Университет ИТМО

Кафедра вычислительной техники

Отчёт по лабораторной работе № 2

по дисциплине: "Схемотехника ЭВМ"

Вариант №5

Студенты:
Куклина М.
Кириллова А.

Преподаватель: Баевских А.

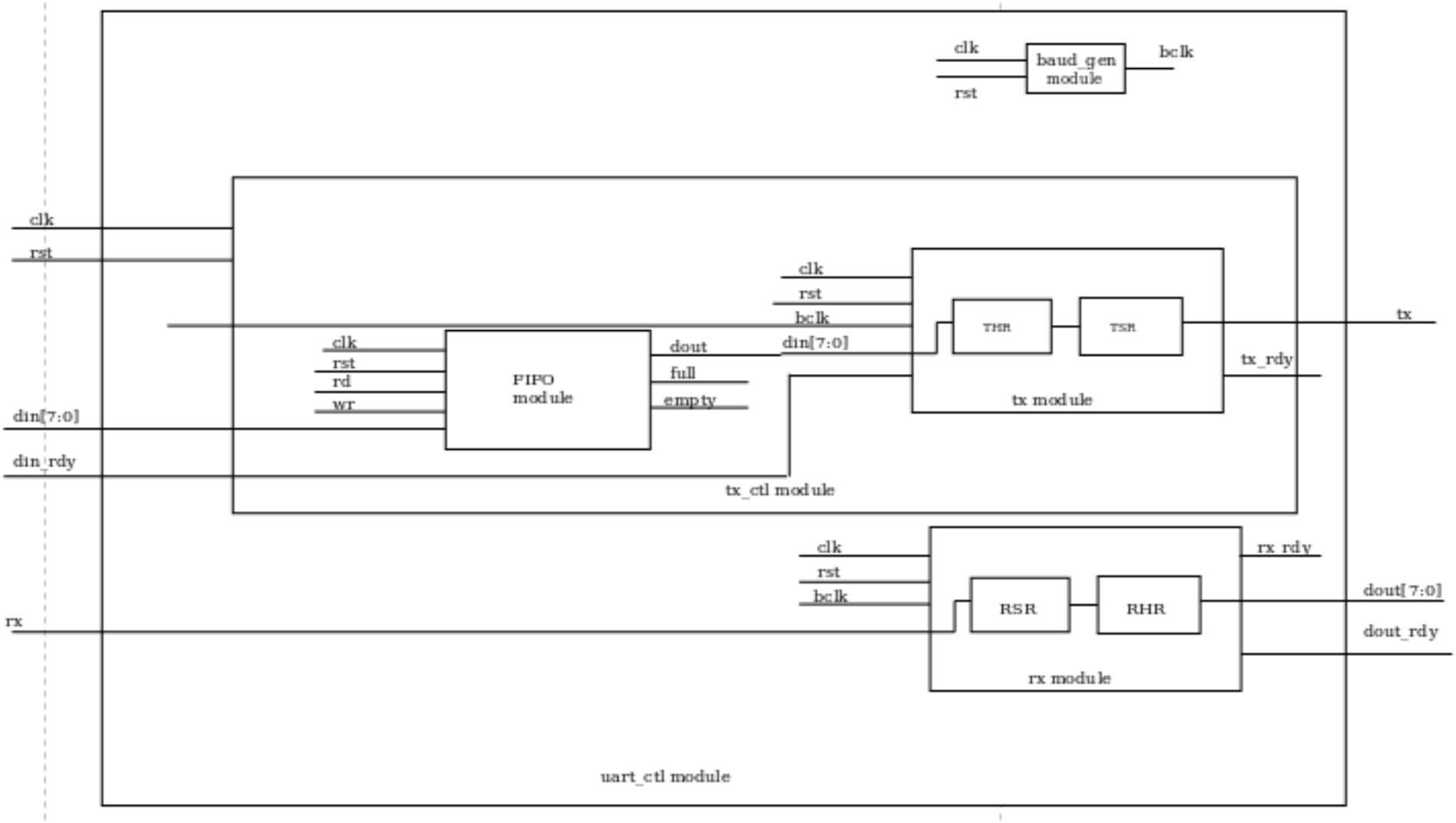
Санкт-Петербург
2016 г.

Содержание

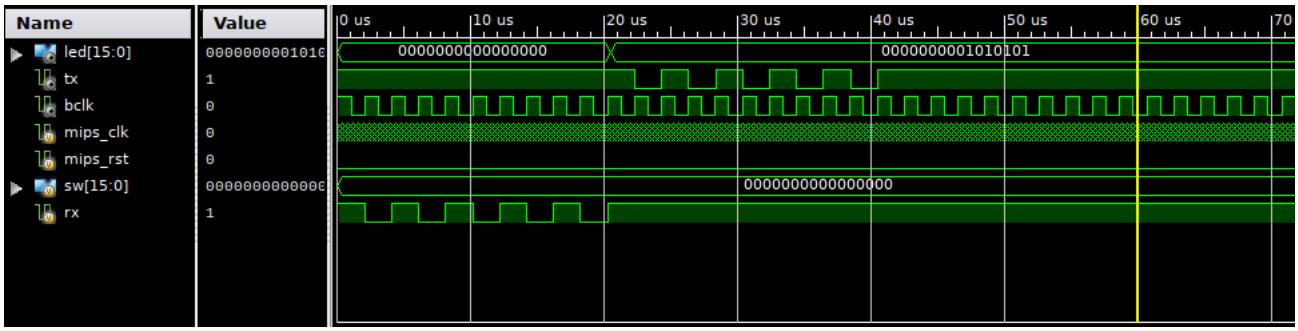
Цели работы

1. Знакомство с 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
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  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 );
21
22     wire     bclk;
23
24     baud_gen baud_gen(
25         .clk(clk),
26         .rst(rst),
27
28         .bclk(bclk)
29     );
30
31     rx_ctl rx_ctl(
32         .clk (clk),
33         .rst (rst),
34
35         .bclk(bclk),
36         .rx  (rx),
37
38         .dout    (dout),
39         .dout_rdy(dout_rdy)
40     );
41
42     tx_ctl tx_ctl(
43         .clk (clk),
44         .rst (rst),
45         .bclk(bclk),
46
47         .din    (din),
48         .din_rdy(din_rdy),
49
50         .tx    (tx),
51         .tx_rdy(tx_rdy)
52     );
53
54 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
```

```
79      end
80  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      wire [7:0] fifo_dout;
17
18      /*
19      always @(negedge clk) begin
20          if (rst) begin
21              wr = 0;
22          end
23          else begin
24              wr = tx_rdy;
25              rd = din_rdy;
26          end
27      end
28      always @(posedge clk) begin
29          if (rst)
30              en = 0;
31          else
32              en = rd;
33      end
34      */
35
36
37      reg [1:0]state = 0;
38
39     localparam IDLE = 0;
40     localparam READ = 1;
41     localparam WRITE = 2;
42     localparam SEND_EN = 3;
43
44  always @(posedge clk)
45      if (rst) begin
46          rd <= 0;
47          wr <= 0;
48          en <= 0;
49          state <= IDLE;
50      end else
51      case (state)
52          IDLE:
53          begin
54              rd <= 0;
55              wr <= 0;
56              en <= 0;
57              if (tx_rdy && !fifo_empty)
58                  state <= READ;
59              else if (din_rdy && !fifo_full)
60                  state <= WRITE;
61          end
62          READ:
63          begin
64              rd <= 1;
65              state <= SEND_EN;
66          end
67          WRITE:
68          begin
69              wr <= 1;
70              state <= IDLE;
71          end
72          SEND_EN:
73          begin
74              rd <= 0;
75              en <= 1;
```

```
76              state <= IDLE;
77          end
78      endcase
79
80      /* Write and read on negedge. */
81      fifo tx_fifo(
82          .clk(clk),
83          .rst(rst),
84
85          .rd (rd),
86          .wr (wr),
87
88          .din(din),
89
90          .full (fifo_full),
91          .empty(fifo_empty),
92
93          .dout (fifo_dout)
94      );
95
96      tx tx_mod(
97          .clk(clk),
98          .rst(rst),
99
100         .bclk(bclk),
101         /* Data to transmit. */
102         .din(fifo_dout),
103         /* Data ready to transmit. */
104         .din_rdy(en),
105         /* TX-pin */
106         //.din(din),
107         //.din_rdy(din_rdy),
108         .tx(tx),
109         /* Transmitter is ready ( 1 ), is busy ( 0 ). */
110         .tx_rdy(tx_rdy)
111     );
112
113 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     localparam IDLE      = 3'd0;
14     localparam START     = 3'd1;
15     localparam TRANSMIT  = 3'd2;
16     localparam STOP      = 3'd3;
17     localparam WAIT      = 3'd4;
18
19     localparam START_BIT = 1'b0;
20     localparam STOP_BIT  = 1'b1;
21
22     localparam WAIT_TIME_IN_BAUDS = 30;
23
24     reg [7:0] thr         = 0;
25     reg [7:0] tsr         = 0;
26     reg [4:0] wait_time   = 0;
27     reg [2:0] dctr        = 0;
28     reg [2:0] state       = 0;
29     reg       was_bclk    = 0;
30     reg       tx_en       = 0;
31     reg       was_din_rdy = 0;
32
33             /*
34     always @(negedge clk or posedge rst)
35         if (rst) begin
36             tx_en = 0;
37             thr   = 0;
38                 tx_rdy = 1;
39                 was_din_rdy = din_rdy;
```

```
40          end
41          else begin
42              if (din_rdy & !was_din_rdy) begin
43                  tx_rdy = 0;
44                  thr    = din;
45                  tx_en  = 1;
46                  was_din_rdy = din_rdy;
47              end
48              else begin
49               if (was_state == IDLE && state == START) begin
50                      tx_en = 0;
51               end
52               if (state == IDLE)
53                  tx_rdy = 1;
54                  was_din_rdy = din_rdy;
55               end
56          end
57           */
58
59      always @(posedge clk or posedge rst)
60           if (rst) begin
61               state <= IDLE;
62               wait_time  <= 0;
63               dctr       <= 0;
64               tsr        <= 0;
65               tx         <= 1;
66               tx_rdy     <= 1;
67           end
68           else begin
69               if (IDLE == state && din_rdy) begin
70                   state <= START;
71                   tsr        <= din;
72                   tx_rdy     <= 0;
73                   was_bclk   <= bclk;
74               end
75               else
76                   if (bclk & !was_bclk) begin
77                       was_bclk = bclk;
78                       case (state)
79                           START:
80                           begin
81                               state <= TRANSMIT;
82                               tx         <= START_BIT;
83                           end
84                           TRANSMIT:
85                           begin
86                               tx   <= tsr[0];
87                               tsr  <= tsr >> 1;
88                               dctr <= dctr + 1'b1;
89                               if (7 == dctr) begin
90                                   state <= STOP;
91                                   dctr       <= 0;
92                               end
93                           end
94                           STOP:
95                           begin
96                               state <= WAIT;
97                               tx         <= STOP_BIT;
98                           end
99                           WAIT:
100                          begin
101                              wait_time <= wait_time + 1'b1;
102                              if (wait_time == WAIT_TIME_IN_BAUDS) begin
103                                  state <= IDLE;
104                                  wait_time  <= 0;
105                                  tx_rdy     <= 1;
106                              end
107                          end
108                      endcase
109                  end
110                  else
111                      was_bclk <= bclk;
112          end
113
114 endmodule
```

```
1 /* Read and write on posedge clk.
2  * 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
```

```
81
82 endmodule
```

```
1 .global entry
2 .data
3     .word   0x1             /* 0x200*/
4     .word   0xf4240         /* 0x201*/
5     .ascii  "Hello, world!"  /* 0x202*/
6     .byte    0x0a
7     .byte    0x0d           /* 0x211*/
8 .text
9 .ent entry
10 entry:
11
12     lw $t0, 0x400  /*  load sw */
13     lw $t2, 0x200   /* $t2 = 0x1 */
14     beq $t0, $t2, SEND_MODE /* if sw[0] == 1 jmp to SEND */
15
16     ECHO_MODE:
17         lw $t1, 0x800
18         sw $t1, 0x400
19         sw $t1, 0x800
20         j entry
21
22     SEND_MODE:
23         lw $t0, 0x201    /* $t0 = time*/
24         andi $t2, 0x0     /* $t2 = 0 */
25         dec_loop:
26             sub $t0, $t0, 1
27             beq $t0, $t2, to_send
28             j dec_loop
29         to_send:
30         lw $t0, 0x202    /* $t0 = ptr to str */
31         lw $t2, 0x211  /* $t0 =  ptr to last symbol*/
32         send_to_uart_loop:
33             lw $t1, ($t0)
34             sw $t1, 0x400
35             addi $t0, 0x1
36             beq $t0, $t2, entry
37             j send_to_uart_loop
38     j entry
39 .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.