Университет ИТМО

Кафедра вычислительной техники

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

Студенты:
Куклина М.
Кириллова А.

Преподаватель: Баевских А.

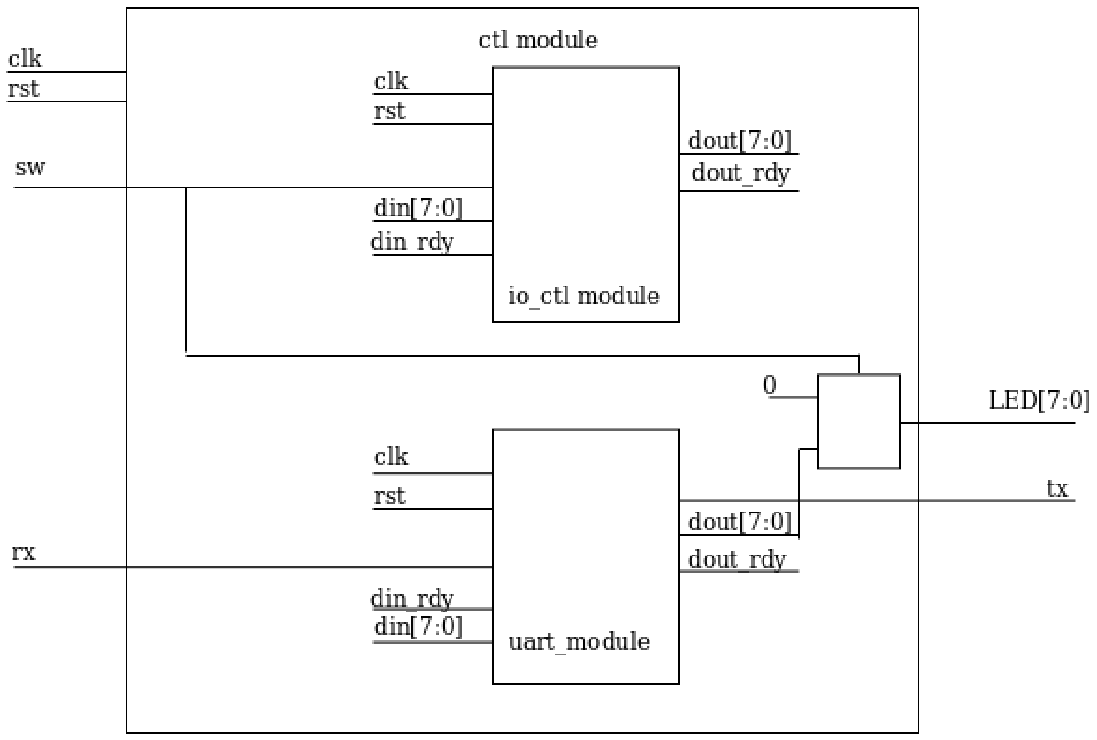
Санкт-Петербург
2016 г.

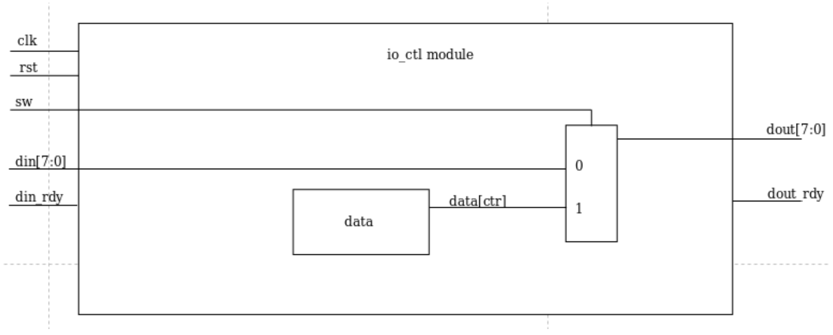
Содержание

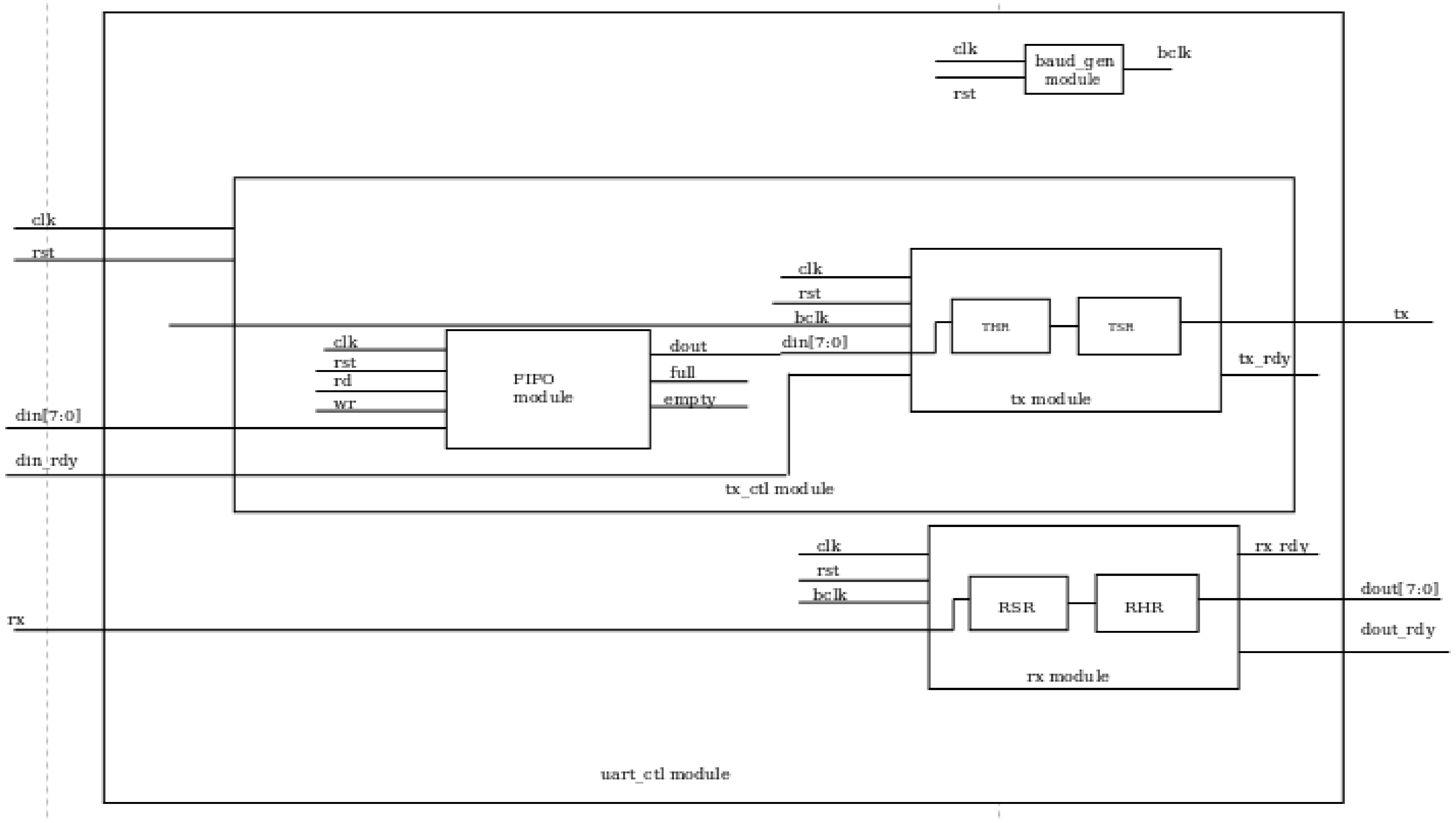
Цели работы

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.

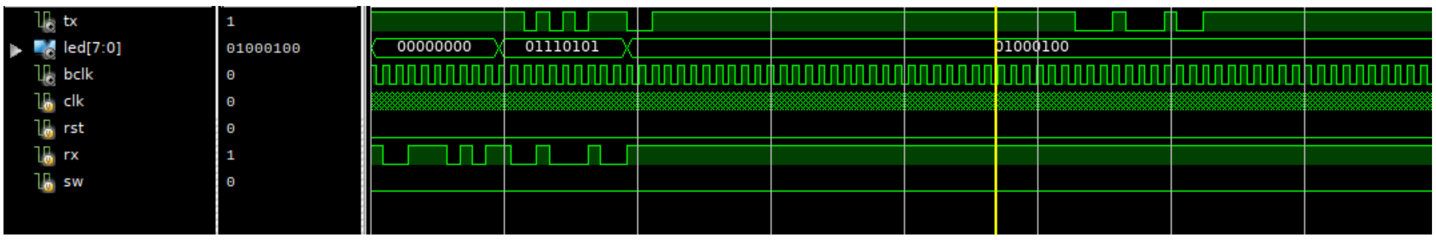
RTL модель

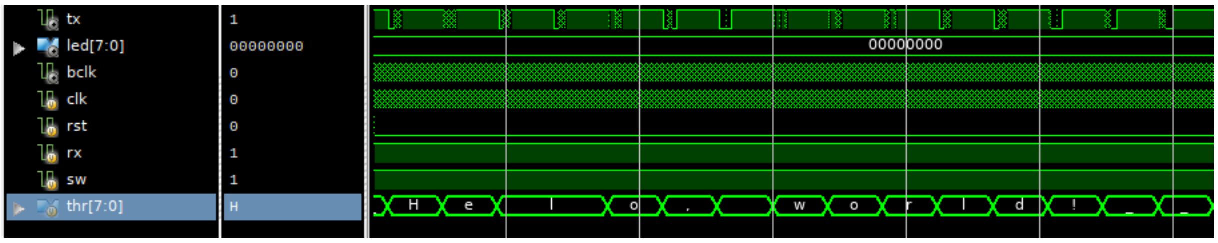





clk
rst
baud_gen module
bclk
clk
rst
clk
rst
bclk
din[7:0]
THR
TSR
tx
tx_rdy
tx module
clk
rst
rd
wr
FIFO module
dout
full
empty
din[7:0]
din_rdy
tx_ctl module
clk
rst
bclk
RSR
RHR
rx_rdy
dout[7:0]
dout_rdy
rx
rx module
uart_ctl module

Временные диаграммы





Листинг

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

    /* UART interface. */
    input           rx,
    output          tx,

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

    wire  [7:0]  dout_uart;
    wire  [7:0]  dout_io;

    wire dout_io_rdy;
    wire dour_uart_rdy;

    io_ctl #( 100000000 )io_ctl(
        .clk(clk),
        .rst(rst),

        .sw(sw),

        .din     (dout_uart), /* dout from uart_ctl to io_ctl din */
        .din_rdy(dout_uart_rdy),

        .dout     (dout_io),
        .dout_rdy(dout_io_rdy) /* dout_io is ready. */
    );

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

        .rx      (rx),
        .din     (dout_io),
        .din_rdy(dout_io_rdy),

        .tx       (tx),
        .dout     (dout_uart),
        .dout_rdy(dout_uart_rdy)
    );

     always @(negedge clk)
         if (rst)
```

```
            led <= 0;
          /* SW == 0: echo mode; SW = 1: send mode. */
          else if ( !sw && dout_uart_rdy)
            led <= dout_uart;


endmodule
```

```
module io_ctl #(
    parameter TIME = 100000000
) (
    input               clk,
    input               rst,
    /* SW = 0: echo-mode. SW = 1: send 'Hello world!\r\n'. */
    input               sw,
    /* Data from uart.*/
    input    [7:0]      din,
    /* Data to in is ready. */
    input               din_rdy,
    /* Allow read from memory in send mode. */
    //input               tx_en,
    /* Data to uart. */
    output   reg[7:0]   dout,
    /* Data to out is ready. */
    output   reg        dout_rdy
);
    localparam ECHO_MODE = 0;
    localparam SEND_MODE = 1;

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


    /* Have no idea what it is. */
    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; /* \r */
            data[14] = 8'h0A; /* \n */
    end

    always @( negedge 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

    always @( posedge clk or posedge rst ) begin
        if( rst ) begin
            was_din_rdy <= 0;
            dout_rdy    <= 0;
            dout        <= 0;
            d_ctr       <= 0;
        end
        else
            case( sw )
                ECHO_MODE:
                begin
                    if( din_rdy && !was_din_rdy ) begin
                        was_din_rdy <= din_rdy;
                        dout        <= din;
                        dout_rdy    <= 1;
                    end
                    else begin
```

```
                        was_din_rdy <= din_rdy;
                        dout_rdy    <= 0;
                    end
                end
                SEND_MODE:
                begin
                    if( tx_flag ) begin
                        dout_rdy <= 1;
                        dout     <= data[d_ctr];
                        d_ctr    <= d_ctr + 1;
                    end
                    else if( d_ctr == 15 ) begin
                        dout_rdy <= 0;
                        d_ctr    <= 0;
                    end
                end
            endcase
    end

endmodule
```

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

    /* I/O buses. */
    input           rx,

    /* Data to send. */
    input    [7:0]  din,
    /* Data to send is ready. */
    input           din_rdy,

    output          tx,
    /* Received data. */
    output   [7:0]  dout,
    /* Receive ends; data ready. */
    output          dout_rdy
    /* Controller ready for transmission. */
//    output          tx_rdy,
    /* Controller ready for reception. */
//    output          rx_rdy
);

    wire    bclk;

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

        .bclk(bclk)
    );

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

        .bclk(bclk),
        .rx  (rx),

        .dout    (dout),
        .dout_rdy(dout_rdy),
        .rx_rdy  (rx_rdy)
    );

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

        .din    (din),
        .din_rdy(din_rdy),

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

endmodule
```

```
module rx(
    input               clk,
    input               rst,
    input               bclk,
    input               rx,

    output  reg[7:0]  dout,
    output  reg       dout_rdy,
    output  reg       rx_rdy
);

    localparam START_BIT = 0;
    localparam STOP_BIT  = 1;

    localparam IDLE      = 2'd0;
    localparam START     = 2'd1;
    localparam STOP      = 2'd2;


    reg [7:0] rsr    = 0;

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

    reg was_bclk  = 0;


    always @(negedge clk or posedge rst) begin
        if (rst)
            state <= IDLE;
        else
            if (!bclk && was_bclk)
                state <= next_state;
    end

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            next_state <= 0;
            was_bclk   <= 0;
            dout_rdy   <= 0;
            rx_rdy     <= 1;
            d_ctr      <= 0;
            rsr        <= 0;
            dout       <= 0;
        end
        else begin
            if (bclk && !was_bclk) begin
                was_bclk <= bclk;
                case (state)
                    IDLE:
                    begin
                        dout_rdy <= 0;
                        if (START_BIT == rx) begin
                            next_state <= START;
                            rx_rdy      <= 0;
                        end
                    end
                    START:
                    begin
                        d_ctr  <= d_ctr + 1'b1;
                        rsr    <= rsr << 1;
                        rsr[0] <= rx;
                        if (3'd7 == d_ctr) begin
                            next_state <= STOP;
                            d_ctr       <= 0;
                        end
                    end
                    STOP:
                    begin
                        if (STOP_BIT == rx) begin
                            dout_rdy <= 1;
                            dout      <= rsr;
                        end
                        next_state <= IDLE;
                        rx_rdy      <= 0;
                    end
                endcase
            end
            else
                was_bclk <= bclk;
        end
    end
endmodule
```

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

    input           bclk,
    input   [7:0]   din,
    input           din_rdy,

    output          tx,
    output          tx_rdy
);

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

        .bclk(bclk),
        /* Data to transmit. */
        .din(fifo_dout),
        /* Data ready to transmit. */
        .din_rdy(en),
        /* TX-pin */
        .tx(tx),
        /* 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( din_rdy && !fifo_full && !rd )
                wr <= 1;
            else
                wr <= 0;

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

```
module tx(
    input           clk,
    input           rst,

    input           bclk,
    input   [7:0]   din,
    input           din_rdy,

    output  reg     tx,
    output  reg     tx_rdy

);

    localparam IDLE     = 3'd0;
    localparam START    = 3'd1;
    localparam TRANSMIT = 3'd2;
    localparam STOP     = 3'd4;
    localparam WAIT     = 3'd5;

    localparam START_BIT = 1'b0;
    localparam STOP_BIT  = 1'b1;

    localparam WAIT_TIME_IN_BAUDS = 30;

    reg [7:0] thr        = 0;
    reg [7:0] tsr        = 0;
    reg [4:0] wait_time  = 0;
    reg [3:0] dctr       = 0;
    reg [2:0] next_state = 0;
    reg [2:0] was_state  = 0;
    reg [2:0] state      = 0;
    reg       was_bclk   = 0;
    reg       tx_en      = 0;
    reg       rdy        = 0;

    always @(negedge clk or posedge rst)
        if (rst)
            state = IDLE;
        else
            if (!bclk && was_bclk)
                state = next_state;

   always @(negedge clk or posedge rst)
       if (rst) begin
            tx_en = 0;
            thr   = 0;
       end
       else begin
           if (din_rdy & !rdy) begin
                tx_rdy = 0;
                thr    = din;
                tx_en  = 1;
                rdy    = 1;
           end
           else
               if (was_state == IDLE && state == START) begin
                    tx_en = 0;
                end
                if (was_state == WAIT && state == IDLE) begin
                    tx_rdy = 1;
                    rdy    = 0;
                end
        end


    always @(posedge clk or posedge rst)
        if (rst) begin
            next_state <= IDLE;
            wait_time  <= 0;
            was_bclk   <= 0;
            dctr       <= 0;
            thr        <= 0;
            tx_rdy     <= 1;
            tx         <= 1;
        end
        else begin
            was_state <= state;
            if (bclk & !was_bclk) begin
                was_bclk  <= bclk;
                case( state )
                    IDLE:
                        if (tx_en) begin
                            next_state <= START;
                            tsr        <= thr;
                        end
                    START:
                    begin
                        next_state <= TRANSMIT;
                        tx         <= START_BIT;
                    end
                    TRANSMIT:
                    begin
                        tx   <= tsr[0];
                        tsr  <= tsr >> 1;
                        dctr <= dctr + 1'b1;
                        if ( 8 == dctr) begin
                            next_state <= STOP;
                            dctr       <= 0;
                        end
                    end
                    STOP:
                    begin
                        next_state <= WAIT;
                        tx         <= STOP_BIT;
                    end
                    /* Need to wait between words sending.
                     * */
                    WAIT:
                    begin
                        wait_time <= wait_time + 1'b1;
                        if (wait_time == WAIT_TIME_IN_BAUDS) begin
                            next_state <= IDLE;
                            wait_time  <= 0;
                        end
                    end
                endcase
            end
            else
                was_bclk <= bclk;
        end

endmodule
```

```
/* 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
```

```
/* Generates bauds. */
/* For 9600 bps 8N1 -- (10417) 9600 bauds. */
module baud_gen #(
    parameter BAUDRATE = 9600
)
(
    input              clk,
    input            rst,

    output   reg     bclk
);
    reg [16:0] ctr  = BAUDRATE;

    always @( posedge clk ) begin
      if( rst )
        ctr <= BAUDRATE;

        if( ctr == BAUDRATE )
            bclk <= 1;
        else if( ctr == BAUDRATE/2 )
            bclk <= 0;

        if( ctr == 0 )
            ctr <= BAUDRATE;
        else
            ctr <= ctr - 16'd1;
    end

endmodule
```

Вывод

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