Университет ИТМО

Кафедра вычислительной техники

Отчёт по лабораторной работе № 2
по дисциплине: "Схемотехника ЭВМ"
Вариант №5

Студенты: Куклина М.Д.
Кириллова А.А.

Преподаватель:

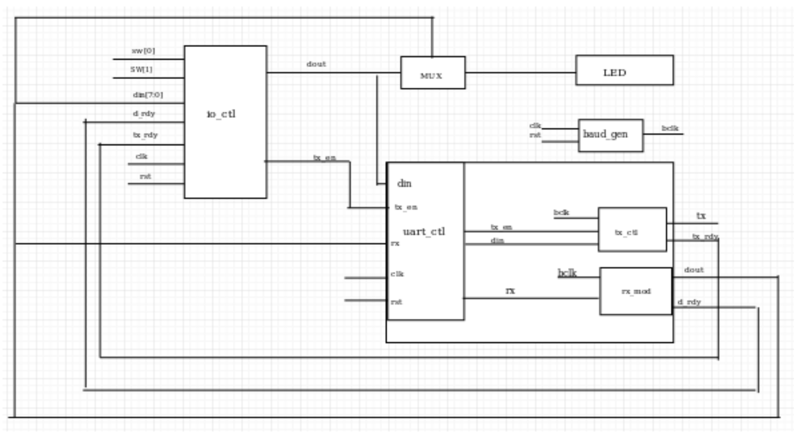
Санкт-Петербург
2016 г.

Содержание

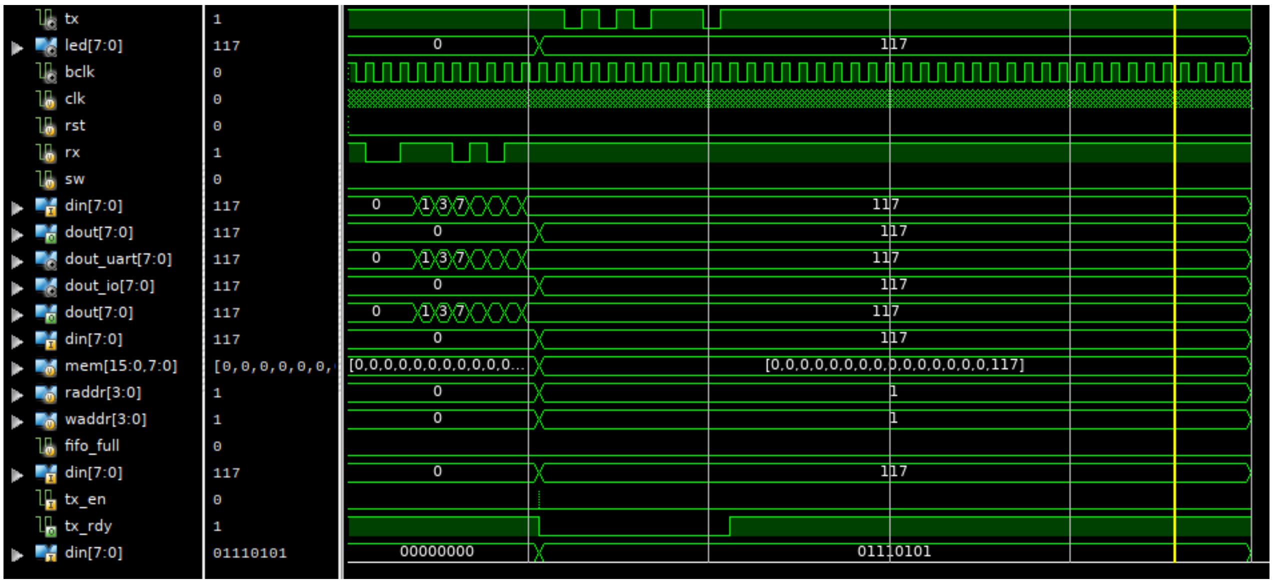
Цели работы

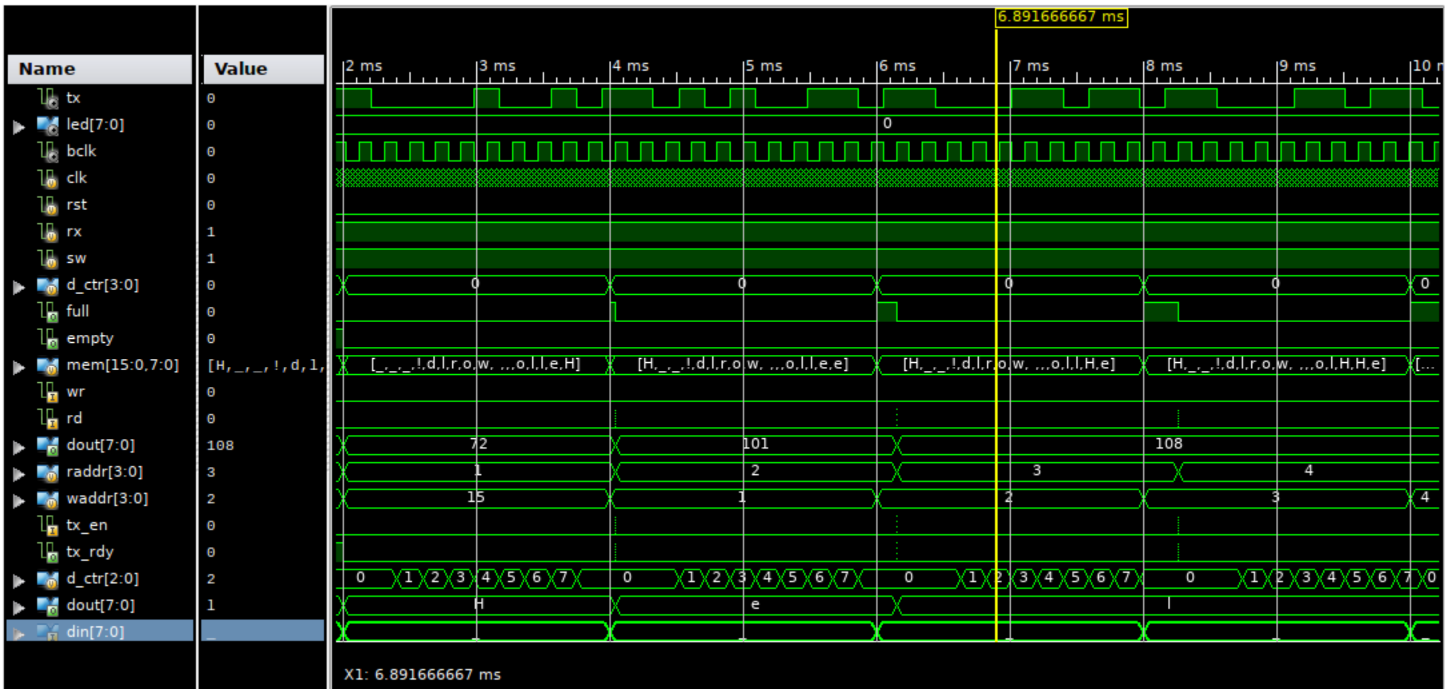
1. Знакомство с принципами работы последовательных интерфейсов ввода/вывода: I2C, SPI, UART.
2. Изучение основ разработки аппаратных контроллеров периферийных устройств.
3. Изучение основ работы с цифровыми датчиками.

RTL модель



Временные диаграммы


```
tx              1
led[7:0]        117
bclk            0
clk             0
rst             0
rx              1
sw              0
din[7:0]        117
dout[7:0]       117
dout_uart[7:0]  117
dout_io[7:0]    117
dout[7:0]       117
din[7:0]        117
mem[15:0,7:0]   [0,0,0,0,0,0,
raddr[3:0]      1
waddr[3:0]      1
fifo_full       0
din[7:0]        117
tx_en           0
tx_rdy          1
din[7:0]        01110101
```



Листинг

```
/* Top module. */
module ctl(
        input               clk,
        input               rst,

        /* UART interface. */
        input               rx,
//      input               cts,

        output              tx,
//      output              rts,

        /* Nexys interface.*/
        input                    sw,
        output   reg[7:0]        led
);

        wire  [7:0]  dout_uart;
        wire  [7:0]  dout_io;
        wire  tx_en;
        wire  tx_rdy;
        wire  d_rdy;

        io_ctl io_ctl(
                .clk(clk),
                .rst(rst),

                .sw(sw),

                .din(dout_uart),
                .d_rdy(d_rdy),
                .tx_rdy(tx_rdy),

                .dout(dout_io),
                .tx_en(tx_en) // dout_io is ready.
        );

        uart_ctl uart_ctl(

                .clk(clk),
                .rst(rst),

                .rx   (rx),
                .din(dout_io),
                .tx_en(tx_en),

                .tx (tx),
                .dout(dout_uart),
                .d_rdy(d_rdy),
                .tx_rdy(tx_rdy)
        );

         always @( negedge clk )
             if( rst )
                 led <= 0;
             else if( d_rdy )
               led <= dout_uart;


endmodule
```

```
module io_ctl(
    input                clk,
    input                rst,
    /* SW = 0: echo-mode. SW = 1: send 'Hello world!\r\n'. */
    input                sw,
    /* Data from uart.*/
    input   [7:0]        din,
    input                d_rdy,
    input                tx_rdy,
    /* Data to uart. */
    output  reg[7:0]     dout,
    output  reg         tx_en
);
    localparam TIME      = 100000;
    localparam ECHO_MODE = 0;
    localparam SEND_MODE = 1;

    reg [7:0]  data [14:0];
    reg [3:0]  d_ctr    = 0;
    reg [26:0] tm_ctr   = 0;
    reg was_d_rdy       = 0;

    wire tx_flag;

    assign tx_flag = (tm_ctr == TIME) ? 1 : ((( d_ctr == 15 ) || ( d_ctr == 0 )) ? 0 : 1 );

    always @( posedge rst ) begin
            data[0]  = "H"; data[1]  = "e";    data[2]  = "l";
            data[3]  = "l"; data[4]  = "o";    data[5]  = ",";
            data[6]  = " "; data[7]  = "w";    data[8]  = "o";
            data[9]  = "r"; data[10] = "l";    data[11] = "d";
            data[12] = "!"; data[13] = 8'h0D; data[14] = 8'h0A;
    end

    always @( negedge clk or posedge rst )
        if( rst ) begin
             tx_en   <= 0;
             dout    <= 0;
             d_ctr   <= 0;
             was_d_rdy <= 0;
        end else
             case( sw )
                 ECHO_MODE:
                     if( d_rdy && !was_d_rdy ) begin
                        dout      <= din;
                        tx_en     <= 1;
                        was_d_rdy <= d_rdy;
                     end
                     else begin
                        tx_en     <= 0;
                        was_d_rdy <= d_rdy;
                    end
                 SEND_MODE:
                    begin
                        if( tx_flag ) begin
                            tx_en <= 1;
                            dout  <= data[d_ctr];
                            d_ctr <= d_ctr + 1;
                        end
                        else if( d_ctr == 15 ) begin
                            tx_en    <= 0;
                            d_ctr <= 0;
                        end
                        end
                endcase

    always @( posedge clk or posedge rst ) begin
            if( rst )
                tm_ctr <= 0;
            else if( sw == SEND_MODE ) begin
                if( tm_ctr == TIME )
                    tm_ctr  <= 0;
                else
                    tm_ctr  <= tm_ctr + 1;
            end
    end

endmodule
```

```
module uart_ctl(
    input               clk,
    input               rst,

    input               rx,
//  input               rts,

    // Data to send.
    input       [7:0]   din,
    // Enable transmission ( data ready ).
    input               tx_en,

    // Transmission wire.
    output              tx,
//  output              cts,

    // Received data.
    output      [7:0]   dout,
    // Receive ends; data ready.
    output              d_rdy,
    // Controller ready for transmission.
    output              tx_rdy
);

    wire        bclk;

    baud_gen baud_gen(
        .clk(clk),
        .rst(rst),

        .bclk(bclk)
    );

    rx_mod rx_mod(
        .clk  (clk),
        .rst  (rst),
        .bclk(bclk),
        .rxd  (rx),

        .dout   (dout),
        .d_rdy  (d_rdy)
    );

    tx_ctl tx_ctl(
        .clk  (clk),
        .rst  (rst),
        .bclk(bclk),

        .din(din),
        .tx_en  (tx_en),

        .txd(tx),
        .tx_rdy(tx_rdy)
    );

endmodule
```

```
// rxd -- ( d0 d1 d2 d3 d4 d5 d6 d7 ) -- > dout [ d7 d6 d5 d4 d3 d2 d1 d0 ]
module rx_mod(
    input              clk,
    input              rst,
    input              bclk,
    input           rxd,

    output    [7:0] dout,
    output  reg     d_rdy
);

    localparam STARTBIT = 0;
    localparam STOPBIT  = 1;

    localparam IDLE     = 2'b00;
    localparam START    = 2'b01;
    localparam STOP     = 2'b10;

    reg [7:0] rhr        = 0;
    reg [2:0] d_ctr      = 0;
    reg [1:0] state      = 0;
    reg [1:0] next_state = 0;

    assign dout = rhr;

     always @( negedge bclk or posedge rst ) begin
        if( rst )
            state <= 0;
        else
            state <= next_state;
     end

    always @( posedge bclk or posedge rst )
        if( rst )begin
            rhr    <= 0;
            d_ctr  <= 0;
            d_rdy  <= 0;
        end else
            case( state )
                IDLE:
                begin
                    d_rdy <= 0;
                    if( rxd == STARTBIT ) begin
                        next_state  <= START;
                    end
                end
                START:
                    begin
                        d_ctr <= d_ctr + 1'b1;
                        //rhr = { rxd, rhr[6:0] };
                        rhr     <= rhr << 1;
                        rhr[0]  <= rxd;
                        if( d_ctr == 3'd7 ) begin
                            next_state <= STOP;
                            d_ctr      <= 0;
                        end
                    end
                STOP:
                    begin
                        if( rxd == STOPBIT ) begin
                            d_rdy  <= 1;
                        end
                        next_state  <= IDLE;
                    end
            endcase

endmodule
```

```
module tx_ctl(
    input           clk,
    input           rst,

    input           bclk,
    input   [7:0]   din,
    input           tx_en,  // din ready
//    input           cts,

    output          txd,
    output          tx_rdy  // 0 -- in process of transmission, 1 otherwise.
);

    reg wr = 0;
    reg rd = 0;
    reg en = 0;

    wire [7:0] fifo_dout;


    fifo tx_fifo(
        .clk(clk),
        .rst(rst),

        .rd (rd),
        .wr (wr),
        /* On next clk data is in memory. */
        .din(din),

        .full (fifo_full),
        .empty(fifo_empty),
        /* On next clk after the rd signal data is in dout. */
        .dout (fifo_dout)
    );

    tx_mod tx_mod(
        .clk(clk),
        .rst(rst),

        .bclk(bclk),
        /* Data to transmit. */
        .din(fifo_dout),
        /* Data ready to transmit. */
        .tx_en(en),
        /* TX-pin */
        .txd(txd),
        /* Transmitter is ready ( 1 ), is busy ( 0 ). */
        .tx_rdy(tx_rdy)
    );

    /* Read data from  buffer to thr for transmission.
     */
    always @( posedge clk )
        if( rst ) begin
            rd <= 0;
        end
        else if( rd ) begin
            rd <= 0;
        end
        /* If fifo isn't empty, tx ready for transmission and no writing
         * we can read from fifo.
         * Need tx_rdy to be in untill next transmision.
         */
        else if( !fifo_empty && tx_rdy && !wr ) begin
            rd <= 1;
        end
        else
            rd <= 0;


    /* Write data from input to fifo-buffer.*/
    always @( negedge clk )
        if( rst ) begin
            wr <= 0;
            en <= 0;
        end
        else begin
        /*
        * If fifo isn't full, input data is enabled  and no reading
        * we can write to fifo.
        */
            if( tx_en && !fifo_full && !rd )
                wr <= 1;
            else
                wr <= 0;

            if( rd )
                en <= 1;
            else
                en <= 0;
        end
endmodule
```

```verilog
// din [ d7 d6 d5 d4 d3 d2 d1 d0 ] > txd -- ( d7 d6 d5 d4 d3 d2 d1 d0 ) -->
module tx_mod(
    input           clk,
    input           rst,

    input           bclk,
    input   [7:0]   din,
    /* Data ready for transmission.*/
    input           tx_en,

    output  reg     txd,
    output  reg     tx_rdy  // 0 -- in process of transmission, 1 otherwise.
);

    localparam IDLE     = 2'b00;
    localparam START    = 2'b01;
    localparam TRANSMIT = 2'b10;
    localparam STOP     = 2'b11;

    localparam READY    = 1'd1;

    localparam START_BIT = 0;
    localparam STOP_BIT  = 1;

    reg [1:0] next_state = IDLE;
    reg [1:0] state      = IDLE;
    reg [1:0] was_state  = IDLE;
    reg [2:0] d_ctr      = 0;
    reg [7:0] tsr        = 0;
    reg [7:0] thr        = 0;

    reg rstate = 0;


    always @( posedge clk or posedge rst )
        if( rst ) begin
            rstate    = IDLE;
            tx_rdy    = 1;
            was_state = IDLE;
        end
        else
            case( rstate )
                IDLE:
                    if( tx_en ) begin
                        tx_rdy  = 0;
                        rstate  = READY;
                    end
                    else
                        tx_rdy = 1;
                READY:
                    if( state == IDLE && was_state == STOP ) begin
                        tx_rdy    = 1;
                        rstate    = IDLE;
                        was_state = state;
                    end
                    else
                        was_state = state;
            endcase

    always @( posedge bclk )
        if( rst )
            state <= 0;
        else
            state <= next_state;

    always @( negedge bclk or posedge rst ) begin
         if( rst )begin
            d_ctr  <= 0;
            txd    <= 1;
            tsr    <= 0;
        end
        else
            case( state )
                IDLE:
                    if( !tx_rdy ) begin
                        next_state  <= START;
                        tsr         <= din;
                    end
                START:
                    begin
                        next_state <= TRANSMIT;
                        txd        <= START_BIT;
                    end
                TRANSMIT:
                    begin
                        d_ctr <= d_ctr + 1;
                        txd   <= tsr[0];
                        tsr   <= { 1'b0, tsr[7:1] };
                        if( d_ctr == 3'd7 ) begin
                            next_state <= STOP;
                            d_ctr      <= 0;
                        end
                    end
                STOP:
                    begin
                        next_state <= IDLE;
                        txd        <= STOP_BIT;
                    end
            endcase
    end
endmodule
```

```verilog
/* Read and write on posedge clk.
 * Set states of fullness and emptiness by negedge.
 */
module fifo #(
    parameter DEPTH = 16,
    parameter CAPACITY = 8
) (
    input                           clk,
    input                           rst,

    input                           rd,
    input                           wr,
    input          [CAPACITY-1:0]   din,

    output                          full,
    output                          empty,
    output    reg  [CAPACITY-1:0]   dout
);

    integer i;
    localparam WRITE = 0;
    localparam READ  = 1;

    reg [CAPACITY-1:0]       mem [DEPTH-1:0];
    reg [$clog2(DEPTH)-1:0]  raddr    = 0;
    reg [$clog2(DEPTH)-1:0]  waddr    = 0;

    reg state;
    reg fifo_full;
    reg fifo_empty;

    assign empty = fifo_empty;
    assign full  = fifo_full;

    always @( posedge clk or posedge rst ) begin
        if( rst ) begin
            for( i = 0; i <= DEPTH-1; i = i + 1 )
                mem[i] <= 0;
            waddr     <= 0;
            raddr     <= 0;
            dout      <= 0;
        end
        else begin
            if( wr & !fifo_full ) begin
                mem[waddr] <= din;
                waddr      <= waddr+1'b1;
                state      <= WRITE;
            end
            else if( rd & !empty ) begin
                dout  <= mem[raddr];
                raddr <= raddr + 1'b1;
                state <= READ;
            end
        end
    end

    always @( negedge clk or posedge rst )
        if( rst ) begin
            fifo_full  <= 0;
            fifo_empty <= 1;
        end
        else
            case( state )
                READ:
                begin
                    fifo_full <= 0;
                    if( waddr == raddr )
                        fifo_empty <= 1;
                    else
                        fifo_empty <= 0;
                end
                WRITE:
                begin
                    fifo_empty <= 0;
                    if( waddr == raddr )
                        fifo_full <= 1;
                    else
                        fifo_full <= 0;
                end
            endcase

endmodule
```

Вывод

В ходы выполнения лабораторной работы были исследованы принципы работы последовательных интерфейсов I/O и в результате разработан универсальный асинхронный интерфейс UART.