Университет ИТМО

Кафедра вычислительной техники

Отчёт по лабораторной работе № 2

по дисциплине: "Схемотехника ЭВМ"

Вариант №5

Студенты: Куклина М.Д.
Кириллова А.А.

Преподаватель:

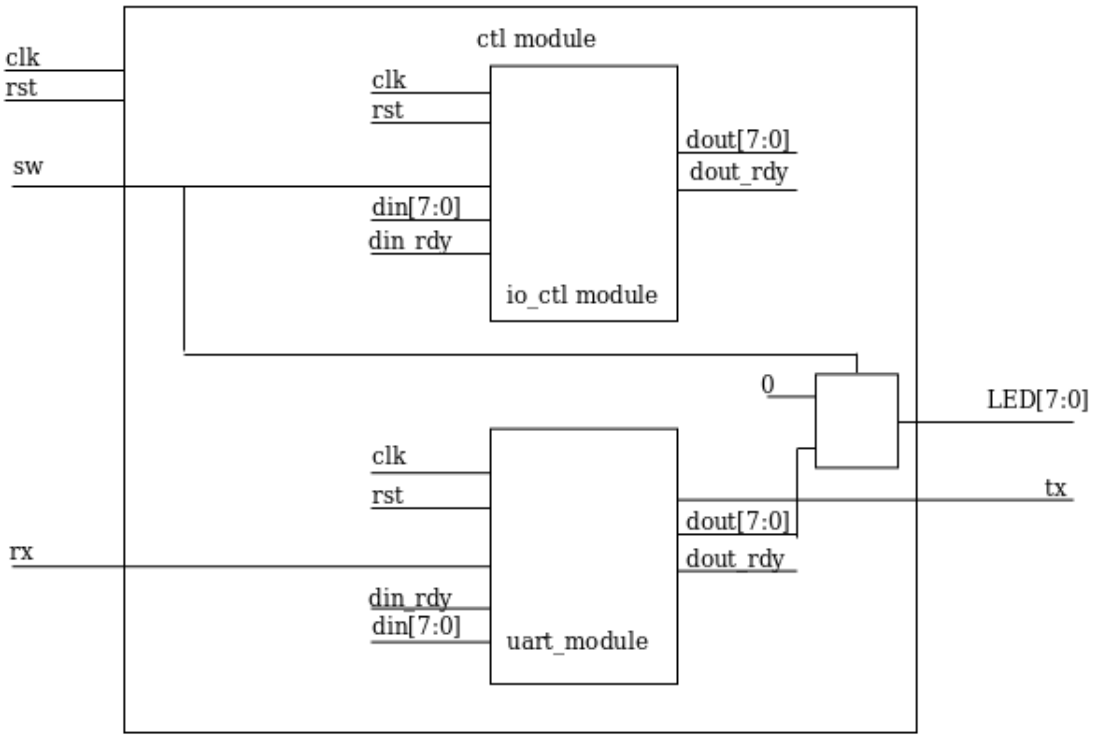
Санкт-Петербург
2016 г.

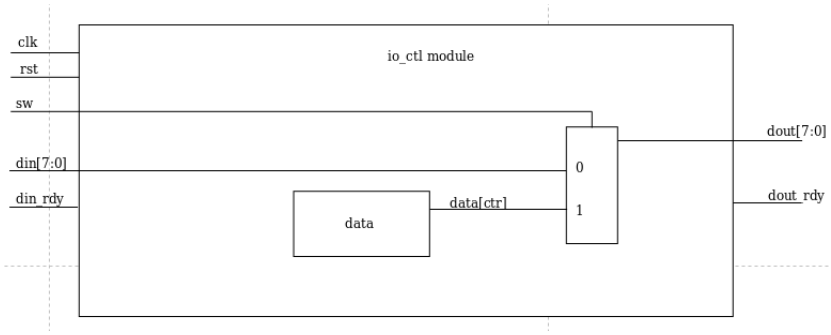
Содержание

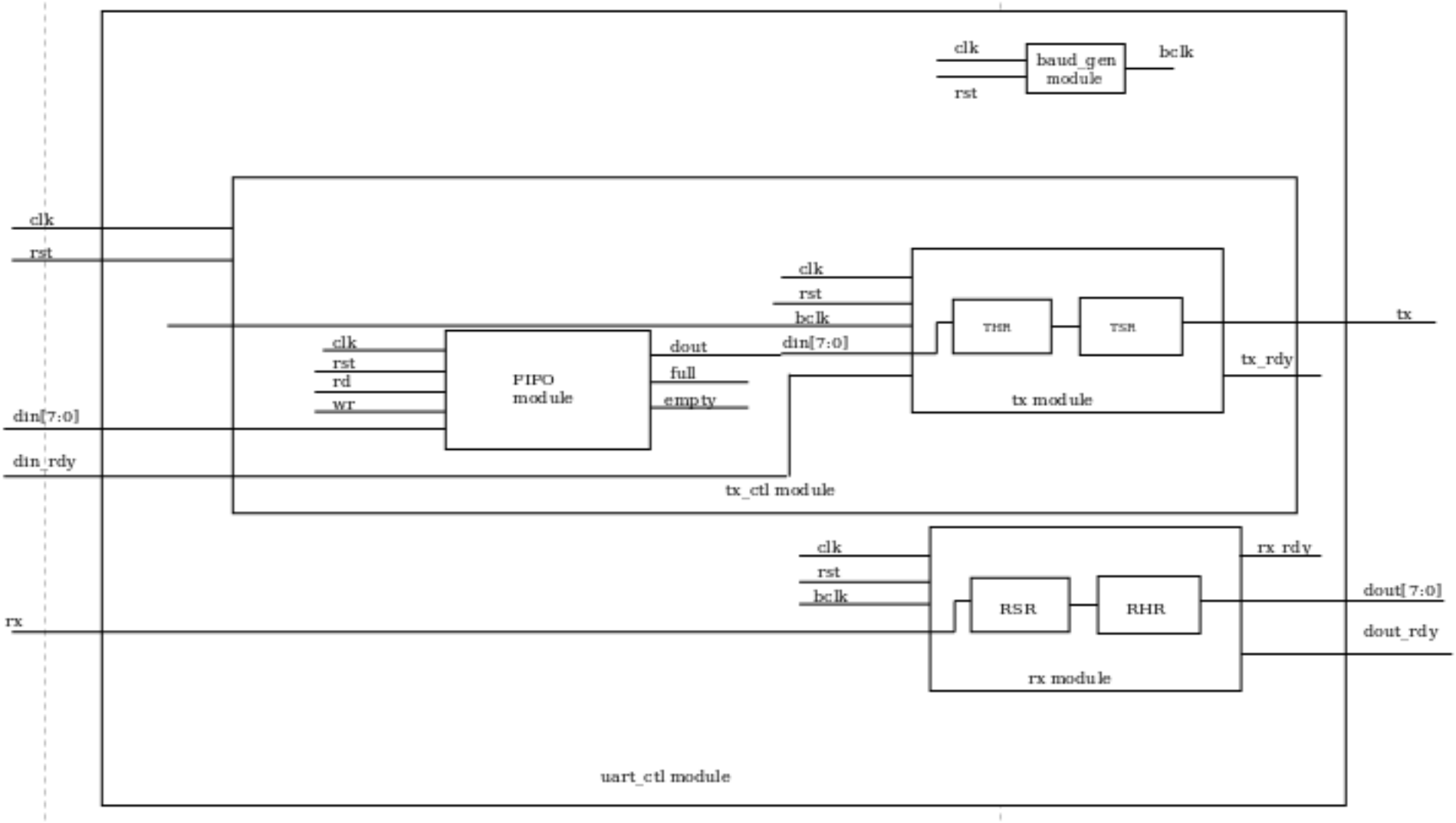
Цели работы

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.

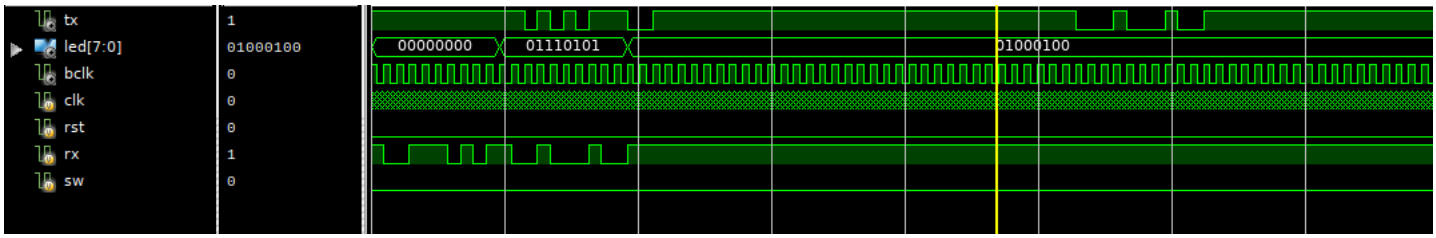
RTL модель

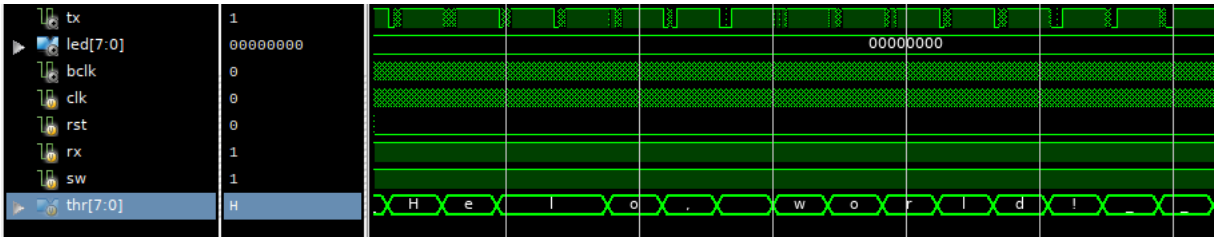





clk
rst
baud_gen module
bclk
clk
rst
bclk
din[7:0]
clk
rst
rd
wr
FIFO module
dout
full
empty
THR
TSR
tx
tx_rdy
tx module
tx_ctl module
din[7:0]
din_rdy
clk
rst
bclk
RSR
RHR
rx_rdy
dout[7:0]
dout_rdy
rx
rx module
uart_ctl module

Временные диаграммы





Листинг

```
1  /* Top module. */
2  module ctl(
3      input               clk,
4      input               rst,
5
6      /* UART interface. */
7      input               rx,
8      output              tx,
9
10     /* Nexys interface.*/
11     input                 sw,
12     output  reg[7:0]    led
13 );
14
15     wire [7:0] dout_uart;
16     wire [7:0] dout_io;
17
18     wire dout_io_rdy;
19     wire dour_uart_rdy;
20
21     io_ctl #( 100000000 )io_ctl(
22         .clk(clk),
23         .rst(rst),
24
25         .sw(sw),
26
27         .din     (dout_uart), /* dout from uart_ctl to io_ctl din */
28         .din_rdy(dout_uart_rdy),
29
30         .dout     (dout_io),
31         .dout_rdy(dout_io_rdy) /* dout_io is ready. */
32     );
33
34     uart_ctl uart_ctl(
35         .clk(clk),
36         .rst(rst),
37
38         .rx      (rx),
39         .din     (dout_io),
40         .din_rdy(dout_io_rdy),
41
42         .tx       (tx),
43         .dout     (dout_uart),
44         .dout_rdy(dout_uart_rdy)
45     );
46
47      always @(negedge clk)
48          if (rst)
```

```
49              led <= 0;
50            /* SW == 0: echo mode; SW = 1: send mode. */
51            else if ( !sw && dout_uart_rdy)
52              led <= dout_uart;
53
54
55 endmodule
```

```
1  module io_ctl #(
2      parameter TIME = 100000000
3  ) (
4      input                clk,
5      input                rst,
6      /* SW = 0: echo-mode. SW = 1: send 'Hello world!\r\n'. */
7      input                sw,
8      /* Data from uart.*/
9      input    [7:0]       din,
10     /* Data to in is ready. */
11     input                din_rdy,
12     /* Allow read from memory in send mode. */
13     //input                tx_en,
14     /* Data to uart. */
15     output  reg[7:0]     dout,
16     /* Data to out is ready. */
17     output  reg          dout_rdy
18 );
19     localparam ECHO_MODE = 0;
20     localparam SEND_MODE = 1;
21
22     reg [7:0]  data [14:0];
23     reg [26:0] tm_ctr = 0;
24     reg [3:0]  d_ctr  = 0;
25     reg was_din_rdy     = 0;
26
27
28     /* Have no idea what it is. */
29     wire tx_flag;
30     assign tx_flag = (tm_ctr == TIME)                    ? 1 :
31                      (( d_ctr == 15 ) || ( d_ctr == 0 )) ? 0 : 1;
32
33     always @( posedge rst ) begin
34             data[0]  = "H"; data[1]  = "e";   data[2]  = "l";
35             data[3]  = "l"; data[4]  = "o";   data[5]  = ",";
36             data[6]  = " "; data[7]  = "w";   data[8]  = "o";
37             data[9]  = "r"; data[10] = "l";   data[11] = "d";
38             data[12] = "!";
39             data[13] = 8'h0D; /* \r */
40             data[14] = 8'h0A; /* \n */
41     end
42
43     always @( negedge clk or posedge rst ) begin
44         if( rst )
45             tm_ctr <= 0;
46         else if( sw == SEND_MODE ) begin
47             if( tm_ctr == TIME )
48                 tm_ctr  <= 0;
49             else
50                 tm_ctr  <= tm_ctr + 1;
51         end
52     end
53
54     always @( posedge clk or posedge rst ) begin
55         if( rst ) begin
56             was_din_rdy <= 0;
57             dout_rdy    <= 0;
58             dout        <= 0;
59             d_ctr       <= 0;
60         end
61         else
62             case( sw )
63                 ECHO_MODE:
64                 begin
65                     if( din_rdy && !was_din_rdy ) begin
66                         was_din_rdy <= din_rdy;
67                         dout        <= din;
68                         dout_rdy    <= 1;
69                     end
70                     else begin
```

```
71                          was_din_rdy <= din_rdy;
72                          dout_rdy    <= 0;
73                      end
74                  end
75                  SEND_MODE:
76                  begin
77                      if( tx_flag ) begin
78                          dout_rdy <= 1;
79                          dout     <= data[d_ctr];
80                          d_ctr    <= d_ctr + 1;
81                      end
82                      else if( d_ctr == 15 ) begin
83                          dout_rdy <= 0;
84                          d_ctr    <= 0;
85                      end
86                  end
87              endcase
88      end
89
90  endmodule
```

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

```
1  module rx(
2      input               clk,
3      input               rst,
4      input               bclk,
5      input               rx,
6
7      output  reg[7:0]    dout,
8      output  reg         dout_rdy,
9      output  reg         rx_rdy
10 );
11
12     localparam START_BIT = 0;
13     localparam STOP_BIT  = 1;
14
15     localparam IDLE      = 2'd0;
16     localparam START     = 2'd1;
17     localparam STOP      = 2'd2;
18
19
20     reg [7:0] rsr    = 0;
21
22     reg [2:0] d_ctr      = 0;
23     reg [1:0] next_state = 0;
24     reg [1:0] state      = 0;
25
26     reg was_bclk  = 0;
27
28
29     always @(negedge clk or posedge rst) begin
30         if (rst)
31             state <= IDLE;
32         else
33             if (!bclk && was_bclk)
34                 state <= next_state;
35     end
36
37     always @(posedge clk or posedge rst) begin
38         if (rst) begin
39             next_state <= 0;
40             was_bclk   <= 0;
41             dout_rdy   <= 0;
42             rx_rdy     <= 1;
43             d_ctr      <= 0;
44             rsr        <= 0;
45             dout       <= 0;
46         end
47         else begin
48             if (bclk && !was_bclk) begin
49                 was_bclk <= bclk;
50                 case (state)
51                     IDLE:
52                     begin
53                         dout_rdy <= 0;
54                         if (START_BIT == rx) begin
55                             next_state <= START;
56                             rx_rdy     <= 0;
57                         end
58                     end
59                     START:
60                     begin
61                         d_ctr  <= d_ctr + 1'b1;
62                         rsr    <= rsr << 1;
63                         rsr[0] <= rx;
64                         if (3'd7 == d_ctr) begin
65                             next_state <= STOP;
66                             d_ctr      <= 0;
67                         end
68                     end
69                     STOP:
70                     begin
71                         if (STOP_BIT == rx) begin
72                             dout_rdy <= 1;
73                             dout     <= rsr;
74                         end
75                         next_state <= IDLE;
76                         rx_rdy     <= 0;
77                     end
78                 endcase
```

```
79              end
80              else
81                  was_bclk <= bclk;
82          end
83      end
84  endmodule
```

```
1  module tx_ctl(
2      input             clk,
3      input             rst,
4
5      input             bclk,
6      input    [7:0]    din,
7      input             din_rdy,
8
9      output            tx,
10     output            tx_rdy
11 );
12
13     reg wr = 0;
14     reg rd = 0;
15     reg en = 0;
16
17     wire [7:0] fifo_dout;
18
19
20     fifo tx_fifo(
21         .clk(clk),
22         .rst(rst),
23
24         .rd (rd),
25         .wr (wr),
26         /* On next clk data is in memory. */
27         .din(din),
28
29         .full (fifo_full),
30         .empty(fifo_empty),
31         /* On next clk after the rd signal data is in dout. */
32         .dout (fifo_dout)
33     );
34
35     tx tx_mod(
36         .clk(clk),
37         .rst(rst),
38
39         .bclk(bclk),
40         /* Data to transmit. */
41         .din(fifo_dout),
42         /* Data ready to transmit. */
43         .din_rdy(en),
44         /* TX-pin */
45         .tx(tx),
46         /* Transmitter is ready ( 1 ), is busy ( 0 ). */
47         .tx_rdy(tx_rdy)
48     );
49
50     /* Read data from  buffer to thr for transmission.
51      */
52     always @( posedge clk )
53         if( rst ) begin
54             rd <= 0;
55         end
56         else if( rd ) begin
57             rd <= 0;
58         end
59         /* If fifo isn't empty, tx ready for transmission and no writing
60          * we can read from fifo.
61          * Need tx_rdy to be in untill next transmision.
62          */
63         else if( !fifo_empty && tx_rdy && !wr ) begin
64             rd <= 1;
65         end
66         else
67             rd <= 0;
68
69
70     /* Write data from input to fifo-buffer.*/
71     always @( negedge clk )
```

```
72          if( rst ) begin
73              wr <= 0;
74              en <= 0;
75          end
76          else begin
77          /*
78           * If fifo isn't full, input data is enabled  and no reading
79           * we can write to fifo.
80           */
81              if( din_rdy && !fifo_full && !rd )
82                  wr <= 1;
83              else
84                  wr <= 0;
85
86              if( rd )
87                  en <= 1;
88              else
89                  en <= 0;
90          end
91 endmodule
```

```
1  module tx(
2      input           clk,
3      input           rst,
4
5      input           bclk,
6      input    [7:0]  din,
7      input           din_rdy,
8
9      output  reg     tx,
10     output  reg     tx_rdy
11
12 );
13
14     localparam IDLE      = 3'd0;
15     localparam START     = 3'd1;
16     localparam TRANSMIT  = 3'd2;
17     localparam STOP      = 3'd4;
18     localparam WAIT      = 3'd5;
19
20     localparam START_BIT = 1'b0;
21     localparam STOP_BIT  = 1'b1;
22
23     localparam WAIT_TIME_IN_BAUDS = 30;
24
25     reg [7:0] thr         = 0;
26     reg [7:0] tsr         = 0;
27     reg [4:0] wait_time   = 0;
28     reg [3:0] dctr        = 0;
29     reg [2:0] next_state  = 0;
30     reg [2:0] was_state   = 0;
31     reg [2:0] state       = 0;
32     reg       was_bclk    = 0;
33     reg       tx_en       = 0;
34     reg       rdy         = 0;
35
36     always @(negedge clk or posedge rst)
37         if (rst)
38             state = IDLE;
39         else
40             if (!bclk && was_bclk)
41                 state = next_state;
42
43    always @(negedge clk or posedge rst)
44         if (rst) begin
45             tx_en = 0;
46             thr   = 0;
47        end
48        else begin
49            if (din_rdy & !rdy) begin
50                tx_rdy = 0;
51                thr    = din;
52                tx_en  = 1;
53                rdy    = 1;
54            end
55            else
56                if (was_state == IDLE && state == START) begin
57                    tx_en = 0;
```

```
58                 end
59                 if (was_state == WAIT && state == IDLE) begin
60                     tx_rdy = 1;
61                     rdy    = 0;
62                 end
63         end
64
65
66     always @(posedge clk or posedge rst)
67         if (rst) begin
68             next_state <= IDLE;
69             wait_time  <= 0;
70             was_bclk   <= 0;
71             dctr       <= 0;
72             thr        <= 0;
73             tx_rdy     <= 1;
74             tx        <= 1;
75         end
76         else begin
77             was_state <= state;
78             if (bclk & !was_bclk) begin
79                 was_bclk  <= bclk;
80                 case( state )
81                     IDLE:
82                         if (tx_en) begin
83                             next_state <= START;
84                             tsr        <= thr;
85                         end
86                     START:
87                     begin
88                         next_state <= TRANSMIT;
89                         tx         <= START_BIT;
90                     end
91                     TRANSMIT:
92                     begin
93                         tx  <= tsr[0];
94                         tsr  <= tsr >> 1;
95                         dctr <= dctr + 1'b1;
96                         if ( 8 == dctr) begin
97                             next_state <= STOP;
98                             dctr       <= 0;
99                         end
100                    end
101                    STOP:
102                    begin
103                        next_state <= WAIT;
104                        tx         <= STOP_BIT;
105                    end
106                    /* Need to wait between words sending.
107                     * */
108                    WAIT:
109                    begin
110                        wait_time <= wait_time + 1'b1;
111                        if (wait_time == WAIT_TIME_IN_BAUDS) begin
112                            next_state <= IDLE;
113                            wait_time  <= 0;
114                        end
115                    end
116                endcase
117            end
118            else
119                was_bclk <= bclk;
120        end
121
122 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 & !empty ) begin
50                  dout  <= mem[raddr];
51                  raddr <= raddr + 1'b1;
52                  state <= READ;
53              end
54          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
```

```
1  /* Generates bauds. */
2  /* For 9600 bps 8N1 -- (10417) 9600 bauds. */
3  module baud_gen #(
4      parameter BAUDRATE = 9600
5  )
6  (
7      input           clk,
```

```
8       input               rst,
9
10      output    reg       bclk
11  );
12      reg [16:0] ctr   = BAUDRATE;
13
14      always @( posedge clk ) begin
15        if( rst )
16          ctr <= BAUDRATE;
17
18          if( ctr == BAUDRATE )
19              bclk <= 1;
20          else if( ctr == BAUDRATE/2 )
21              bclk <= 0;
22
23          if( ctr == 0 )
24              ctr <= BAUDRATE;
25          else
26              ctr <= ctr - 16'd1;
27      end
28
29  endmodule
```

Вывод

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