**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**

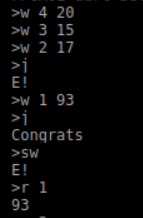
**地址2：数码管 3~2 位所表示的字节数据**

**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

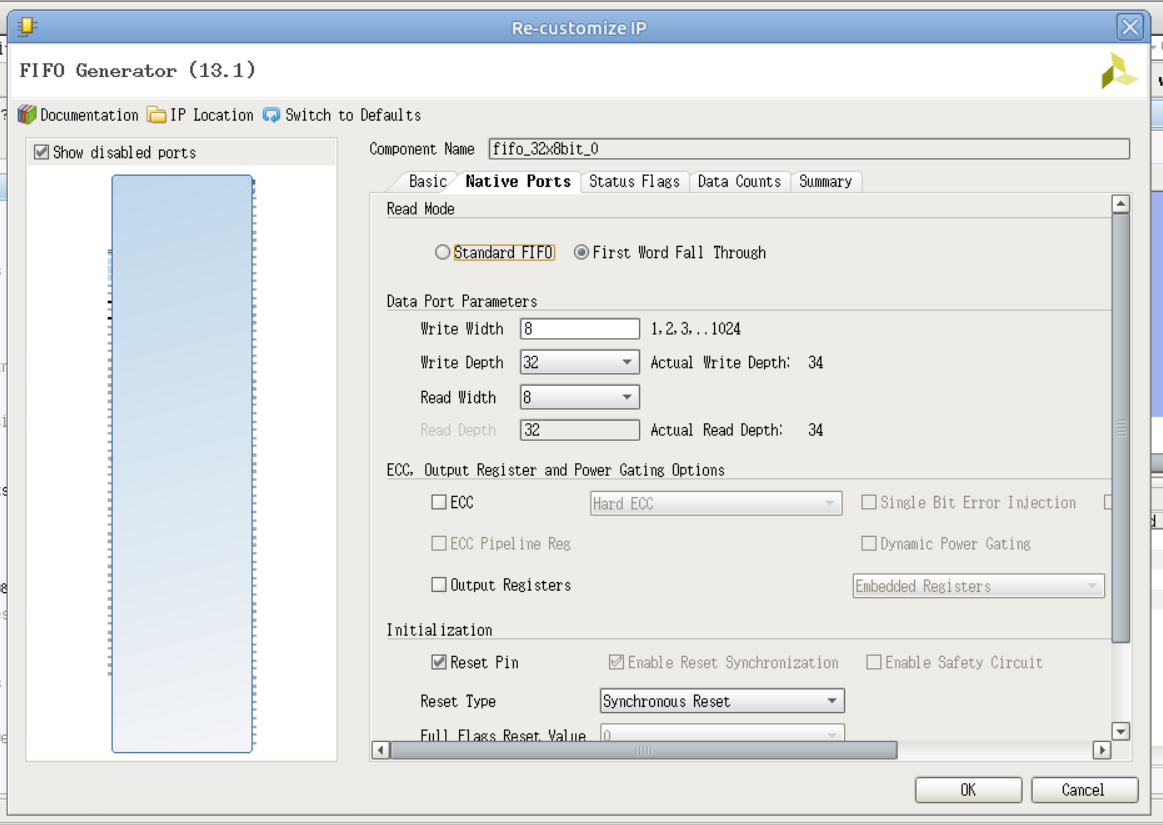
end

endmodule

**③ip核-FIFO**

用以临时存储数据，起到缓冲作用.

大小是32×8 bit.

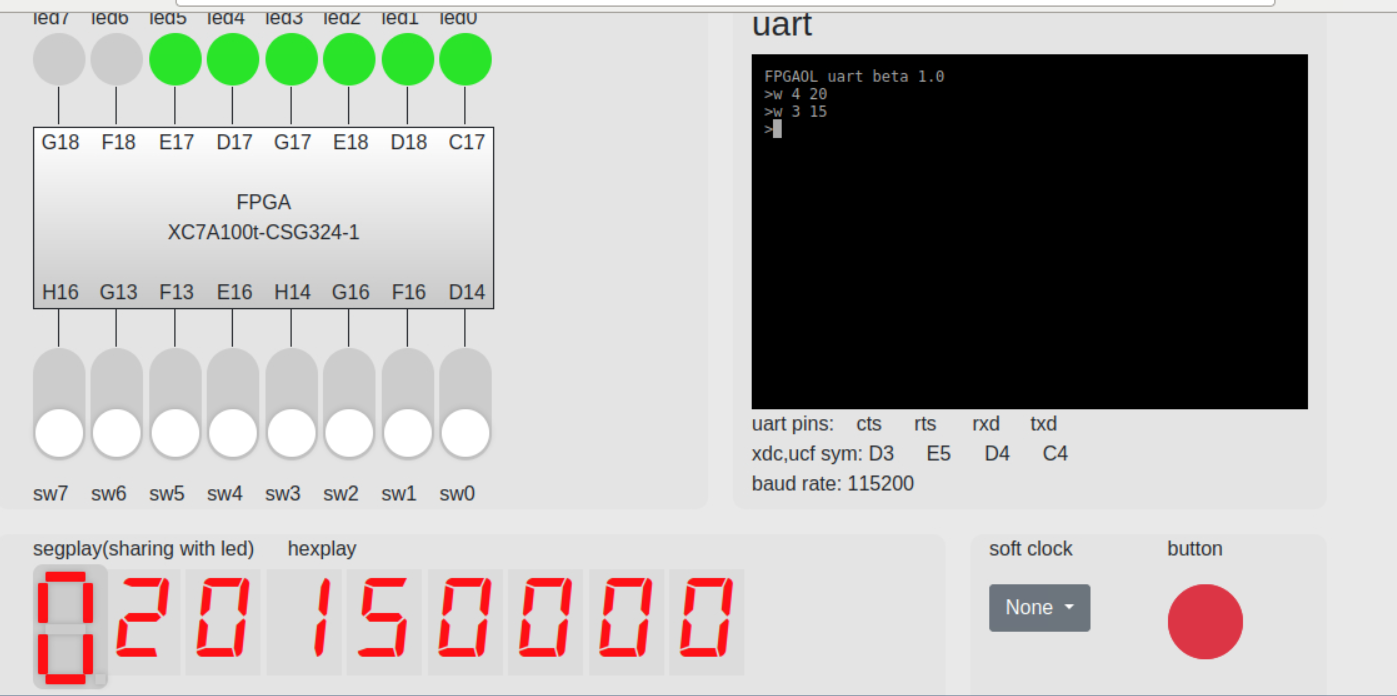


**step2测试**

（1）>w 4 20

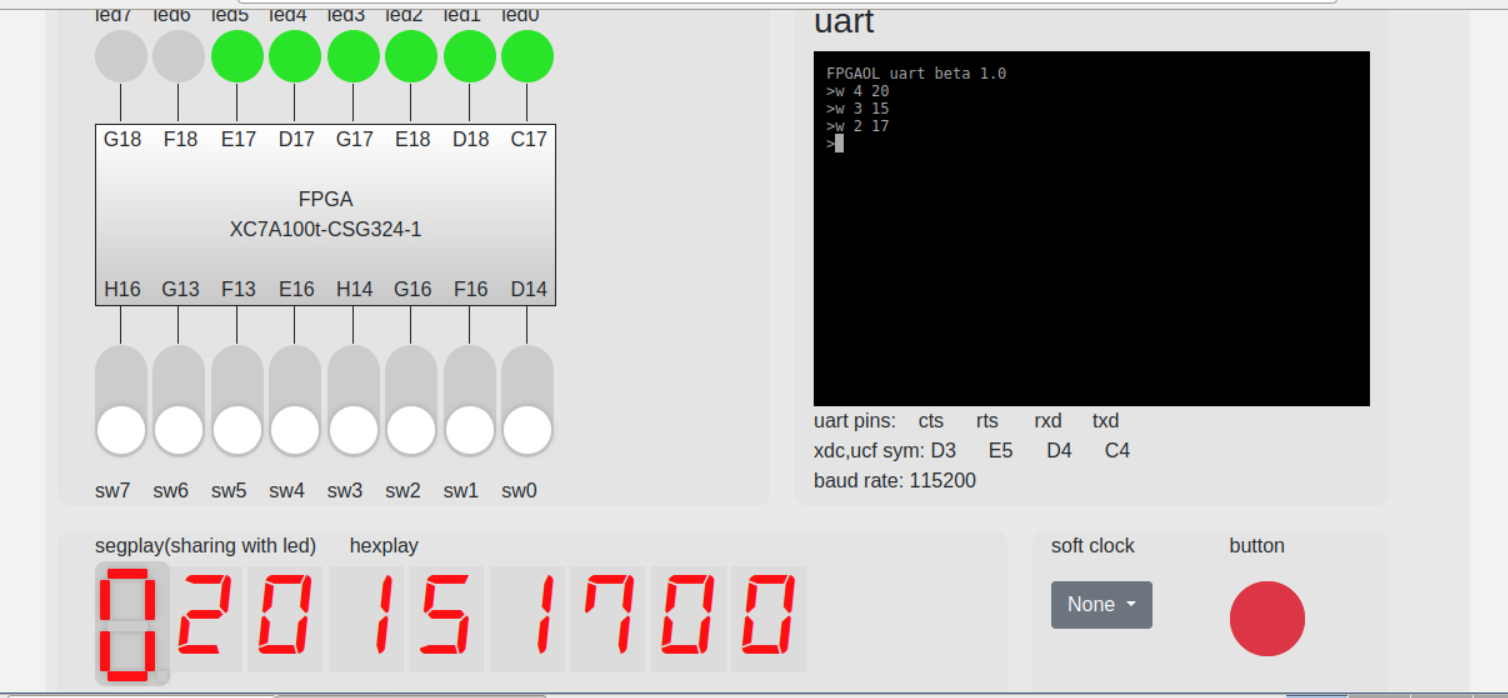
>w 3 15

地址4写入20，地址3写入15.



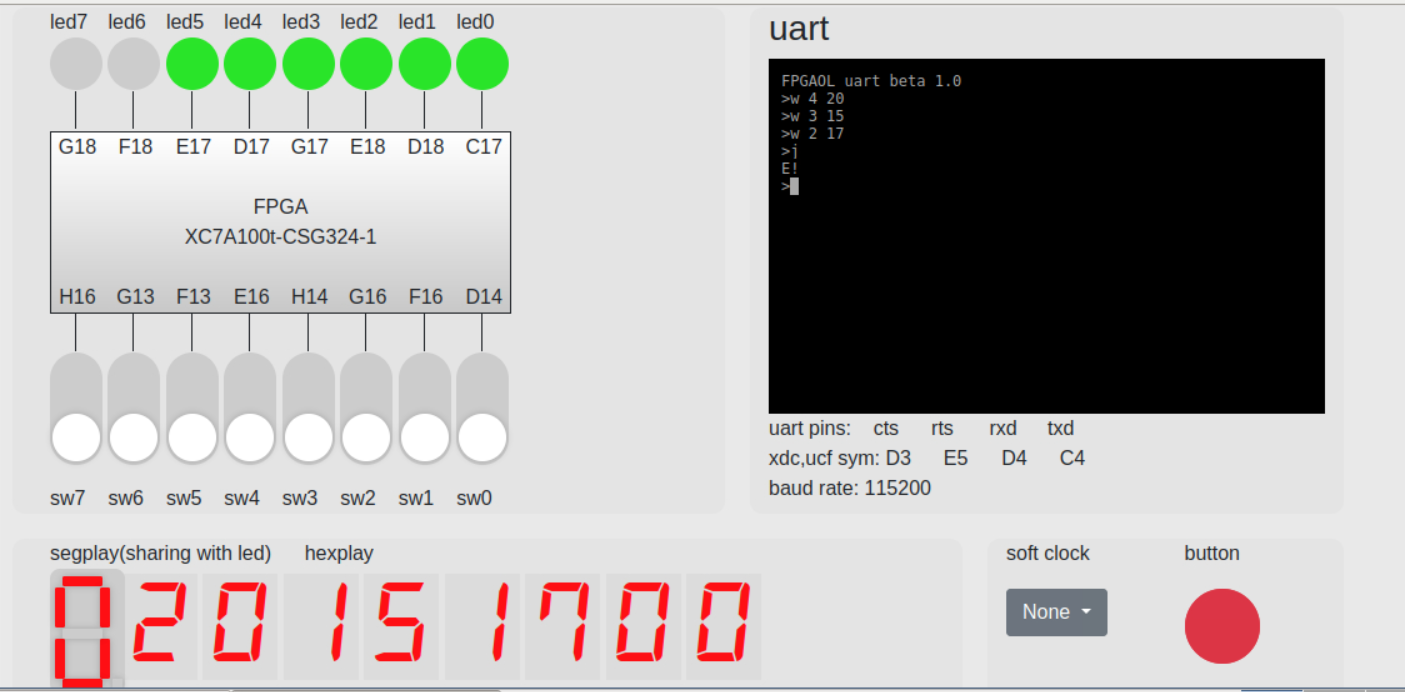
（2）>w 2 17

地址2写入17.



（3）>j

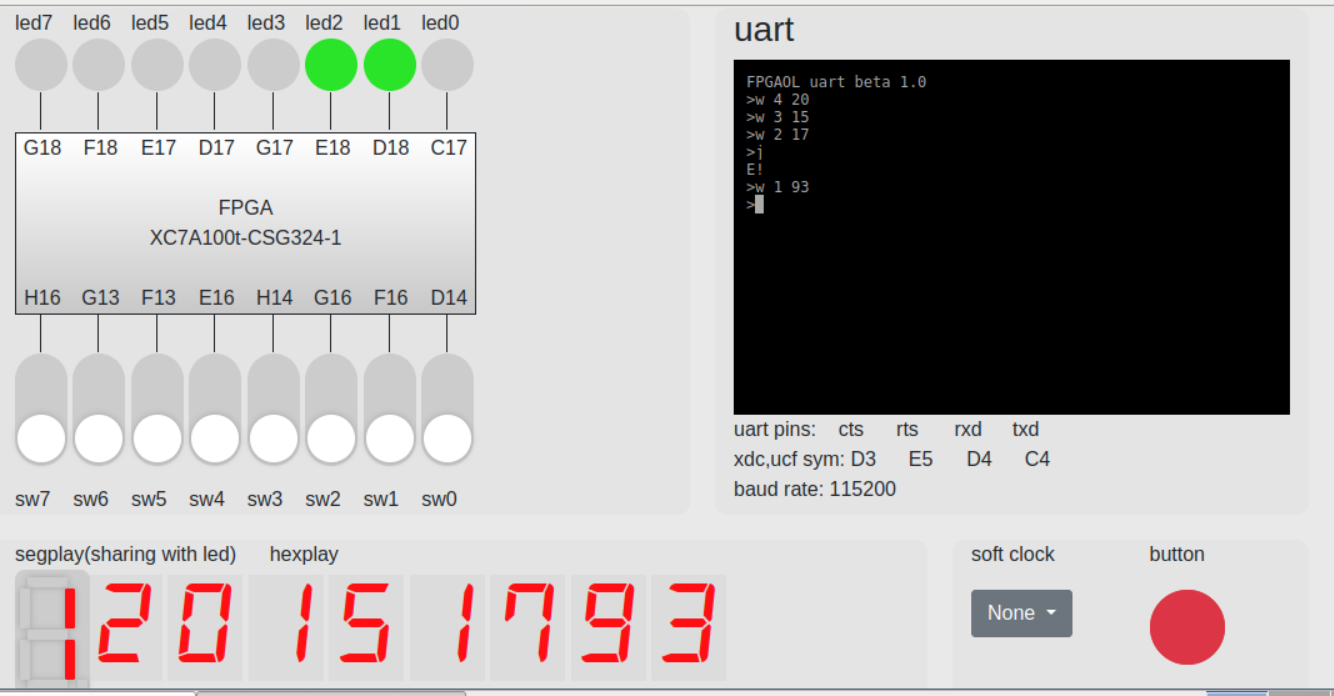
判断



输出E！表明当前八位不是本人学号

（4）>w 1 93

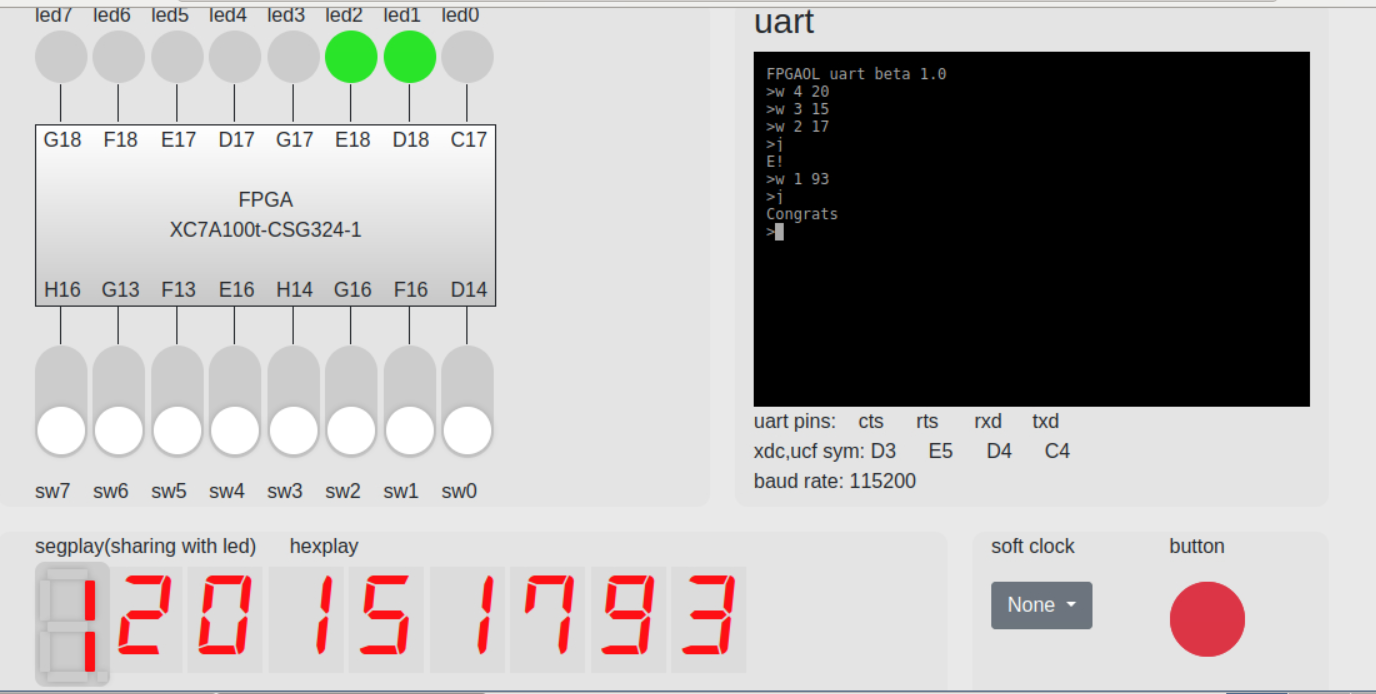
地址1写入93



此时七段数码管变为1，表明当前八位为学号.

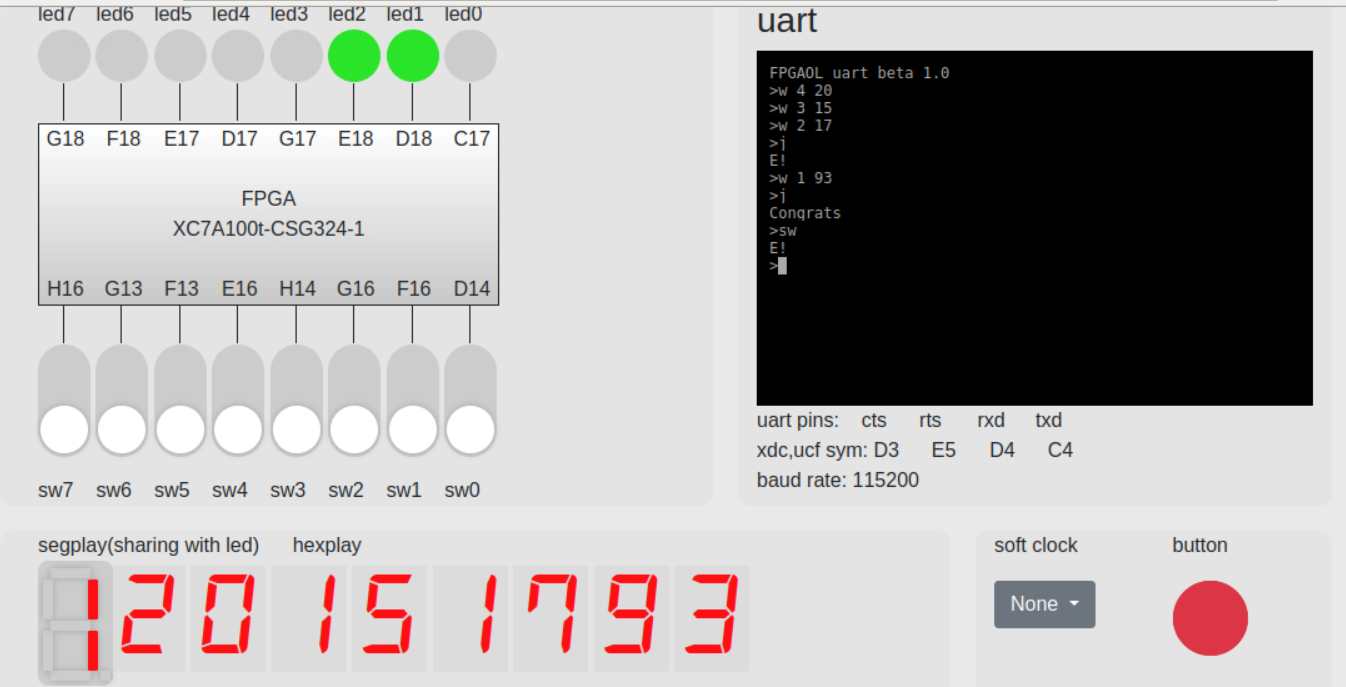
（5）>j

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

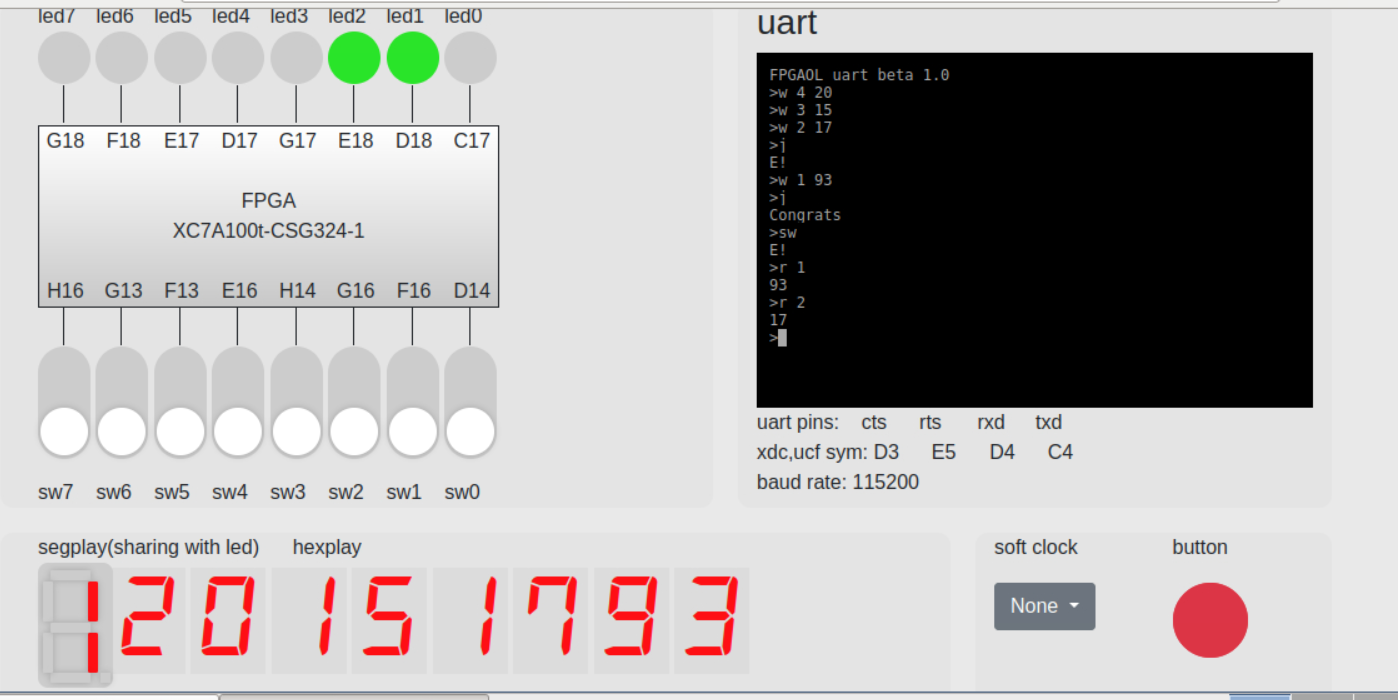


输出E！ 表示当前为错误输入.

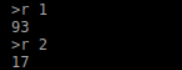
（7）>r 1

>r 2

读地址1和2的数据



输出分别为93和17.

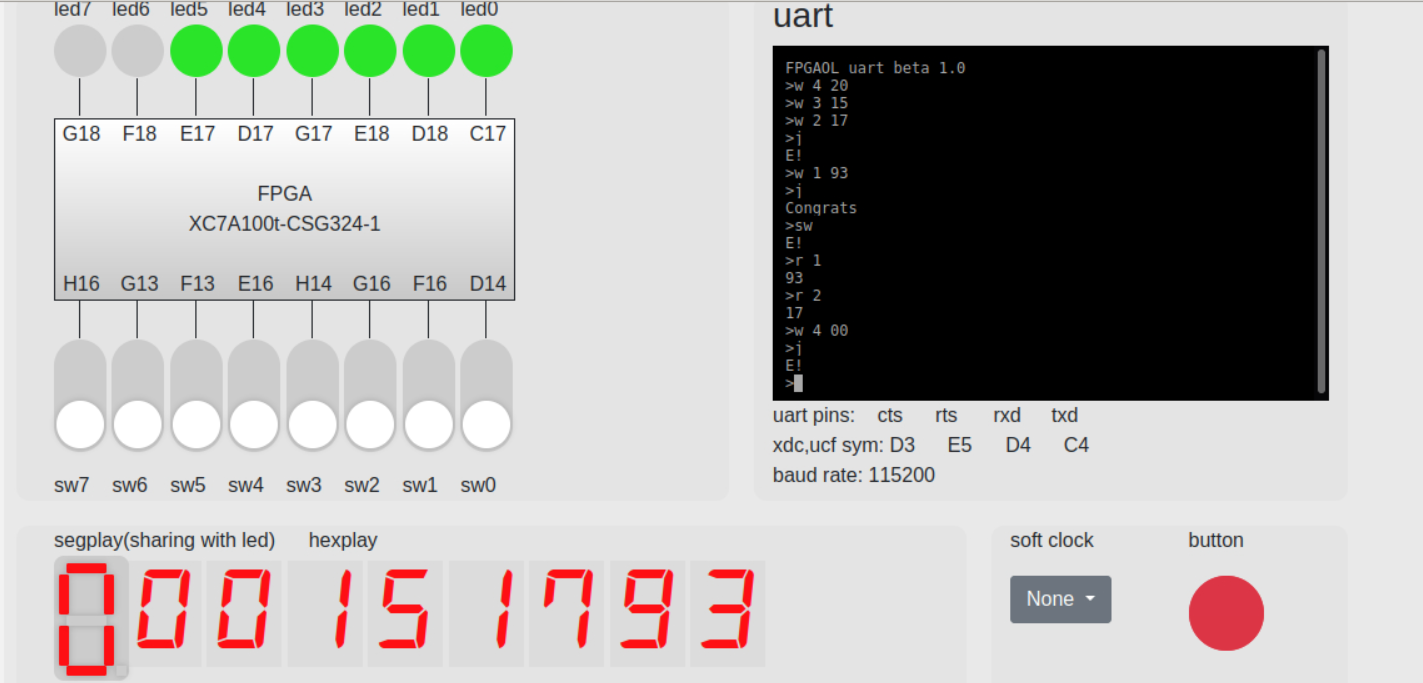


（8）>w 4 00

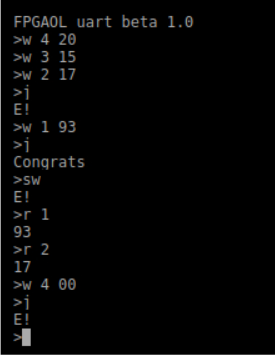
>j

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！