Университет ИТМО

Кафедра вычислительной техники

Отчёт по лабораторной работе № 3
по дисциплине: "Схемотехника ЭВМ"
Вариант №5

Студенты:
Куклина М.
Кириллова А.

Преподаватель: Баевских А.

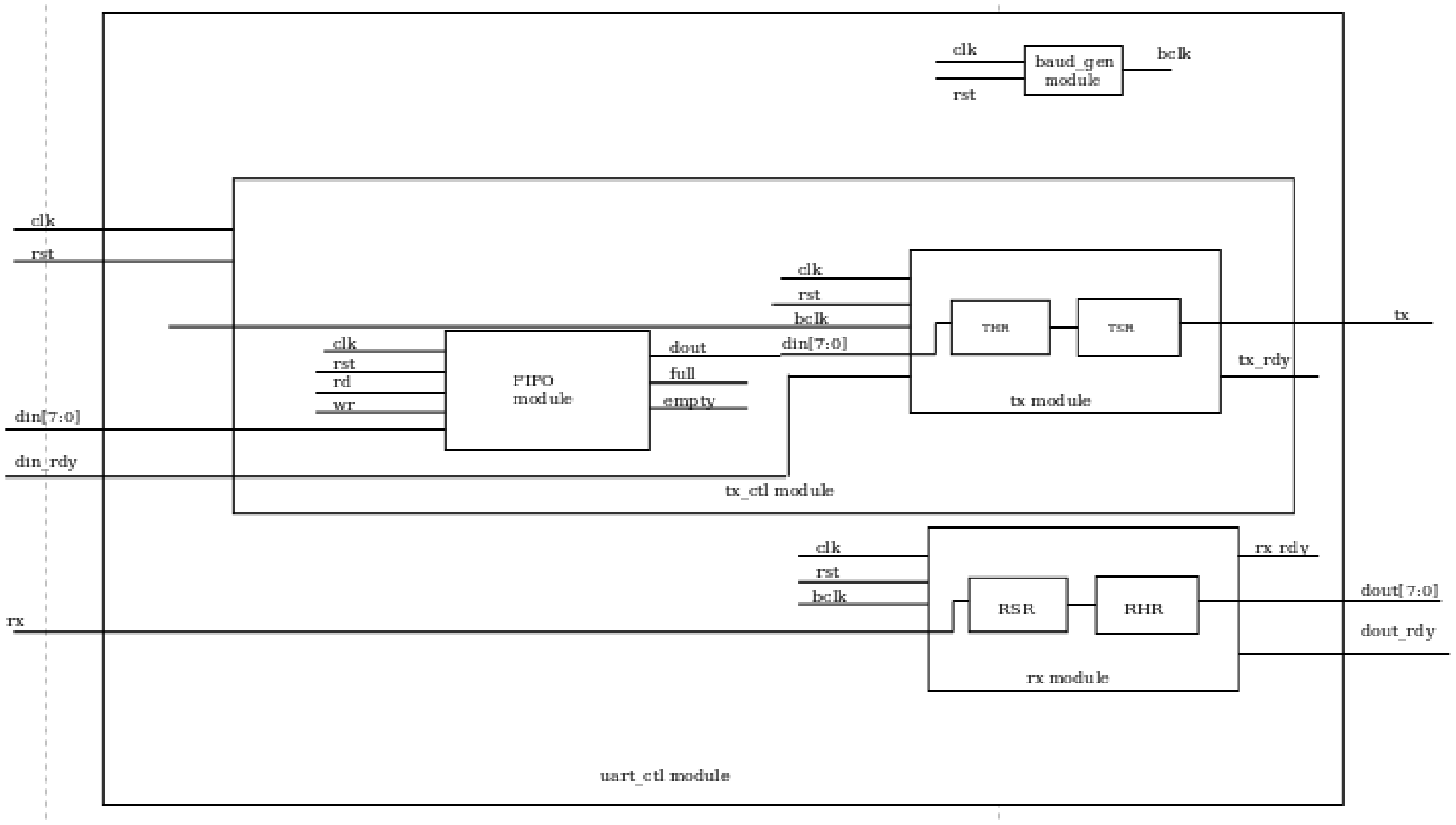
Санкт-Петербург
2016 г.

Содержание

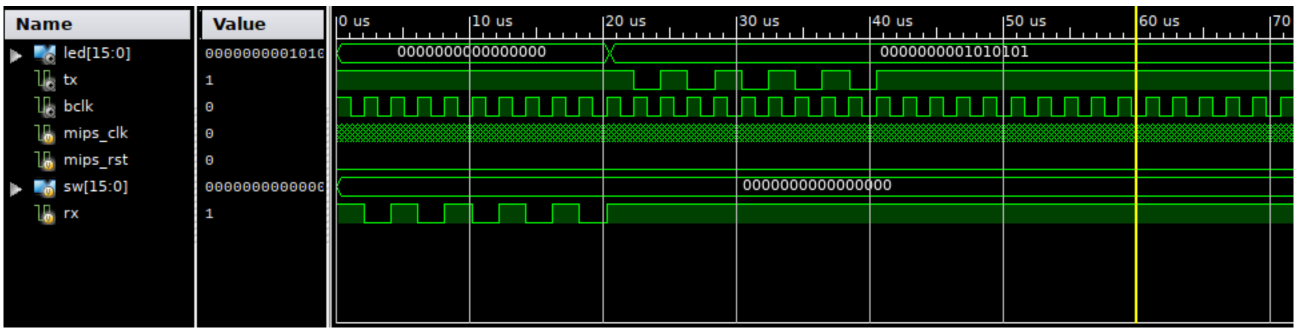
Цели работы

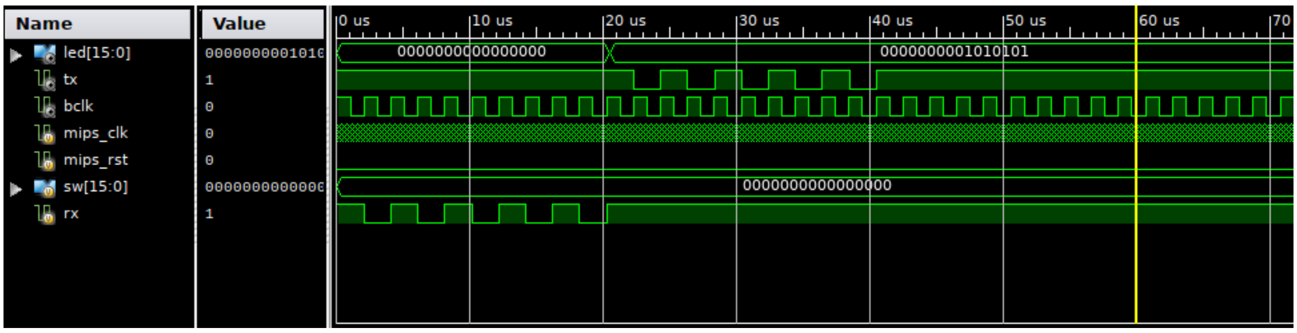
1. Знакомство с шинной организацией вычислительных систем.
2. Знакомство с методами использования адресного пространства в вычислительной системе с шинной организацией.
3. Изучение принципов подключения цифровых блоков в состав вычислительной системы посредством системного интерфейса.

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

```
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
);

        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)
        );

```

```
    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
);

    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;
            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;
                        end
                    end
                    START:
                    begin
                        d_ctr  <= d_ctr + 1'b1;
                        rsr    <= { rx, rsr[7:1] }; //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;
                    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;

    reg [2:0] state = 0;

   localparam IDLE  = 0;
   localparam CHSE  = 1;
   localparam READ  = 2;
   localparam WRITE = 3;
   localparam SEN   = 4;

    always @(posedge clk)
        if (rst) begin
            rd <= 0;
            wr <= 0;
            en <= 0;
            state <= IDLE;
        end else
        case (state)
            IDLE:
            begin
                rd <= 0;
                wr <= 0;
                en <= 0;
                state <= CHSE;
            end
            CHSE:
            begin
                if (tx_rdy && !fifo_empty)
                    state <= READ;
                else if (din_rdy && !fifo_full)
                    state <= WRITE;
            end
            READ:
            begin
                rd <= 1;
                state <= SEN;
            end
            WRITE:
            begin
                wr <= 1;
                state <= IDLE;
            end
            SEN:
            begin
                rd <= 0;
                en <= 1;
                state <= IDLE;
            end
        endcase

    /* Write and read on negedge. */
    fifo tx_fifo(
        .clk(clk),
        .rst(rst),

        .rd (rd),
        .wr (wr),

        .din(din),

        .full (fifo_full),
        .empty(fifo_empty),

        .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 */
        //.din(din),
        //.din_rdy(din_rdy),
        .tx(tx),
        /* Transmitter is ready ( 1 ), is busy ( 0 ). */
        .tx_rdy(tx_rdy)
    );

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'd3;
    localparam WAIT     = 3'd4;

    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 [2:0] dctr        = 0;
    reg [2:0] state       = 0;
    reg       was_bclk    = 0;
    reg       tx_en       = 0;
    reg       was_din_rdy = 0;

    always @(posedge clk or posedge rst)
        if (rst) begin
            state <= IDLE;
            wait_time  <= 0;
            dctr       <= 0;
            tsr        <= 0;
            tx         <= 1;
            tx_rdy     <= 1;
        end
        else begin
            if (IDLE == state && din_rdy) begin
                state <= START;
                tsr        <= din;
                tx_rdy     <= 0;
                was_bclk   <= bclk;
            end
            else
                if (bclk & !was_bclk) begin
                    was_bclk = bclk;
                    case (state)
                        START:
                        begin
                            state <= TRANSMIT;
                            tx          <= START_BIT;
                        end
                        TRANSMIT:
                        begin
                            tx   <= tsr[0];
                            tsr  <= tsr >> 1;
                            dctr <= dctr + 1'b1;
                            if (7 == dctr) begin
                                state <= STOP;
                                dctr       <= 0;
                            end
                        end
                        STOP:
                        begin
                            state <= WAIT;
                            tx          <= STOP_BIT;
                        end
                        WAIT:
                        begin
                            wait_time <= wait_time + 1'b1;
                            if (wait_time == WAIT_TIME_IN_BAUDS) begin
                                state <= IDLE;
                                wait_time  <= 0;
                                tx_rdy     <= 1;
                            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;
```

```verilog
    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
```

```asm
.global entry
.data
    .word   0x1             /* 0x200*/
    .word   0xf4240         /* 0x201*/
    .ascii  "Hello, world!"  /* 0x202*/
    .byte    0x0a
    .byte    0x0d           /* 0x211*/
.text
.ent entry
entry:

    lw $t0, 0x400  /*  load sw */
    lw $t2, 0x200   /* $t2 = 0x1 */
    beq $t0, $t2, SEND_MODE /* if sw[0] == 1 jmp to SEND */

    ECHO_MODE:
        lw $t1, 0x800
        sw $t1, 0x400
        sw $t1, 0x800
        j entry

    SEND_MODE:
        lw $t0, 0x201   /* $t0 = time*/
        andi $t2, 0x0    /* $t2 = 0 */
        dec_loop:
            sub $t0, $t0, 1
            beq $t0, $t2, to_send
            j dec_loop
        to_send:
        lw $t0, 0x202   /* $t0 = ptr to str */
        lw $t2, 0x211  /* $t0 =  ptr to last symbol*/
        send_to_uart_loop:
            lw $t1, ($t0)
            sw $t1, 0x400
            addi $t0, 0x1
            beq $t0, $t2, entry
            j send_to_uart_loop
    j entry
.end entry
```

Вывод

В ходы выполнения лабораторной работы были исследованы принципы работы шины Wishbone, с помощью которого был встроен в микропроцессорную систему MIPS32 контроллер UART.