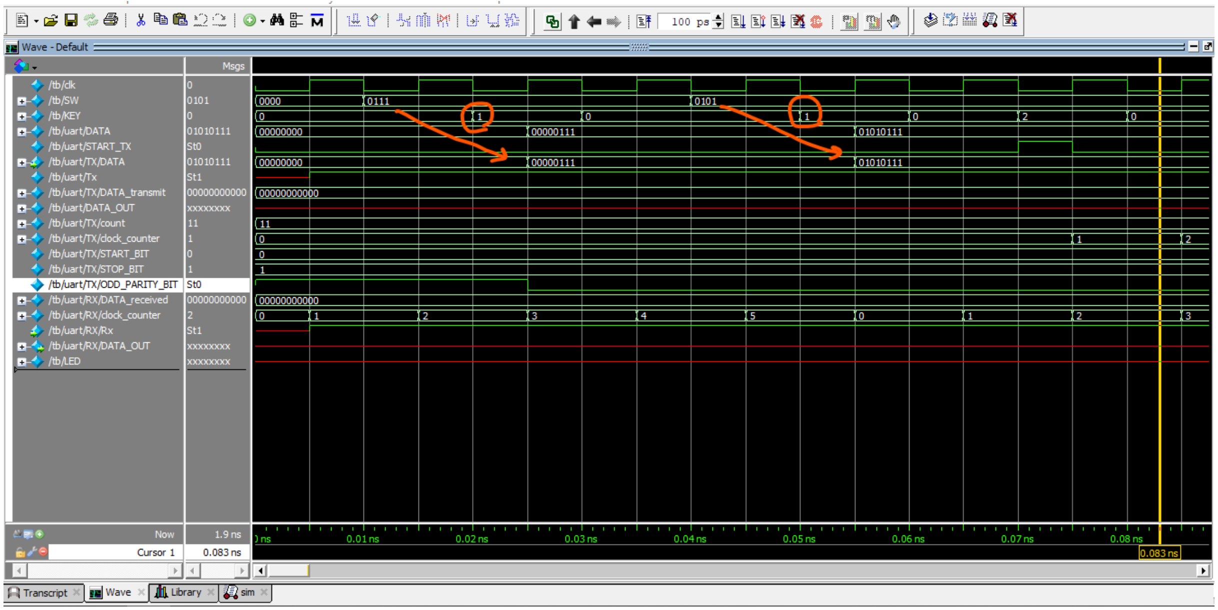
As it was required, I use clock enable, in other words, I use the same but slower clock instead of another clock. This is achieved by enabling clock after being disabled for the indicated period.

In transmitter, when `START_TX` signal is 1 or transmitting process is happening (or has just started), firstly I write `START_BIT, DATA, ODD_PARITY_BIT, and STOP_BIT` into 11-bit `DATA_transmit` and set counter to 10. Then I use this counter to read bits from `DATA_transmit` and send them from Tx (`GPIO_0[0]`) to Rx (`GPIO_0[1]`) one by one. If not transmitting data, Tx always sends 1 to Rx.

In receiver, clock is configured the same way as in transmitter. If the first bit sent from transmitter that is `START_BIT` is 0, Rx begins to receive data bits and store them into `DATA_received`. When `STOP_BIT` is received, `PARITY` is checked and if it's correct, data bits are sent to the output and displayed on `LED`s. Otherwise, `8'b11111111` is displayed.

3 ModelSim Simulation

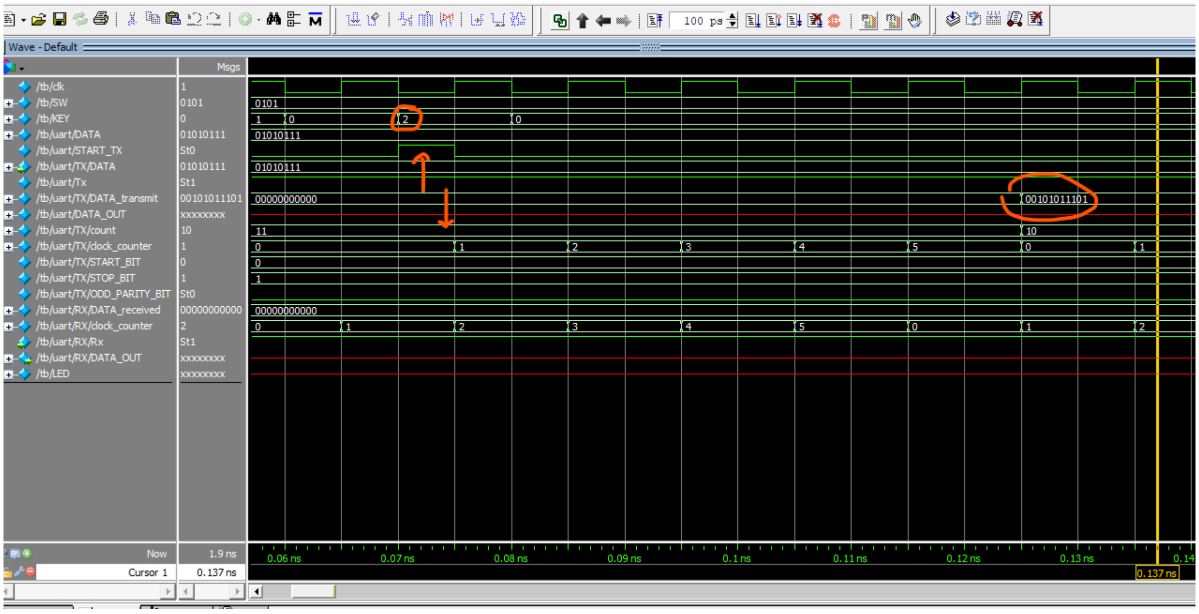
3.1 Writing SWITCH values into DATA



Wave - Default

	Msgs
/tb/clk	0
/tb/SW	0101
/tb/KEY	0
/tb/uart/DATA	01010111
/tb/uart/START_TX	St0
/tb/uart/TX/DATA	01010111
/tb/uart/Tx	St1
/tb/uart/TX/DATA_transmit	00000000000
/tb/uart/DATA_OUT	xxxxxxxx
/tb/uart/TX/count	11
/tb/uart/TX/clock_counter	1
/tb/uart/TX/START_BIT	0
/tb/uart/TX/STOP_BIT	1
/tb/uart/TX/ODD_PARITY_BIT	St0
/tb/uart/RX/DATA_received	00000000000
/tb/uart/RX/clock_counter	2
/tb/uart/RX/Rx	St1
/tb/uart/RX/DATA_OUT	xxxxxxxx
/tb/LED	xxxxxxxx
Now	1.9 ns
Cursor 1	0.083 ns

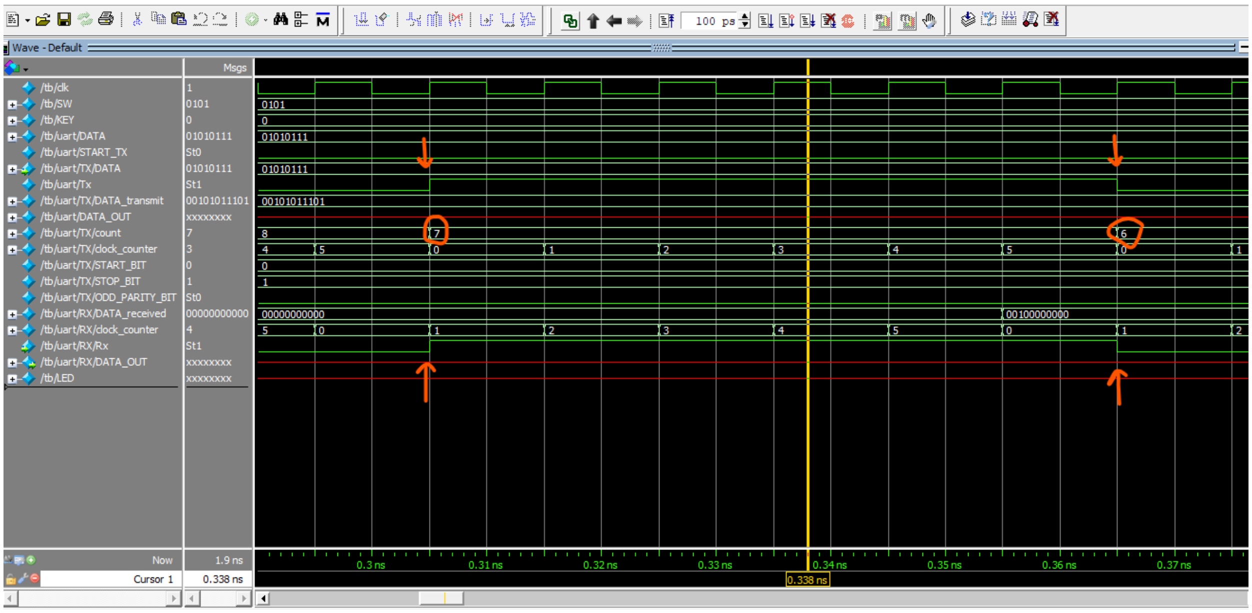
3.2 Creating a transmission package



Wave - Default

	Msgs
/tb/clk	1
/tb/SW	0101
/tb/KEY	0
/tb/uart/DATA	01010111
/tb/uart/START_TX	St0
/tb/uart/TX/DATA	01010111
/tb/uart/Tx	St1
/tb/uart/TX/DATA_transmit	00101011101
/tb/uart/DATA_OUT	xxxxxxxx
/tb/uart/TX/count	10
/tb/uart/TX/clock_counter	1
/tb/uart/TX/START_BIT	0
/tb/uart/TX/STOP_BIT	1
/tb/uart/TX/ODD_PARITY_BIT	St0
/tb/uart/RX/DATA_received	00000000000
/tb/uart/RX/clock_counter	2
/tb/uart/RX/Rx	St1
/tb/uart/RX/DATA_OUT	xxxxxxxx
/tb/LED	xxxxxxxx
Now	1.9 ns
Cursor 1	0.137 ns

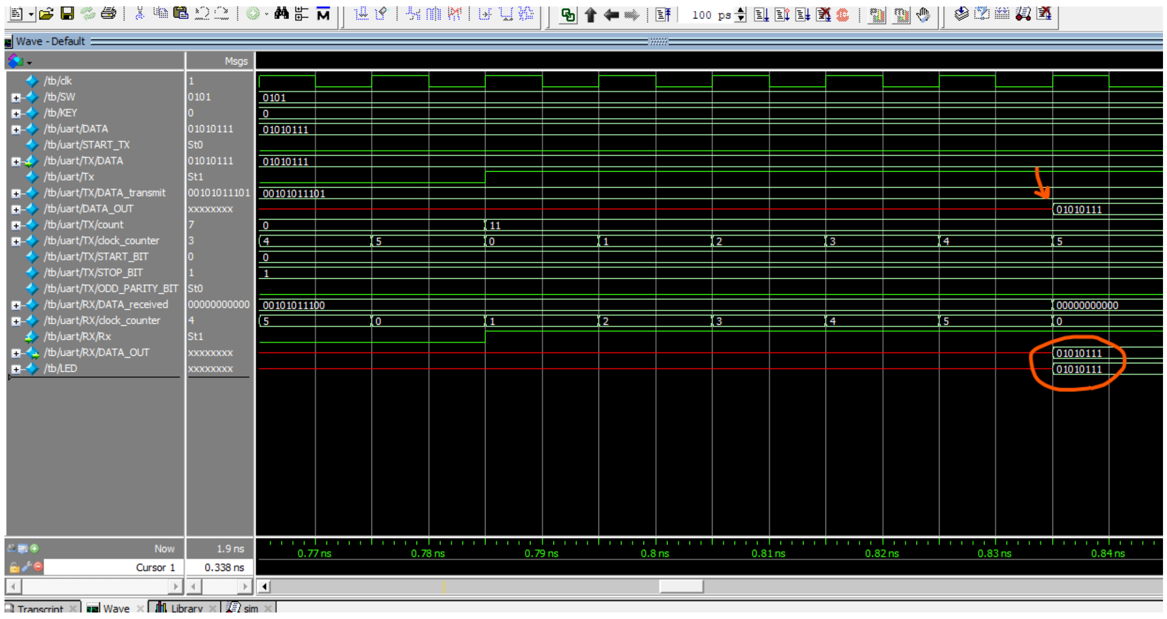
3.3 Bit transmission process from Tx to Rx



Wave - Default

	Msgs
/tb/clk	1
/tb/SW	0101
/tb/KEY	0
/tb/uart/DATA	01010111
/tb/uart/START_TX	St0
/tb/uart/TX/DATA	01010111
/tb/uart/Tx	St1
/tb/uart/TX/DATA_transmit	00101011101
/tb/uart/DATA_OUT	xxxxxxxx
/tb/uart/TX/count	7
/tb/uart/TX/clock_counter	3
/tb/uart/TX/START_BIT	0
/tb/uart/TX/STOP_BIT	1
/tb/uart/TX/ODD_PARITY_BIT	St0
/tb/uart/RX/DATA_received	00000000000
/tb/uart/RX/clock_counter	4
/tb/uart/RX/Rx	St1
/tb/uart/RX/DATA_OUT	xxxxxxxx
/tb/LED	xxxxxxxx
Now	1.9 ns
Cursor 1	0.338 ns

3.4 Received data on LEDs



```
Wave - Default
                                Msgs
/tb/clk                         1
/tb/SW                          0101
/tb/KEY                         0
/tb/uart/DATA                   01010111
/tb/uart/START_TX               St0
/tb/uart/TX/DATA                01010111
/tb/uart/Tx                     St1
/tb/uart/TX/DATA_transmit       00101011101
/tb/uart/DATA_OUT               xxxxxxxx
/tb/uart/TX/count               7
/tb/uart/TX/clock_counter       3
/tb/uart/TX/START_BIT           0
/tb/uart/TX/STOP_BIT            1
/tb/uart/TX/ODD_PARITY_BIT      St0
/tb/uart/RX/DATA_received       00000000000
/tb/uart/RX/clock_counter       4
/tb/uart/RX/Rx                  St1
/tb/uart/RX/DATA_OUT            xxxxxxxx
/tb/LED                         xxxxxxxx

Now        1.9 ns
Cursor 1   0.338 ns
```

4 Appendix

4.1 Transmitter

```
module transmitter(
  input clk,
  input [19:0] baudRate,
  input [7:0] DATA,
  input START_TX,
//  output reg Tx // for simulation
  inout reg [35:0] GPIO_0
 );

localparam CLK_HZ = 50000000;
integer clock_counter = 0;

localparam START_BIT = 0;
localparam STOP_BIT = 1;

reg [10:0] DATA_transmit = 0;
reg [3:0] count = 11;
reg transmitting = 0;
wire ODD_PARITY_BIT;

assign ODD_PARITY_BIT = ~(^DATA);

always @(posedge clk)
begin
  if(START_TX == 1 || transmitting == 1)
  begin
    transmitting <= 1;
    if(clock_counter == (CLK_HZ / baudRate))
    begin
      if(count == 11)
      begin
        DATA_transmit[10] <= START_BIT;
        DATA_transmit[9:2] <= DATA;
        DATA_transmit[1] <= ODD_PARITY_BIT;
        DATA_transmit[0] <= STOP_BIT;
        count <= 10;
      end
      else
      begin
//          Tx <= DATA_transmit[count];
        GPIO_0[0] <= DATA_transmit[count];
        count <= count == 0 ? 11 : count - 1;
        if(count == 0) transmitting <= 0;
      end
      clock_counter <= 0;
    end
    else clock_counter <= clock_counter + 1;
  end
  else
  begin
    if(count == 11)
    begin
      DATA_transmit <= 0;
      GPIO_0[0] <= 1;
//        Tx <= 1;
    end
  end
end

endmodule
```

4.2 Receiver

```
module receiver(
  input clk,
  input [19:0] baudRate,
//  input Rx, // for simulation
  output reg [7:0] DATA_OUT,
  inout reg [35:0] GPIO_0
);

reg [3:0] count = 11;
reg [10:0] DATA_received = 0;

localparam CLK_HZ = 50000000;
integer clock_counter = 0;

always @(posedge clk)
begin
  if(clock_counter == (CLK_HZ / baudRate))
  begin
    if(count == 0)
    begin
      if(~(^DATA_received[9:2]) == DATA_received[1])
      begin
        DATA_OUT <= DATA_received[9:2];
      end
      else DATA_OUT <= 8'b11111111;
      count <= 11;
    end
    else
    begin
      if(count == 11 && GPIO_0[1] == 0) // Rx insted of GPIO_0[1] for simulation
      begin
        DATA_received[10] <= GPIO_0[1];
//        DATA_received[10] <= Rx;
        count <= 9;
      end
      else if(count < 11)
      begin
        DATA_received[count] <= GPIO_0[1];
//        DATA_received[count] <= Rx;
        count <= count - 1;
      end
    end

    clock_counter <= 0;
  end
  else clock_counter <= clock_counter + 1;
end

endmodule
```

4.3 Top module

```
module de0_nano_soc_baseline(
  input CLOCK_50,
  input [3:0] SW,
  input [1:0] KEY,
  inout [35:0] GPIO_0,
  output [7:0] LED
);

reg [1:0] prevKey = 3;
reg [2:0] state = 0;
reg [7:0] DATA = 0;
reg [19:0] baudRate = 1000;

wire START_TX;
wire [7:0] DATA_OUT;

//// instcances for simulation
// transmitter TX(CLOCK_50, baudRate, DATA, START_TX, Tx);
// receiver RX(CLOCK_50, baudRate, Tx, DATA_OUT);

// instcances for board
transmitter TX(CLOCK_50, baudRate, DATA, START_TX, GPIO_0);
receiver RX(CLOCK_50, baudRate, DATA_OUT, GPIO_0);

assign LED = DATA_OUT;
assign START_TX = (KEY[1] == 1 && prevKey[1] == 0);

always @(posedge CLOCK_50)
begin
  if(KEY[0] == 1 && prevKey[0] == 0)
  begin
    case(state)
      0: begin
          DATA[3:0] <= SW;
          state <= 1;
        end
      1: begin
          DATA[7:4] <= SW;
          state <= 0;
        end
    endcase
  end

  prevKey <= KEY;
end

endmodule
```

4.4 Testbench

```
module tb;

reg clk = 0;
reg [1:0] KEY = 0;
reg [3:0] SW = 0;
wire [7:0] LED;

always #5 clk <= ~clk;

de0_nano_soc_baseline uart(clk, SW, KEY, LED);

initial
begin
  #10 SW = 7;
  #10 KEY = 1;
  #10 KEY = 0;

  #10 SW = 5;
  #10 KEY = 1;
  #10 KEY = 0;

  // output must be 01010111

  #10 KEY = 2;
  #10 KEY = 0;

  #1000 KEY = 1;
  #10 KEY = 0;
end


endmodule
```