中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：_综合实验-“猜学号”shell _

学生姓名：___宋玮_____

学生学号：_PB20151793_

完成日期：__2021.12.22_

计算机实验教学中心制

【实验题目】

综合实验-“猜学号”shell

【实验目的】

熟练掌握前面实验中的所有知识点
熟悉几种常用通信接口的工作原理及使用
独立完成具有一定规模的功能电路设计

【实验环境】

VLAB：vlab.ustc.edu.cn
FPGAOL：fpgaol.ustc.edu.cn
Vivado

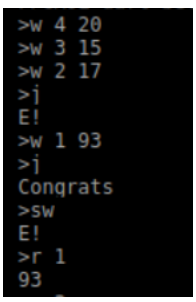
【实验过程】

实验概述： 首先，在 FPGAOL 平台上，利用串口终端等外设，实现一个可以简单读写的 shell。然后，在此功能之上，再加入判断（judge）功能，即判断写入的八位是否为本人学号（20151793），若是，则输出 congrats，否则输出 E!

注意，在输入命令时须符合如下规范，否则，也将输出 E!

命令	格式	范围	示例	说明
Write	w addr data	1<=addr<=4 00<data<ff	>w 1 ff	向地址 1 写入 ff
Read	r addr	1<=addr<=4	>r 1	读地址 1 中的数据并将其打印
Judge	j		>j	判断写入的是否为本人的八位学号
Else				输出为 E!

地址 1：数码管 1~0 位所表示的字节数据
地址 2：数码管 3~2 位所表示的字节数据
地址 3：数码管 5~4 位所表示的字节数据
地址 4：数码管 7~6 位所表示的字节数据
七段数码管(led[7:0])在写入的八位为本人学号时显示 1，其他时候均显示 0
示例：



step1 程序

（1）首先是主体部分。

总共包含 8 个状态：

C_CMD_CO　　判断正确阶段，即发送"Congrats"
C_IDLE　　空闲状态，接收到串口数据后跳转到下一状态 C_CMD_DC
C_CMD_DC　　解码命令状态，解析缓冲区中的命令类型，如写字节、读字节等
C_CMD_WB　　写数据阶段，根据命令内容，将数据写入对应地址
C_CMD_RB　　读数据阶段，根据命令内容，读取对应地址的数据
C_CMD_ERR　　错误状态，上位机发送的命令格式有误时进入此状态，向上位机发送"E！"
C_TXFIFO_WR　　等待阶段，将数据或返回信息以 ASCII 码格式存入发送缓冲区
C_TXFIFO_WAIT 发送等待阶段，将发送缓冲区中的数据依次以 ASCII 码格式从串口发出

根据不同的情况，状态在这八个当中跳转，并在特定状态完成相应的特定操作。

主体 design 代码如下：

```
module lab10(
input               clk,rst,
input               rx,
output              tx,
output reg  [7:0]   led,
output wire [2:0]   hex_seg_an,
output reg  [3:0]   hex_seg_data
);
wire                tx_ready;
wire        [7:0]   tx_data;
wire        [7:0]   rx_data;

reg         [3:0]   curr_state;
reg         [3:0]   next_state;

wire                is_wb_cmd;
wire                is_rb_cmd;
wire                is_judge;

reg         [3:0]   tx_byte_cnt;
reg         [3:0]   rx_byte_cnt;

reg         [7:0]   rx_byte_buff_0;
reg         [7:0]   rx_byte_buff_1;
reg         [7:0]   rx_byte_buff_2;
reg         [7:0]   rx_byte_buff_3;
reg         [7:0]   rx_byte_buff_4;
reg         [7:0]   rx_byte_buff_5;
```

```
reg         [7:0]   rx_byte_buff_6;
reg         [7:0]   rx_byte_buff_7;

reg         [7:0]   tx_byte_buff_0;
reg         [7:0]   tx_byte_buff_1;
reg         [7:0]   tx_byte_buff_2;
reg         [7:0]   tx_byte_buff_3;
reg         [7:0]   tx_byte_buff_4;
reg         [7:0]   tx_byte_buff_5;
reg         [7:0]   tx_byte_buff_6;
reg         [7:0]   tx_byte_buff_7;
reg         [7:0]   tx_byte_buff_8;

reg                 rx_fifo_en;
wire    [7:0]       rx_fifo_data;
wire                rx_fifo_empty;

reg                 wr_en;
reg     [3:0]       wr_addr;
reg     [7:0]       wr_data;

reg                 rd_en;
reg     [3:0]       rd_addr;
reg     [7:0]       rd_data;

reg     [7:0]       tx_fifo_din;
reg                 tx_fifo_wr_en;
wire                tx_fifo_full;
wire                tx_fifo_empty;

reg    [18:0]  hex_seg_scan;
reg    [31:0]  hex_seg_buff;

parameter   C_IDLE          = 4'b0000;
parameter   C_CMD_DC        = 4'b0010;
parameter   C_CMD_WB        = 4'b0011;
parameter   C_CMD_RB        = 4'b0100;
parameter   C_CMD_ERR       = 4'b0111;
parameter   C_TXFIFO_WR     = 4'b0101;
parameter   C_TXFIFO_WAIT   = 4'b0110;
parameter   C_CMD_CO        = 4'b1000;

always@(posedge clk or posedge rst)
begin
```

```
    if(rst)
    begin
        tx_fifo_wr_en   <= 1'b0;
        tx_fifo_din     <= 8'h0;
    end
    else if(curr_state==C_TXFIFO_WR)
    begin
        tx_fifo_wr_en   <= 1'b1;
        case(tx_byte_cnt)
            4'h8:   tx_fifo_din <= tx_byte_buff_8;
            4'h7:   tx_fifo_din <= tx_byte_buff_7;
            4'h6:   tx_fifo_din <= tx_byte_buff_6;
            4'h5:   tx_fifo_din <= tx_byte_buff_5;
            4'h4:   tx_fifo_din <= tx_byte_buff_4;
            4'h3:   tx_fifo_din <= tx_byte_buff_3;
            4'h2:   tx_fifo_din <= tx_byte_buff_2;
            4'h1:   tx_fifo_din <= tx_byte_buff_1;
            4'h0:   tx_fifo_din <= tx_byte_buff_0;
            default:tx_fifo_din <= 8'h0;
        endcase
    end
    else
    begin
        tx_fifo_wr_en   <= 1'b0;
        tx_fifo_din     <= 8'h0;
    end
end

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        rd_en        <= 1'b0;
        rd_addr[3:0] <= 4'h0;
    end
    else if(curr_state==C_CMD_RB)
    begin
        rd_en   <= 1'b1;
        if((rx_byte_buff_2>="0")&&(rx_byte_buff_2<="9"))
            rd_addr[3:0] <= rx_byte_buff_2[3:0];
        else
            rd_addr[3:0] <= rx_byte_buff_2[3:0]+ 4'h9;
    end
    else
```

```
    begin
        rd_en   <= 1'b0;
        rd_addr[3:0] <= 4'h0;
    end
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_byte_cnt <= 4'h0;
    else if(curr_state==C_IDLE)
        tx_byte_cnt <= 4'h0;
    else if(curr_state==C_CMD_RB)
        tx_byte_cnt <= 4'h2;
    else if(curr_state==C_CMD_ERR)
        tx_byte_cnt <= 4'h2;
    else if(curr_state==C_CMD_CO)
        tx_byte_cnt <= 4'h8;
    else if(curr_state==C_TXFIFO_WR)
    begin
        if(tx_byte_cnt!=4'h0)
            tx_byte_cnt <= tx_byte_cnt - 4'h1;
    end
end

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        tx_byte_buff_0  <= 8'h0;
        tx_byte_buff_1  <= 8'h0;
        tx_byte_buff_2  <= 8'h0;
        tx_byte_buff_3  <= 8'h0;
        tx_byte_buff_4  <= 8'h0;
        tx_byte_buff_5  <= 8'h0;
        tx_byte_buff_6  <= 8'h0;
        tx_byte_buff_7  <= 8'h0;
        tx_byte_buff_8  <= 8'h0;
    end
    else if(curr_state==C_IDLE)
    begin
        tx_byte_buff_0  <= 8'h0;
        tx_byte_buff_1  <= 8'h0;
        tx_byte_buff_2  <= 8'h0;
        tx_byte_buff_3  <= 8'h0;
```

```
        tx_byte_buff_4  <= 8'h0;
        tx_byte_buff_5  <= 8'h0;
        tx_byte_buff_6  <= 8'h0;
        tx_byte_buff_7  <= 8'h0;
         tx_byte_buff_8  <= 8'h0;
    end
    else if(curr_state==C_CMD_RB)
    begin
        tx_byte_buff_0  <= "\n";
        if(rd_data[7:4]<=4'h9)//0~9
            tx_byte_buff_2  <= {4'h3,rd_data[7:4]};
        else
            tx_byte_buff_2  <= rd_data[7:4] - 4'ha + "a";
        if(rd_data[3:0]<=4'h9)//0~9
            tx_byte_buff_1  <= {4'h3,rd_data[3:0]};
        else
            tx_byte_buff_1  <= rd_data[3:0] - 4'ha + "a";
    end
    else if(curr_state==C_CMD_ERR)
    begin
        tx_byte_buff_2  <= "E";
        tx_byte_buff_1  <= "!";
        tx_byte_buff_0  <= "\n";
    end
    else if(curr_state==C_CMD_CO)
        begin
            tx_byte_buff_8  <= "C";
            tx_byte_buff_7  <= "o";
            tx_byte_buff_6  <= "n";
            tx_byte_buff_5  <= "g";
            tx_byte_buff_4  <= "r";
            tx_byte_buff_3  <= "a";
            tx_byte_buff_2  <= "t";
            tx_byte_buff_1  <= "s";
            tx_byte_buff_0  <= "\n";
        end
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <= C_IDLE;
    else
        curr_state  <= next_state;
```

```
end
always@(*)
begin
    case(curr_state)
        C_IDLE:
            if((rx_vld==1'b1)&&(rx_data==8'h0a))
                next_state  = C_CMD_DC;
            else
                next_state  = C_IDLE;
        C_CMD_DC:
            if(rx_fifo_empty)
            begin
                if(is_wb_cmd)
                    next_state  = C_CMD_WB;
                else if(is_rb_cmd)
                    next_state  = C_CMD_RB;
                else if(is_judge)
                    next_state  = C_CMD_CO;
                else
                    next_state  = C_CMD_ERR;
            end
            else
                next_state  = C_CMD_DC;
        C_CMD_WB:
            next_state  = C_IDLE;
        C_CMD_RB:
            if(rd_en==1'b1)
                next_state  = C_TXFIFO_WR;
            else
                next_state  = C_CMD_RB;
        C_CMD_ERR:
            next_state  = C_TXFIFO_WR;
        C_CMD_CO:
            next_state  = C_TXFIFO_WR;
        C_TXFIFO_WR:
            if(tx_byte_cnt==4'h0)
                next_state = C_TXFIFO_WAIT;
            else
                next_state = C_TXFIFO_WR;
        C_TXFIFO_WAIT:
            if(tx_fifo_empty)
                next_state = C_IDLE;
            else
                next_state = C_TXFIFO_WAIT;
```

```
        default:
                next_state = C_IDLE;
    endcase
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        rx_fifo_en  <= 1'b0;
    else if(curr_state==C_CMD_DC)
        rx_fifo_en  <= 1'b1;
    else
        rx_fifo_en  <= 1'b0;
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        rx_byte_cnt <= 4'h0;
    else if(curr_state==C_CMD_DC)
    begin
        if((rx_fifo_en)&&(rx_fifo_empty==1'b0)&&(rx_byte_cnt<4'hf))
            rx_byte_cnt <= rx_byte_cnt + 4'b1;
    end
    else
        rx_byte_cnt <= 4'h0;
end

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        rx_byte_buff_0  <= 8'h0;
        rx_byte_buff_1  <= 8'h0;
        rx_byte_buff_2  <= 8'h0;
        rx_byte_buff_3  <= 8'h0;
        rx_byte_buff_4  <= 8'h0;
        rx_byte_buff_5  <= 8'h0;
        rx_byte_buff_6  <= 8'h0;
        rx_byte_buff_7  <= 8'h0;
    end
    else if(curr_state==C_IDLE)
    begin
        rx_byte_buff_0  <= 8'h0;
```

```
        rx_byte_buff_1  <= 8'h0;
        rx_byte_buff_2  <= 8'h0;
        rx_byte_buff_3  <= 8'h0;
        rx_byte_buff_4  <= 8'h0;
        rx_byte_buff_5  <= 8'h0;
        rx_byte_buff_6  <= 8'h0;
        rx_byte_buff_7  <= 8'h0;
    end
    else if(curr_state==C_CMD_DC)
    begin
        case(rx_byte_cnt)
            4'h0:   rx_byte_buff_0 <= rx_fifo_data;
            4'h1:   rx_byte_buff_1 <= rx_fifo_data;
            4'h2:   rx_byte_buff_2 <= rx_fifo_data;
            4'h3:   rx_byte_buff_3 <= rx_fifo_data;
            4'h4:   rx_byte_buff_4 <= rx_fifo_data;
            4'h5:   rx_byte_buff_5 <= rx_fifo_data;
            4'h6:   rx_byte_buff_6 <= rx_fifo_data;
            4'h7:   rx_byte_buff_7 <= rx_fifo_data;
        endcase
    end
end

assign  is_wb_cmd = (curr_state==C_CMD_DC)
                  &&(rx_byte_buff_0=="w")
                  &&(rx_byte_buff_1==" ")&&(rx_byte_buff_3==" ")
&&(((rx_byte_buff_2>="0")&&(rx_byte_buff_2<="9"))||((rx_byte_buff_2>="a")&&(rx_byte_buff_2<="f")))

&&(((rx_byte_buff_4>="0")&&(rx_byte_buff_4<="9"))||((rx_byte_buff_4>="a")&&(rx_byte_buff_4<="f")))

&&(((rx_byte_buff_5>="0")&&(rx_byte_buff_5<="9"))||((rx_byte_buff_5>="a")&&(rx_byte_buff_5<="f")));

assign  is_rb_cmd = (curr_state==C_CMD_DC)
                  &&(rx_byte_buff_0=="r")
                  &&(rx_byte_buff_1==" ")
&&(((rx_byte_buff_2>="0")&&(rx_byte_buff_2<="9"))||((rx_byte_buff_2>="a")&&(rx_byte_buff_2<="f")));

assign is_judge = (curr_state==C_CMD_DC)
                   &&(rx_byte_buff_0=="j")
                   &&(hex_seg_buff==32'h20151793);

always@(posedge clk or posedge rst)
begin
    if(rst)
```

```
    begin
        wr_en   <= 1'b0;
        wr_addr[3:0] <= 4'h0;
        wr_data[7:4] <= 4'h0;
        wr_data[3:0] <= 4'h0;
    end
    else if(curr_state == C_CMD_WB)
    begin
        wr_en   <= 1'b1;
        if((rx_byte_buff_2>="0")&&(rx_byte_buff_2<="9"))
            wr_addr[3:0] <= rx_byte_buff_2[3:0];
        else
            wr_addr[3:0] <= rx_byte_buff_2[2:0] + 4'h9;
        if((rx_byte_buff_4>="0")&&(rx_byte_buff_4<="9"))
            wr_data[7:4] <= rx_byte_buff_4[3:0];
        else
            wr_data[7:4] <= rx_byte_buff_4[2:0] + 4'h9;
        if((rx_byte_buff_5>="0")&&(rx_byte_buff_5<="9"))
            wr_data[3:0] <= rx_byte_buff_5[3:0];
        else
            wr_data[3:0] <= rx_byte_buff_5[2:0] + 4'h9;
    end
    else
    begin
        wr_en   <= 1'b0;
        wr_addr[3:0] <= 4'h0;
        wr_data[7:4] <= 4'h0;
        wr_data[3:0] <= 4'h0;
    end
end

rx                  rx_inst(
.clk                (clk),
.rst                (rst),
.rx                 (rx),
.rx_vld             (rx_vld),
.rx_data            (rx_data)
);

tx                  tx_inst(
.clk                (clk),
.rst                (rst),
.tx                 (tx ),
.tx_ready           (~tx_fifo_empty),
```

```
.tx_rd              (tx_rd),
.tx_data            (tx_data)
);

fifo_32x8bit_0      rx_fifo(
.clk                (clk),
.srst                (rst),
.din                (rx_data),
.wr_en              (rx_vld),
.rd_en              (rx_fifo_en),
.dout               (rx_fifo_data),
.full               (),
.empty              (rx_fifo_empty)
);

fifo_32x8bit_0      tx_fifo(
.clk                (clk),
.srst                (rst),
.din                (tx_fifo_din),
.wr_en              (tx_fifo_wr_en),
.rd_en              (tx_rd),
.dout               (tx_data),
.full               (tx_fifo_full),
.empty              (tx_fifo_empty)
);

always@(posedge clk or posedge rst)
begin
    if(rst)
        hex_seg_scan <= 19'h0;
    else
        hex_seg_scan <= hex_seg_scan + 1'b1;
end
assign hex_seg_an = hex_seg_scan[18:16];
always@(*)
begin
    case(hex_seg_an)
        3'h0: hex_seg_data = hex_seg_buff[3:0];
        3'h1: hex_seg_data = hex_seg_buff[7:4];
        3'h2: hex_seg_data = hex_seg_buff[11:8];
        3'h3: hex_seg_data = hex_seg_buff[15:12];
        3'h4: hex_seg_data = hex_seg_buff[19:16];
        3'h5: hex_seg_data = hex_seg_buff[23:20];
        3'h6: hex_seg_data = hex_seg_buff[27:24];
```

```
        3'h7: hex_seg_data = hex_seg_buff[31:28];
    endcase
end

always@(*)
begin
    if(rd_en)
    begin
        case(rd_addr)
            4'h1:  rd_data = hex_seg_buff[7:0];
            4'h2:  rd_data = hex_seg_buff[15:8];
            4'h3:  rd_data = hex_seg_buff[23:16];
            4'h4:  rd_data = hex_seg_buff[31:24];
            default:rd_data = 4'h0;
        endcase
    end
    else
        rd_data = 8'h0;
end

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        hex_seg_buff    <= 32'h0;
    end
    else if(wr_en)
    begin
        case(wr_addr)
            4'h1:  hex_seg_buff[7:0]   <= wr_data;
            4'h2:  hex_seg_buff[15:8]  <= wr_data;
            4'h3:  hex_seg_buff[23:16] <= wr_data;
            4'h4:  hex_seg_buff[31:24] <= wr_data;
        endcase
    end
end

always@(*)
begin
    if(hex_seg_buff==32'h20151793) led<=8'h06;
    else led<=8'h3f;
end
endmodule
```

（2）其他部分。

①rx

代码如下：

```
module rx(
    input               clk,rst,
    input               rx,
    output  reg         rx_vld,
    output  reg  [7:0]  rx_data
    );
parameter   DIV_CNT   = 10'd867;
parameter   HDIV_CNT  = 10'd433;
parameter   RX_CNT    = 4'h8;
parameter   C_IDLE    = 1'b0;
parameter   C_RX      = 1'b1;
reg         curr_state;
reg         next_state;
reg [9:0]   div_cnt;
reg [3:0]   rx_cnt;
reg         rx_reg_0,rx_reg_1,rx_reg_2,rx_reg_3,rx_reg_4,rx_reg_5,rx_reg_6,rx_reg_7;
wire        rx_pulse;
always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <= C_IDLE;
    else
        curr_state  <= next_state;
end
always@(*)
begin
    case(curr_state)
        C_IDLE:
            if(div_cnt==HDIV_CNT)
                next_state  = C_RX;
            else
                next_state  = C_IDLE;
        C_RX:
            if((div_cnt==DIV_CNT)&&(rx_cnt>=RX_CNT))
                next_state  = C_IDLE;
            else
                next_state  = C_RX;
    endcase
end
```

```
always@(posedge clk or posedge rst)
begin
    if(rst)
        div_cnt <= 10'h0;
    else if(curr_state == C_IDLE)
    begin
        if(rx==1'b1)
            div_cnt <= 10'h0;
        else if(div_cnt < HDIV_CNT)
            div_cnt <= div_cnt + 10'h1;
        else
            div_cnt <= 10'h0;
    end
    else if(curr_state == C_RX)
    begin
        if(div_cnt >= DIV_CNT)
            div_cnt <= 10'h0;
        else
            div_cnt <= div_cnt + 10'h1;
    end
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        rx_cnt  <= 4'h0;
    else if(curr_state == C_IDLE)
        rx_cnt  <= 4'h0;
    else if((div_cnt == DIV_CNT)&&(rx_cnt<4'hF))
        rx_cnt  <= rx_cnt + 1'b1;
end
assign rx_pulse = (curr_state==C_RX)&&(div_cnt==DIV_CNT);
always@(posedge clk)
begin
    if(rx_pulse)
    begin
        case(rx_cnt)
            4'h0: rx_reg_0 <= rx;
            4'h1: rx_reg_1 <= rx;
            4'h2: rx_reg_2 <= rx;
            4'h3: rx_reg_3 <= rx;
            4'h4: rx_reg_4 <= rx;
            4'h5: rx_reg_5 <= rx;
            4'h6: rx_reg_6 <= rx;
            4'h7: rx_reg_7 <= rx;
```

```
        endcase
    end
end
always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        rx_vld  <= 1'b0;
        rx_data <= 8'h55;
    end
    else if((curr_state==C_RX)&&(next_state==C_IDLE))
    begin
        rx_vld  <= 1'b1;
        rx_data <=
{rx_reg_7,rx_reg_6,rx_reg_5,rx_reg_4,rx_reg_3,rx_reg_2,rx_reg_1,rx_reg_0};
    end
    else
        rx_vld  <= 1'b0;
end
endmodule
```

②tx

代码如下：

```
module tx(
    input           clk,rst,
    output  reg     tx,
    input           tx_ready,
    output  reg     tx_rd,
    input   [7:0]   tx_data
);
parameter   DIV_CNT   = 10'd867;
parameter   HDIV_CNT  = 10'd433;
parameter   TX_CNT    = 4'h9;
parameter   C_IDLE    = 1'b0;
parameter   C_TX      = 1'b1;

reg         curr_state,next_state;
reg [9:0]   div_cnt;
reg [4:0]   tx_cnt;
reg [7:0]   tx_reg;
always@(posedge clk or posedge rst)
begin
```

```
    if(rst)
        curr_state  <= C_IDLE;
    else
        curr_state  <= next_state;
end
always@(*)
begin
    case(curr_state)
        C_IDLE:
            if(tx_ready==1'b1)
                next_state  = C_TX;
            else
                next_state  = C_IDLE;
        C_TX:
            if((div_cnt==DIV_CNT)&&(tx_cnt>=TX_CNT))
                next_state  = C_IDLE;
            else
                next_state  = C_TX;
    endcase
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        div_cnt <= 10'h0;
    else if(curr_state==C_TX)
    begin
        if(div_cnt>=DIV_CNT)
            div_cnt <= 10'h0;
        else
            div_cnt <= div_cnt + 10'h1;
    end
    else
        div_cnt <= 10'h0;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_cnt  <= 4'h0;
    else if(curr_state==C_TX)
    begin
        if(div_cnt==DIV_CNT)
            tx_cnt <= tx_cnt + 1'b1;
    end
    else
```

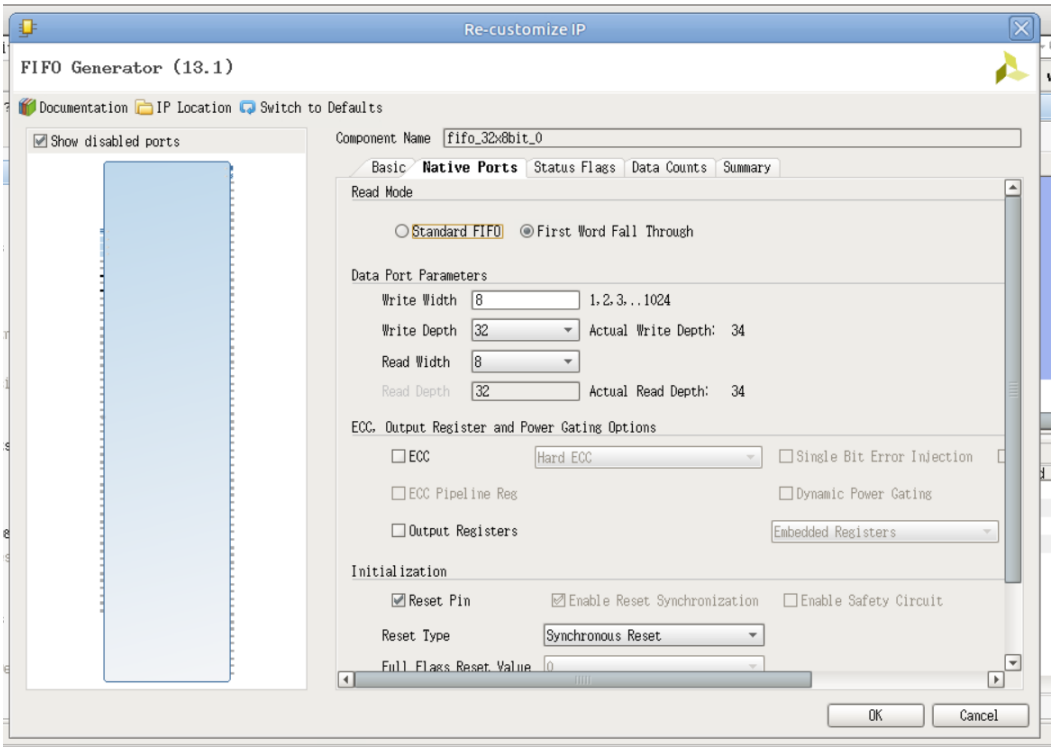
```
        tx_cnt <= 4'h0;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_rd   <= 1'b0;
    else if((curr_state==C_IDLE)&&(tx_ready==1'b1))
        tx_rd   <= 1'b1;
    else
        tx_rd   <= 1'b0;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_reg  <= 8'b0;
    else if((curr_state==C_IDLE)&&(tx_ready==1'b1))
        tx_reg  <= tx_data;
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        tx  <= 1'b1;
    else if(curr_state==C_IDLE)
        tx  <= 1'b1;
    else if(div_cnt==10'h0)
    begin
        case(tx_cnt)
            4'h0:   tx  <= 1'b0;
            4'h1:   tx  <= tx_reg[0];
            4'h2:   tx  <= tx_reg[1];
            4'h3:   tx  <= tx_reg[2];
            4'h4:   tx  <= tx_reg[3];
            4'h5:   tx  <= tx_reg[4];
            4'h6:   tx  <= tx_reg[5];
            4'h7:   tx  <= tx_reg[6];
            4'h8:   tx  <= tx_reg[7];
            4'h9:   tx  <= 1'b1;
        endcase
    end
end
endmodule
```

③ip 核-FIFO

用以临时存储数据，起到缓冲作用.

大小是 32×8 bit.

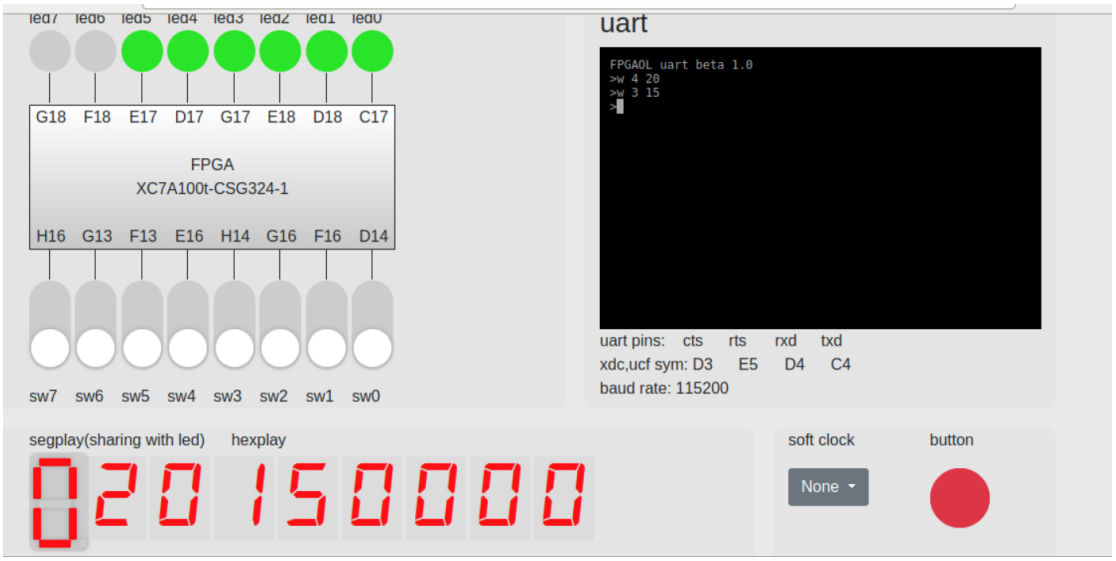


step2 测试

（1）>w 4 20

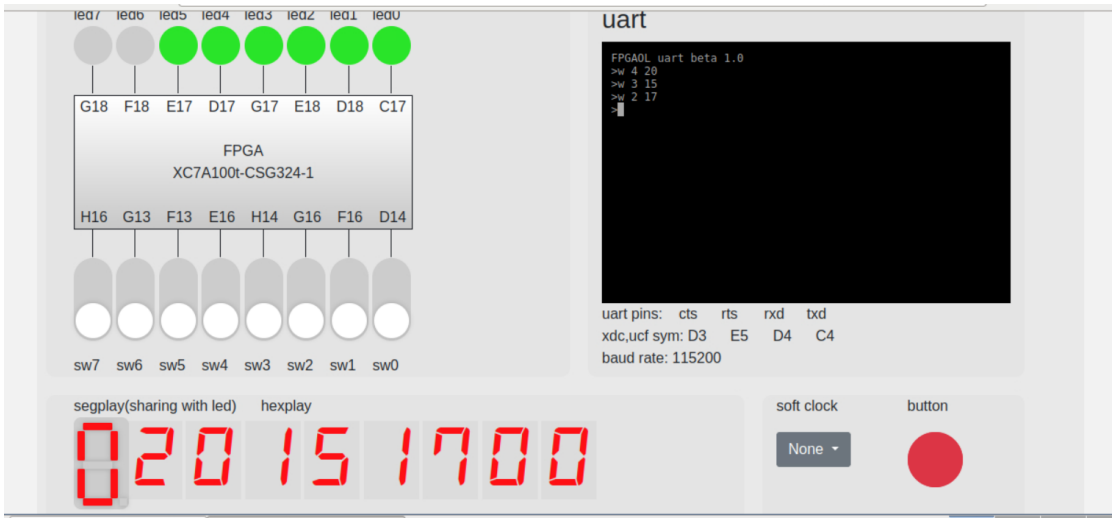
>w 3 15

地址 4 写入 20，地址 3 写入 15.



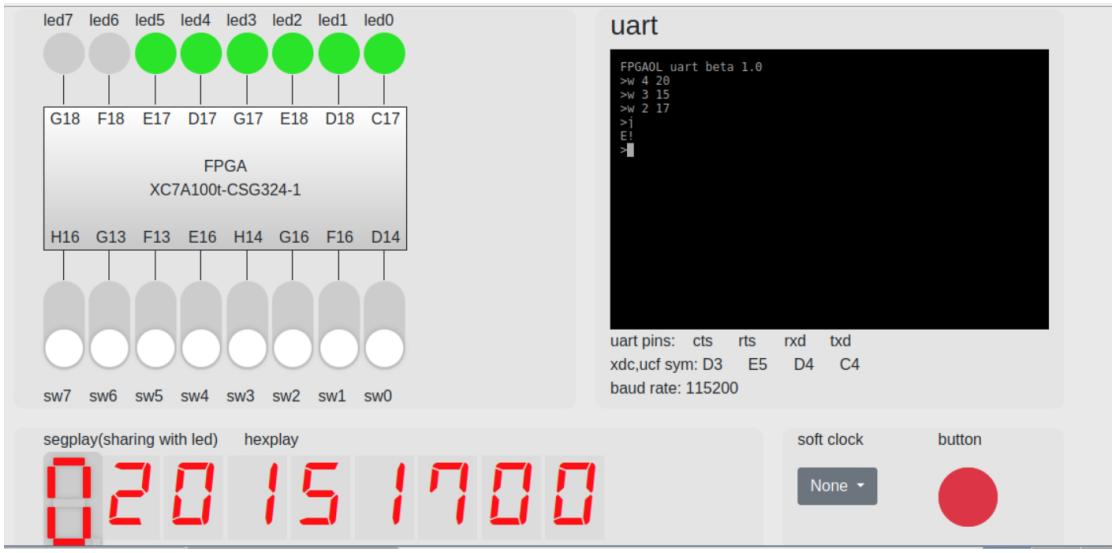
（2）>w 2 17

地址 2 写入 17.



（3）>j

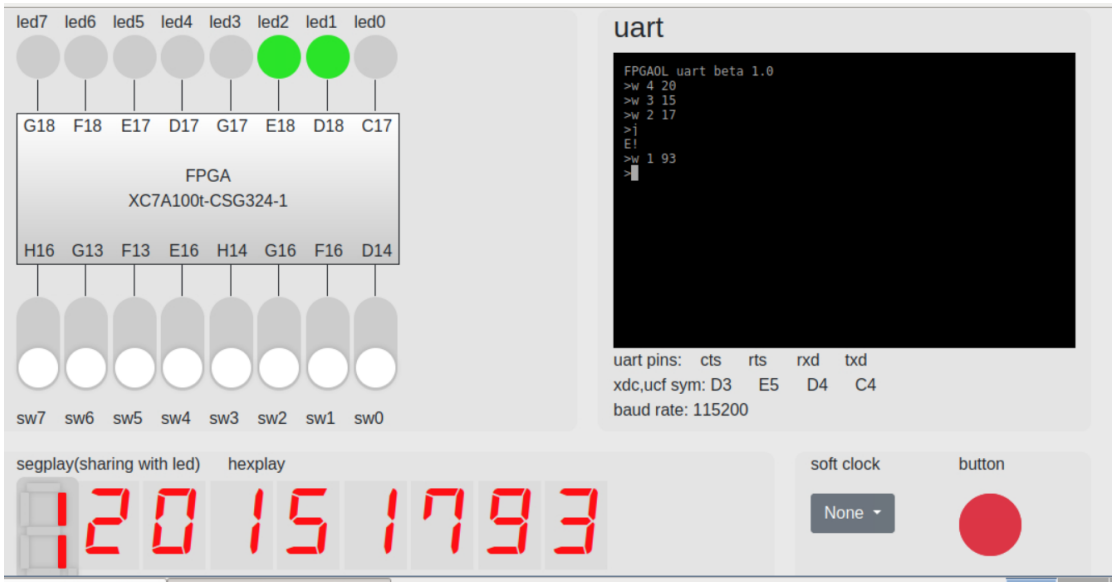
判断



输出 E！表明当前八位不是本人学号

（4）>w 1 93

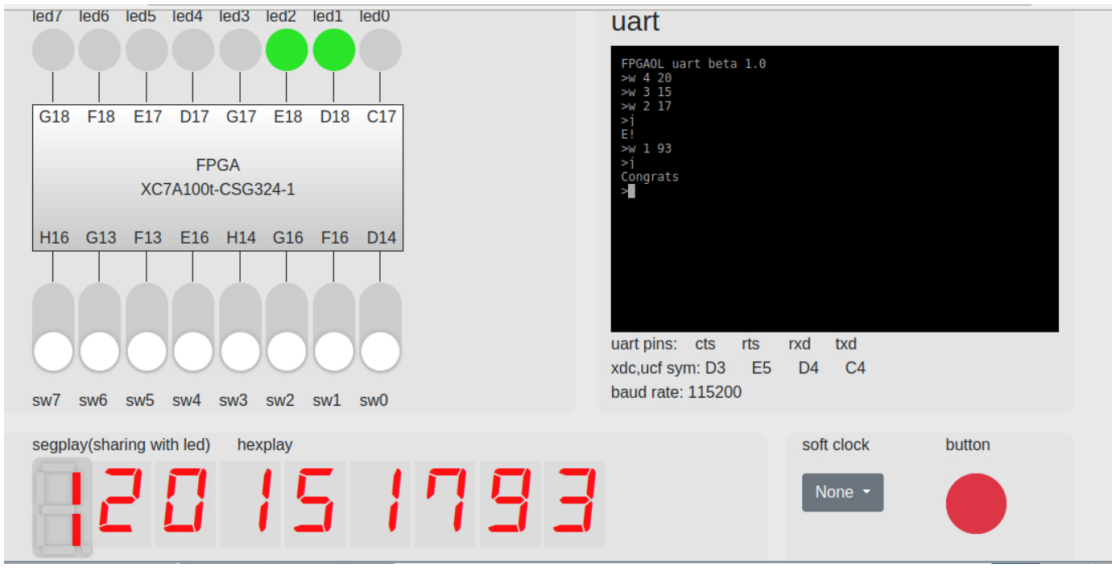
地址 1 写入 93



此时七段数码管变为 1，表明当前八位为学号.

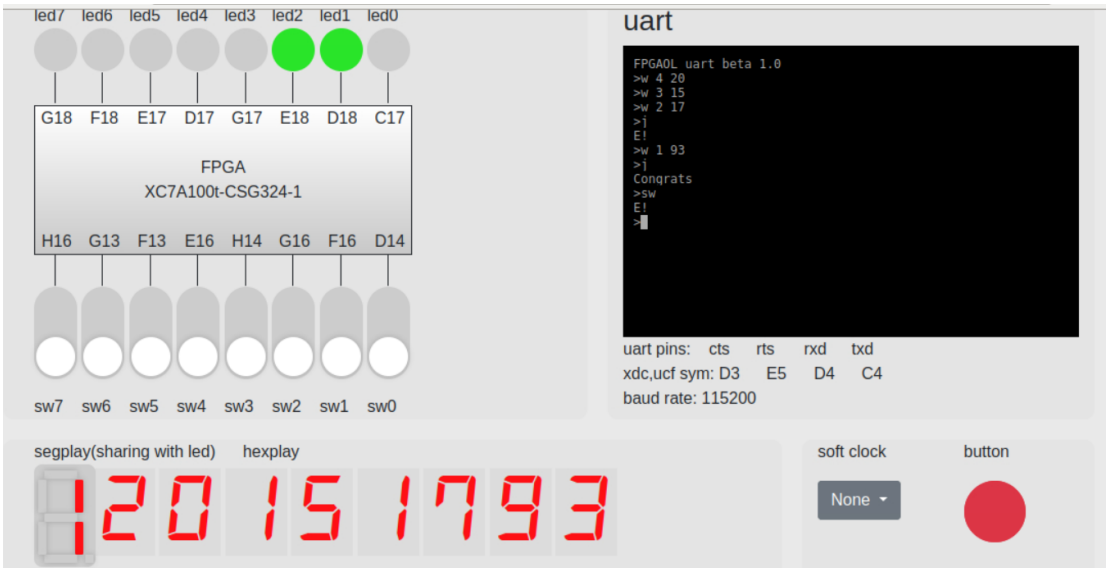
（5）>j

判断



输出 congrats，恭喜“猜”对学号，当前写入的八位为本人学号。

（6）>sw

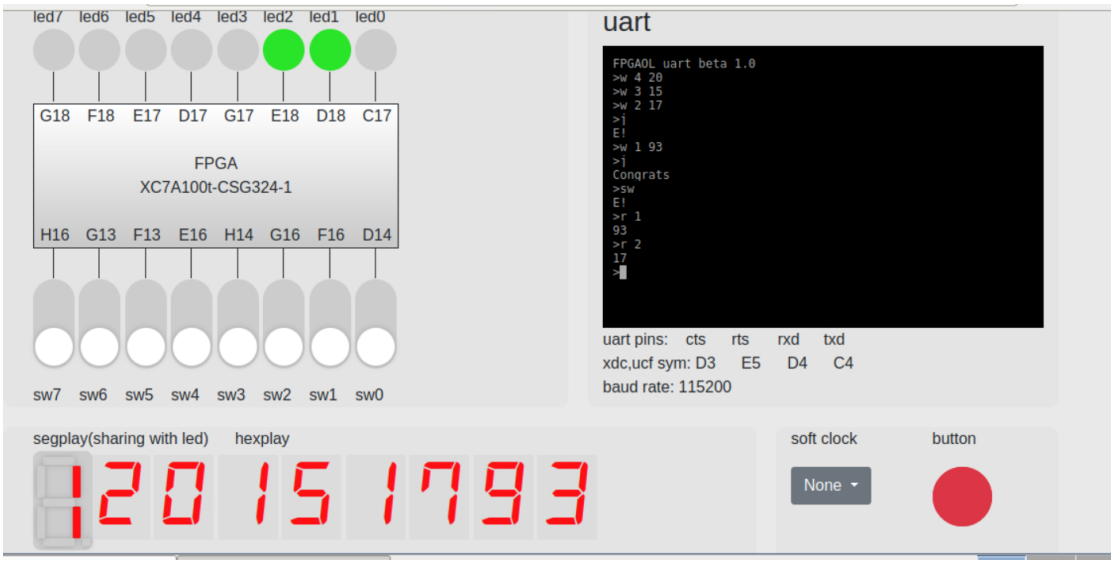


输出 E！　表示当前为错误输入.

（7）>r 1

>r 2

读地址 1 和 2 的数据



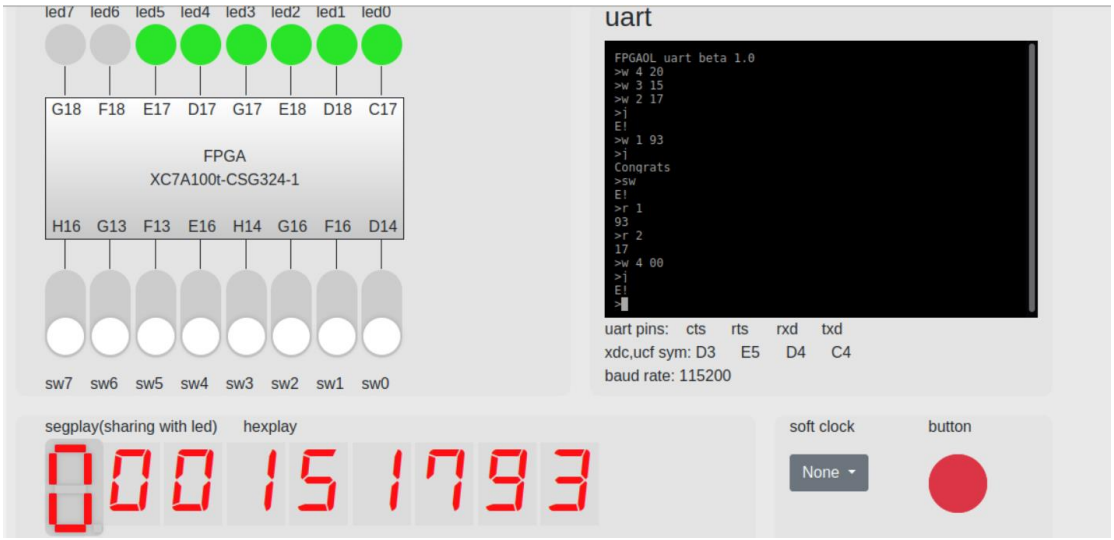
输出分别为 93 和 17.

（8）>w 4 00

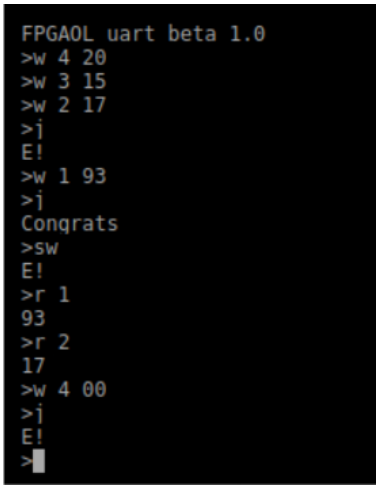
>j

地址 4 写入 00

并判断



七段数码管显示 0.　输出 E!



【总结与思考】

本次实验是建立在“串口测试说明文档”基础之上，经过改写，添加等，最终实现的。起初在 fifo 参数设置以及串口（tx，rx）的连接上出现了一些问题，不过最后在助教和老师的帮助下，解决了这些问题。此次实验由于逻辑较复杂，实现步骤繁多，因此花费了不少时间。难度较前 8 次实验有明显的增大。也让我站在一个新的高度，体验了实现复杂功能的代码的编写。此次综合实验——“猜学号”shell 让我尝试了新的思考，新的创意。

最后，感谢一直为我提供帮助的助教，老师以及同学们！