

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.

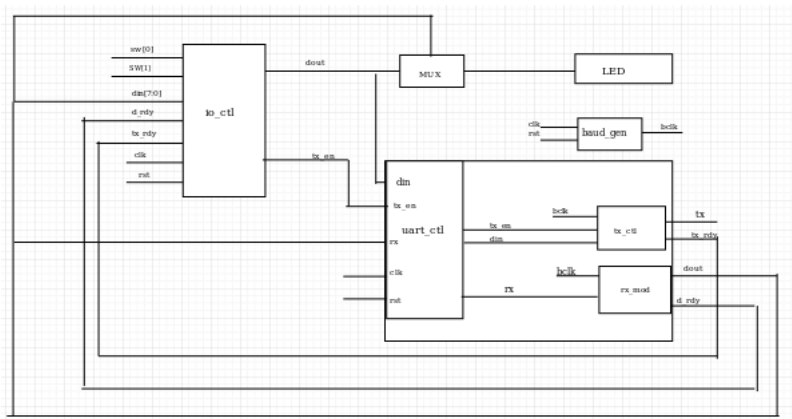
Содержание

- 1. Цели работы.
- 2. RTL модель.
- 3. Временные диаграммы.
- 4. Листинг.
- 5. Вывод.

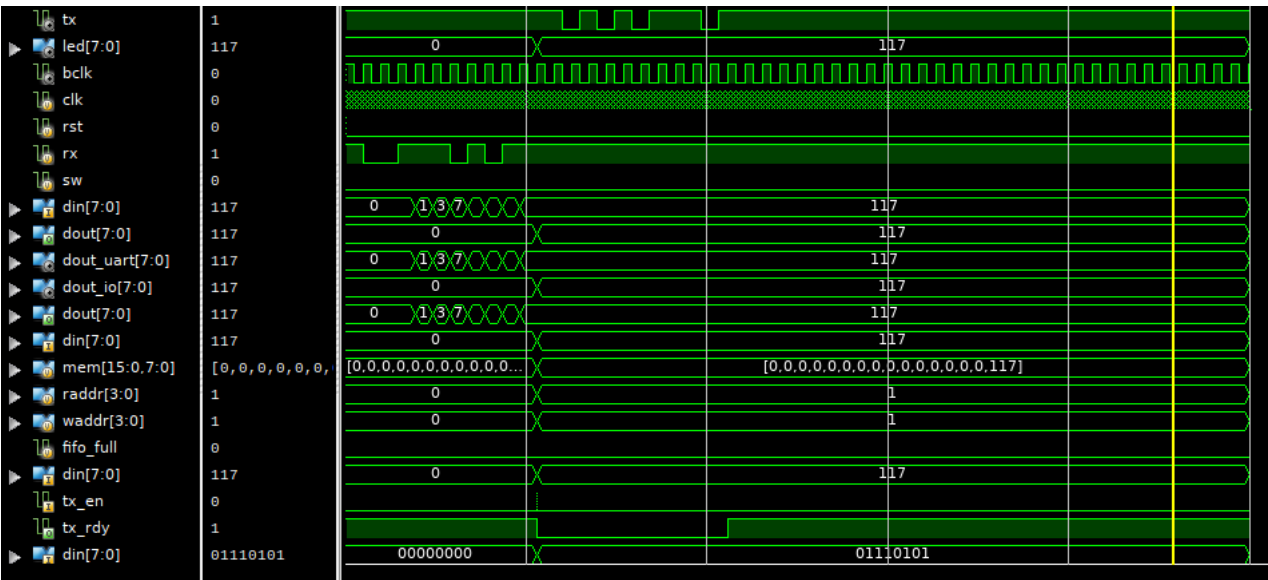
Цели работы

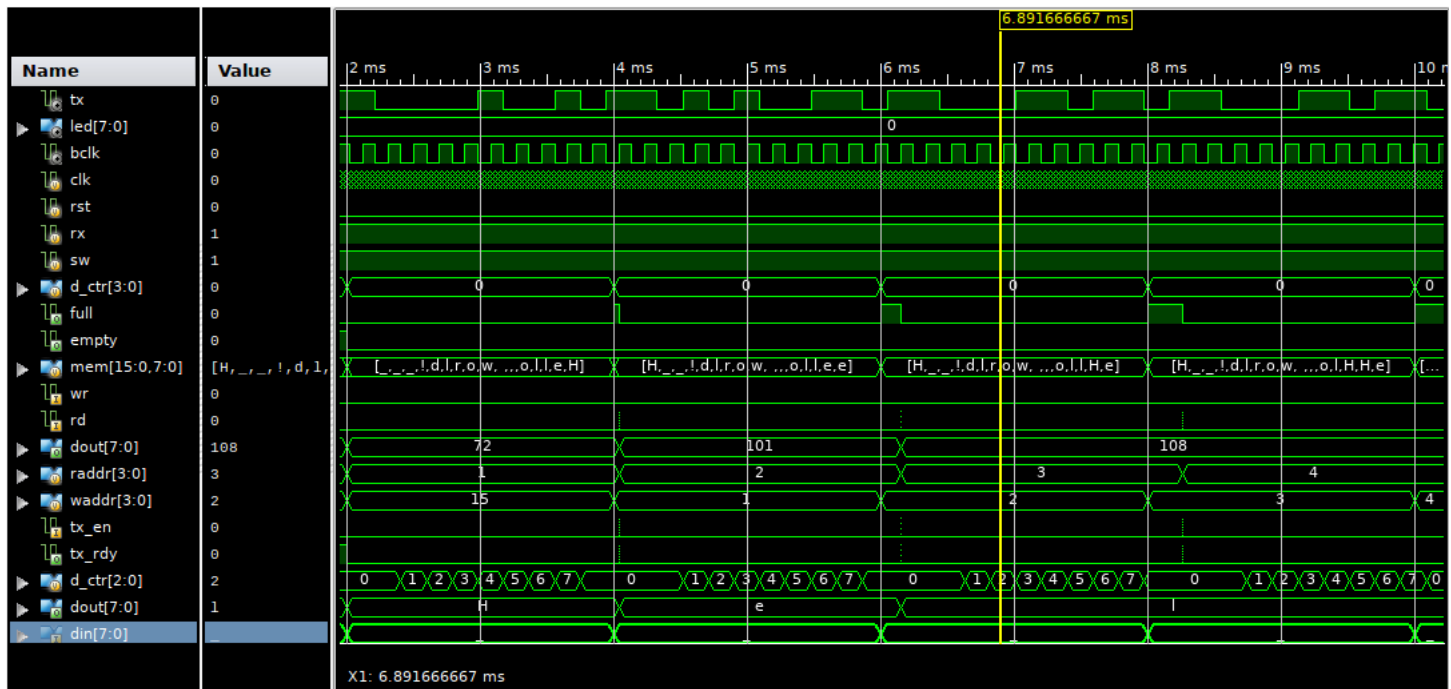
- 1. Знакомство с принципами работы последовательных интерфейсов ввода/вывода: I2C, SPI, UART.
- 2. Изучение основ разработки аппаратных контроллеров периферийных устройств.
- 3. Изучение основ работы с цифровыми датчиками.

RTL модель



Временные диаграммы





ЛИСТИНГ

```

1  /* Top module. */
2  module ctl(
3      input          clk ,
4      input          rst ,
5
6      /* UART interface. */
7      input          rx ,
8      // input          cts ,
9
10     output          tx ,
11     // output          rts ,
12
13     /* Nexys interface.*/
14     input          sw ,
15     output reg[7:0] led
16 );
17
18     wire [7:0] dout_uart;
19     wire [7:0] dout_io;
20     wire tx_en;
21     wire tx_rdy;
22     wire d_rdy;
23
24     io_ctl io_ctl(
25         .clk(clk),
26         .rst(rst),
27
28         .sw(sw),
29
30         .din(dout_uart),
31         .d_rdy(d_rdy),
32         .tx_rdy(tx_rdy),
33
34         .dout(dout_io),
35         .tx_en(tx_en) // dout_io is ready.
36     );
37
38     uart_ctl uart_ctl(
39
40         .clk(clk),
41         .rst(rst),
42
43         .rx(rx),
44         .din(dout_io),
45         .tx_en(tx_en),
46

```

```

47     .tx (tx),
48     .dout(dout_uart),
49     .d_rdy(d_rdy),
50     .tx_rdy(tx_rdy)
51 );
52
53     always @( negedge clk )
54         if( rst )
55             led <= 0;
56         else if( d_rdy )
57             led <= dout_uart;
58
59
60 endmodule

1 module io_ctl(
2     input                clk ,
3     input                rst ,
4     /* SW = 0: echo-mode. SW = 1: send 'Hello world!\r\n'. */
5     input                sw,
6     /* Data from uart.*/
7     input  [7:0]         din ,
8     input                d_rdy,
9     input                tx_rdy,
10    /* Data to uart. */
11    output reg [7:0]      dout ,
12    output reg            tx_en
13 );
14    localparam TIME      = 100000;
15    localparam ECHO_MODE = 0;
16    localparam SEND_MODE = 1;
17
18    reg [7:0]  data [14:0];
19    reg [3:0]  d_ctr   = 0;
20    reg [26:0] tm_ctr   = 0;
21    reg was_d_rdy      = 0;
22
23    wire tx_flag;
24
25    assign tx_flag = (tm_ctr == TIME) ? 1 : ((( d_ctr == 15 ) || ( d_ctr == 0 )) ? 0 : 1 );
26
27    always @( posedge rst ) begin
28        data[0] = "H"; data[1] = "e"; data[2] = "l";
29        data[3] = "l"; data[4] = "o"; data[5] = ",";
30        data[6] = " "; data[7] = "w"; data[8] = "o";
31        data[9] = "r"; data[10] = "l"; data[11] = "d";
32        data[12] = "!"; data[13] = 8'h0D; data[14] = 8'h0A;
33    end
34
35    always @( negedge clk or posedge rst )
36        if( rst ) begin
37            tx_en <= 0;
38            dout <= 0;
39            d_ctr <= 0;
40            was_d_rdy <= 0;
41        end else
42            case( sw )
43                ECHO_MODE:
44                    if( d_rdy && !was_d_rdy ) begin
45                        dout <= din;
46                        tx_en <= 1;
47                        was_d_rdy <= d_rdy;
48                    end
49                    else begin
50                        tx_en <= 0;
51                        was_d_rdy <= d_rdy;
52                    end
53                SEND_MODE:
54                    begin
55                        if( tx_flag ) begin
56                            tx_en <= 1;
57                            dout <= data[d_ctr];
58                            d_ctr <= d_ctr + 1;
59                        end
60                        else if( d_ctr == 15 ) begin
61                            tx_en <= 0;
62                            d_ctr <= 0;
63                        end

```

```

64         end
65     endcase
66
67     always @( posedge clk or posedge rst ) begin
68         if( rst )
69             tm_ctr <= 0;
70         else if( sw == SEND_MODE ) begin
71             if( tm_ctr == TIME )
72                 tm_ctr <= 0;
73             else
74                 tm_ctr <= tm_ctr + 1;
75         end
76     end
77
78 endmodule

```

```

1 module uart_ctl(
2     input          clk ,
3     input          rst ,
4
5     input          rx ,
6 // input          rts ,
7
8     // Data to send.
9     input  [7:0]  din ,
10    // Enable transmission ( data ready ).
11    input          tx_en ,
12
13    // Transmission wire.
14    output         tx ,
15 // output         cts ,
16
17    // Received data.
18    output  [7:0]  dout ,
19    // Receive ends; data ready.
20    output         d_rdy ,
21    // Controller ready for transmission.
22    output         tx_rdy
23 );
24
25 wire      bclk;
26
27 baud_gen baud_gen(
28     .clk( clk ),
29     .rst( rst ),
30
31     .bclk( bclk )
32 );
33
34 rx_mod rx_mod(
35     .clk  ( clk ),
36     .rst  ( rst ),
37     .bclk( bclk ),
38     .rx   ( rx ),
39
40     .dout  ( dout ),
41     .d_rdy ( d_rdy )
42 );
43
44 tx_ctl tx_ctl(
45     .clk  ( clk ),
46     .rst  ( rst ),
47     .bclk( bclk ),
48
49     .din( din ),
50     .tx_en ( tx_en ),
51
52     .txd( tx ),
53     .tx_rdy( tx_rdy )
54 );
55
56 endmodule

```

```

1 // rxd — ( d0 d1 d2 d3 d4 d5 d6 d7 ) — > dout [ d7 d6 d5 d4 d3 d2 d1 d0 ]
2 module rx_mod(
3     input          clk ,
4     input          rst ,
5     input          bclk ,

```

```

6      input          rxd ,
7
8      output [7:0] dout ,
9      output reg     d_rdy
10 );
11
12      localparam STARTBIT = 0;
13      localparam STOPBIT  = 1;
14
15      localparam IDLE      = 2'b00;
16      localparam START     = 2'b01;
17      localparam STOP      = 2'b10;
18
19      reg [7:0] rhr         = 0;
20      reg [2:0] d_ctr       = 0;
21      reg [1:0] state       = 0;
22      reg [1:0] next_state = 0;
23
24      assign dout = rhr;
25
26      always @( negedge bclk or posedge rst ) begin
27          if( rst )
28              state <= 0;
29          else
30              state <= next_state;
31      end
32
33      always @( posedge bclk or posedge rst )
34          if( rst )begin
35              rhr <= 0;
36              d_ctr <= 0;
37              d_rdy <= 0;
38          end else
39              case( state )
40                  IDLE:
41                      begin
42                          d_rdy <= 0;
43                          if( rxd == STARTBIT ) begin
44                              next_state <= START;
45                          end
46                      end
47                  START:
48                      begin
49                          d_ctr <= d_ctr + 1'b1;
50                          //rhr = { rxd, rhr[6:0] };
51                          rhr <= rhr << 1;
52                          rhr[0] <= rxd;
53                          if( d_ctr == 3'd7 ) begin
54                              next_state <= STOP;
55                              d_ctr <= 0;
56                          end
57                      end
58                  STOP:
59                      begin
60                          if( rxd == STOPBIT ) begin
61                              d_rdy <= 1;
62                          end
63                          next_state <= IDLE;
64                      end
65              endcase
66
67      endmodule

```

```

1 module tx_ctl(
2     input          clk ,
3     input          rst ,
4
5     input          bclk ,
6     input [7:0]    din ,
7     input          tx_en, // din ready
8     // input          cts ,
9
10    output          txd ,
11    output          tx_rdy // 0 — in process of transmission, 1 otherwise.
12 );
13
14    reg wr = 0;
15    reg rd = 0;

```

```

16  reg en = 0;
17
18  wire [7:0] fifo_dout;
19
20
21  fifo tx_fifo(
22      .clk(clk),
23      .rst(rst),
24
25      .rd (rd),
26      .wr (wr),
27      /* On next clk data is in memory. */
28      .din(din),
29
30      .full (fifo_full),
31      .empty(fifo_empty),
32      /* On next clk after the rd signal data is in dout. */
33      .dout (fifo_dout)
34  );
35
36  tx_mod tx_mod(
37      .clk(clk),
38      .rst(rst),
39
40      .bclk(bclk),
41      /* Data to transmit. */
42      .din(fifo_dout),
43      /* Data ready to transmit. */
44      .tx_en(en),
45      /* TX-pin */
46      .txd(txd),
47      /* Transmitter is ready ( 1 ), is busy ( 0 ). */
48      .tx_rdy(tx_rdy)
49  );
50
51  /* Read data from buffer to thr for transmission.
52  */
53  always @( posedge clk )
54      if( rst ) begin
55          rd <= 0;
56      end
57      else if( rd ) begin
58          rd <= 0;
59      end
60      /* If fifo isn't empty, tx ready for transmission and no writing
61      * we can read from fifo.
62      * Need tx_rdy to be in untill next transmission.
63      */
64      else if( !fifo_empty && tx_rdy && !wr ) begin
65          rd <= 1;
66      end
67      else
68          rd <= 0;
69
70
71  /* Write data from input to fifo-buffer.*/
72  always @( negedge clk )
73      if( rst ) begin
74          wr <= 0;
75          en <= 0;
76      end
77      else begin
78          /*
79          * If fifo isn't full, input data is enabled and no reading
80          * we can write to fifo.
81          */
82          if( tx_en && !fifo_full && !rd )
83              wr <= 1;
84          else
85              wr <= 0;
86
87          if( rd )
88              en <= 1;
89          else
90              en <= 0;
91      end
92  endmodule

```

```

1 // din [ d7 d6 d5 d4 d3 d2 d1 d0 ] > txd — ( d7 d6 d5 d4 d3 d2 d1 d0 ) —>
2 module tx_mod(
3     input          clk ,
4     input          rst ,
5
6     input          bclk ,
7     input [7:0]    din ,
8     /* Data ready for transmission.*/
9     input          tx_en ,
10
11     output reg     txd ,
12     output reg     tx_rdy // 0 — in process of transmission, 1 otherwise.
13 );
14
15 localparam IDLE      = 2'b00;
16 localparam START     = 2'b01;
17 localparam TRANSMIT  = 2'b10;
18 localparam STOP      = 2'b11;
19
20 localparam READY     = 1'd1;
21
22 localparam START_BIT = 0;
23 localparam STOP_BIT  = 1;
24
25 reg [1:0] next_state = IDLE;
26 reg [1:0] state      = IDLE;
27 reg [1:0] was_state  = IDLE;
28 reg [2:0] d_ctr      = 0;
29 reg [7:0] tsr        = 0;
30 reg [7:0] thr        = 0;
31
32 reg rstate = 0;
33
34
35 always @( posedge clk or posedge rst )
36     if( rst ) begin
37         rstate = IDLE;
38         tx_rdy = 1;
39         was_state = IDLE;
40     end
41     else
42         case( rstate )
43             IDLE:
44                 if( tx_en ) begin
45                     tx_rdy = 0;
46                     rstate = READY;
47                 end
48                 else
49                     tx_rdy = 1;
50             READY:
51                 if( state == IDLE && was_state == STOP ) begin
52                     tx_rdy = 1;
53                     rstate = IDLE;
54                     was_state = state;
55                 end
56                 else
57                     was_state = state;
58             endcase
59
60 always @( posedge bclk )
61     if( rst )
62         state <= 0;
63     else
64         state <= next_state;
65
66 always @( negedge bclk or posedge rst ) begin
67     if( rst )begin
68         d_ctr <= 0;
69         txd <= 1;
70         tsr <= 0;
71     end
72     else
73         case( state )
74             IDLE:
75                 if( !tx_rdy ) begin
76                     next_state <= START;
77                     tsr <= din;
78                 end

```



```

79         START:
80             begin
81                 next_state <= TRANSMIT;
82                 txd         <= START_BIT;
83             end
84         TRANSMIT:
85             begin
86                 d_ctr <= d_ctr + 1;
87                 txd    <= tsr[0];
88                 tsr    <= { 1'b0, tsr[7:1] };
89                 if( d_ctr == 3'd7 ) begin
90                     next_state <= STOP;
91                     d_ctr      <= 0;
92                 end
93             end
94         STOP:
95             begin
96                 next_state <= IDLE;
97                 txd        <= STOP_BIT;
98             end
99     endcase
100 end
101 endmodule

```

```

1  /* Read and write on posedge clk.
2  * Set states of fullness and emptiness by negedge.
3  */
4  module fifo #(
5      parameter DEPTH = 16,
6      parameter CAPACITY = 8
7  ) (
8      input          clk ,
9      input          rst ,
10
11     input          rd ,
12     input          wr ,
13     input          [CAPACITY-1:0] din ,
14
15     output          full ,
16     output          empty ,
17     output reg [CAPACITY-1:0] dout
18 );
19
20     integer i;
21     localparam WRITE = 0;
22     localparam READ  = 1;
23
24     reg [CAPACITY-1:0] mem [DEPTH-1:0];
25     reg [$clog2(DEPTH)-1:0] raddr = 0;
26     reg [$clog2(DEPTH)-1:0] waddr = 0;
27
28     reg state;
29     reg fifo_full;
30     reg fifo_empty;
31
32     assign empty = fifo_empty;
33     assign full  = fifo_full;
34
35     always @( posedge clk or posedge rst ) begin
36         if( rst ) begin
37             for( i = 0; i <= DEPTH-1; i = i + 1 )
38                 mem[i] <= 0;
39             waddr <= 0;
40             raddr <= 0;
41             dout  <= 0;
42         end
43         else begin
44             if( wr & !fifo_full ) begin
45                 mem[waddr] <= din;
46                 waddr <= waddr+1'b1;
47                 state <= WRITE;
48             end
49             else if( rd & !fifo_empty ) begin
50                 dout <= mem[raddr];
51                 raddr <= raddr + 1'b1;
52                 state <= READ;
53             end
54         end
55     end

```

```

55     end
56
57     always @( negedge clk or posedge rst )
58         if( rst ) begin
59             fifo_full  <= 0;
60             fifo_empty <= 1;
61         end
62         else
63             case( state )
64                 READ:
65                     begin
66                         fifo_full <= 0;
67                         if( waddr == raddr )
68                             fifo_empty <= 1;
69                         else
70                             fifo_empty <= 0;
71                     end
72                 WRITE:
73                     begin
74                         fifo_empty <= 0;
75                         if( waddr == raddr )
76                             fifo_full <= 1;
77                         else
78                             fifo_full <= 0;
79                     end
80             endcase
81
82 endmodule

```

Вывод

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