Lab 8 report

PB2111711 陈昕琪

方向2 串口通信与应用

实验目的与内容

> 1. 基于串口通信协议，实现串口模块的输入输出。
> 2. 根据实验教程的要求和介绍，实现基础版猜数字小游戏。
> 3. 进一步掌握串口通信协议等有关知识，并运用所给的代码，在基础版小游戏的基础上，增加一些高级功能，实现进阶版猜数字小游戏。

Part 1：串口协议

2-1 串口回显

> **要求：**
>
> 在`FPGAOL`平台上验证串口回显程序的功能。在正常情况下，在发送窗口输入数据并点击发送，之后会在接收窗口收到同样的内容。

代码已经在文档中给出，实现结果如下

2-2 串口发送模块

> **要求：**
>
> 根据教程的相关内容补充 `Send` 模块的完整代码。上板测试要求，通过 `sw[6:0]` 输入数据，按下按钮后将 `{25'B0, sw[6:0]}` 通过串口发送回来。串口界面显示的数据为 ASCII = `{25'B0, sw[6:0]}` 的字符。

> **逻辑实现：**
>
> 根据实验文档中串口发送模块的部分，可以分析出发送模块需要一个状态机和对应的分频计数器、位计数器。在到达相应的时间时，发送当前位的内容，并加入起始位和停止位作为标志。补全代码后可以正常运行。
>
> 同时要进行btn的上升沿检测，以避免出现输出多个字符这种情况。

代码如下（只展示Send模块）：

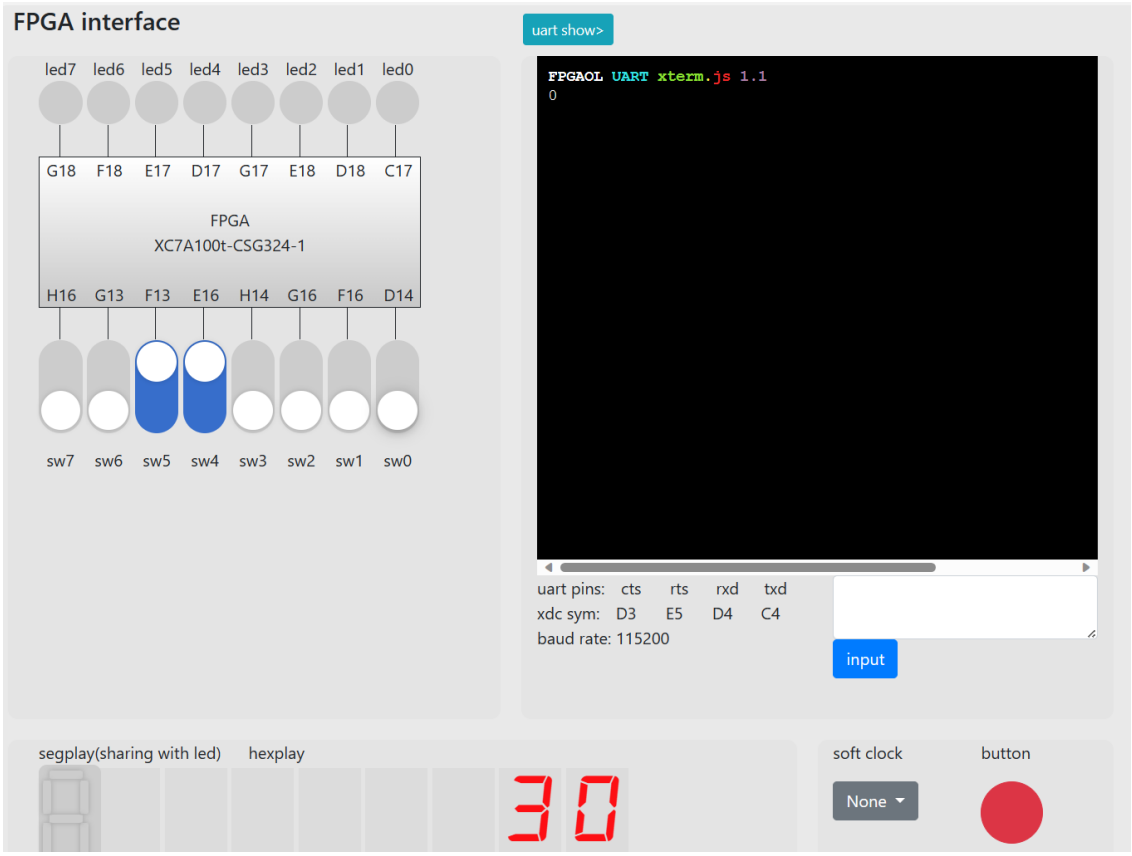
```
module Send(
    input                   [ 0 : 0]        clk,
    input                   [ 0 : 0]        rst,

    output      reg         [ 0 : 0]        dout,

    input                   [ 0 : 0]        dout_vld,
    input                   [ 7 : 0]        dout_data
);

// Counter and parameters
localparam FullT        = 867;
localparam TOTAL_BITS   = 9;
reg [ 9 : 0] div_cnt;
reg [ 4 : 0] dout_cnt;
// Main FSM
localparam WAIT     = 0;
localparam SEND     = 1;
reg current_state, next_state;
always @(posedge clk) begin
    if (rst)
        current_state <= WAIT;
    else
        current_state <= next_state;
end
//Main FSM
always @(*) begin
    next_state = current_state;
    case (current_state)
        WAIT: begin
            if(dout_vld == 1) next_state = SEND;
            else next_state = WAIT;
        end
        SEND:begin
            if(dout_cnt == TOTAL_BITS && div_cnt == FullT) next_state = WAIT;
            else next_state = SEND;
        end
```

```
    endcase
end

// Counter,0~867
always @(posedge clk) begin
    if (rst)
        div_cnt <= 10'H0;
    else if (current_state == SEND) begin
        if(div_cnt != FullT) div_cnt <= div_cnt + 1'b1;
        else div_cnt <= 10'H0;
    end
    else
        div_cnt <= 10'H0;
end
always @(posedge clk) begin
    if (rst)
        dout_cnt <= 4'H0;
    else if (current_state == SEND) begin
        if(dout_cnt != TOTAL_BITS && div_cnt == FullT) dout_cnt <= dout_cnt +
1'b1;
        else if(dout_cnt == TOTAL_BITS && div_cnt == FullT) dout_cnt <= 4'H0;
    end
    else
        dout_cnt <= 4'H0;
end

reg [7 : 0] temp_data;
always @(posedge clk) begin
    if (rst)
        temp_data <= 8'H0;
    else if (current_state == WAIT && dout_vld)
        temp_data <= dout_data;
end

always @(posedge clk) begin
    if (rst)
        dout <= 1'B1;
    else begin
        if(next_state == WAIT) dout <= 1'B1;
        else
        if(next_state == SEND)begin
           if(dout_cnt == 0) begin
            if(div_cnt == FullT) dout <= temp_data[dout_cnt];
            else dout <= 1'B0;
            end
            else begin
            if (div_cnt == FullT) begin
                if(dout_cnt != TOTAL_BITS-1) begin
                    dout <= temp_data[dout_cnt];
                end else dout <= 1'B1;
            end  else dout <= temp_data[dout_cnt-1];
            end
            if(dout_cnt == TOTAL_BITS) dout <= 1'B1;
        end
        end
```

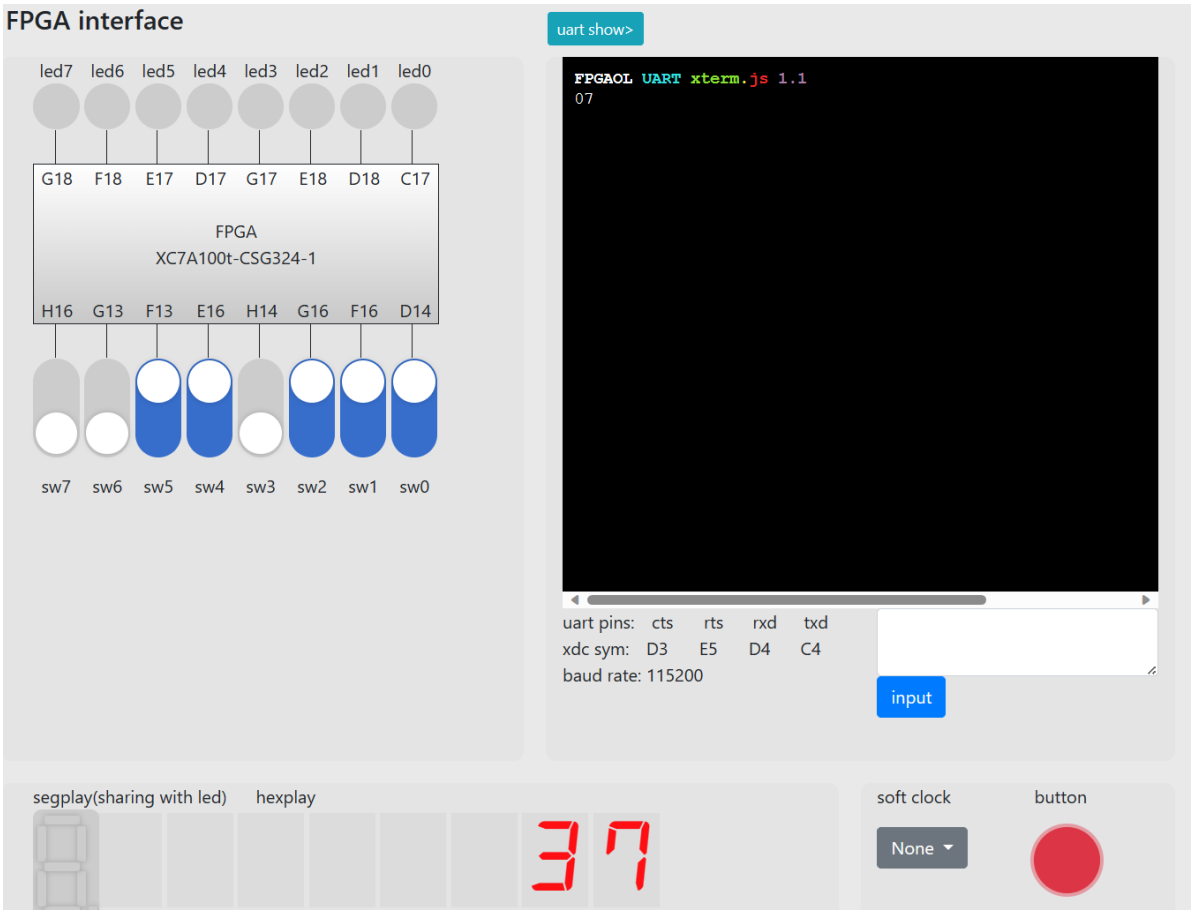
```
end
endmodule
```

上板测试结果如下：

1. `sw[6:0] =8'H30`，`ASCII`值对应的字符为`0`



2. `sw[6:0] =8'H37`，`ASCII`值对应的字符为`7`(0为上次测试留下的)



2-3 串口接收模块

> **要求：**
>
> 根据教程的相关内容补充 `Receive` 模块的完整代码。上板要求接收来自串口输入的数据，并将其以十六进制整数的形式显示在数码管上。

> **逻辑实现：**
>
> 与发送模块类似，根据实验文档中接受模块的波形图可以看出，同样需要分频计数器和位计数器，并进行分频计数。不同的是，接收模块的起始位是在分频计数器打到433时，位计数器初始化为0，而当计数器的值变为867时，是最佳采样时间，则进行采样。

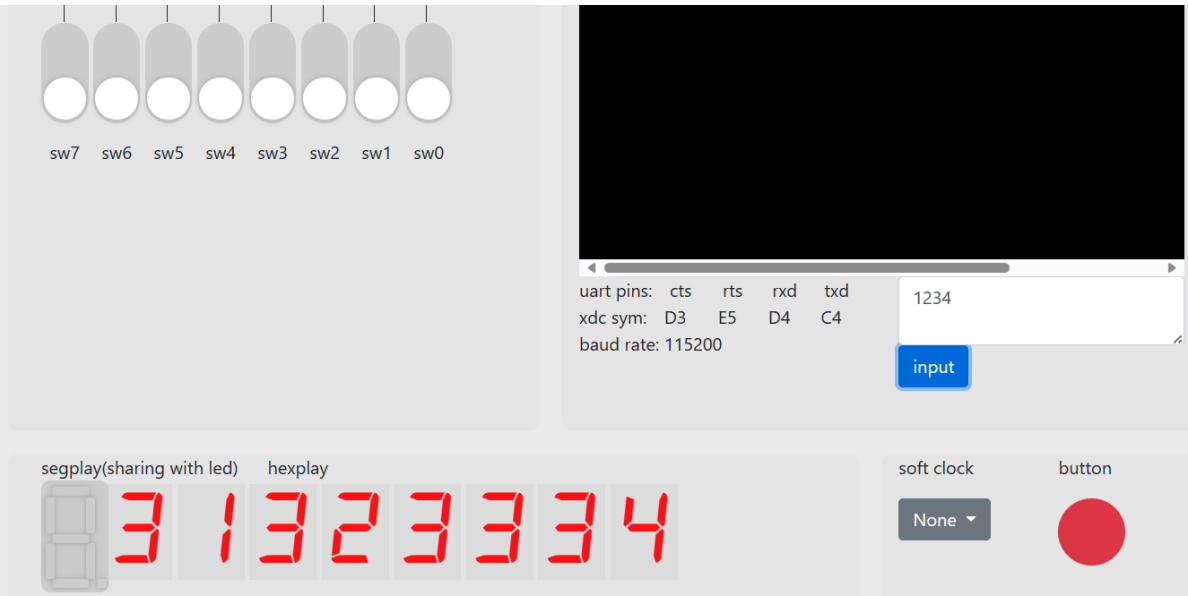
代码如下（只展示Receive模块）：

```
module Receive(
    input                   [ 0 : 0]        clk,
    input                   [ 0 : 0]        rst,

    input                   [ 0 : 0]        din,

    output      reg         [ 0 : 0]        din_vld,
    output      reg         [ 7 : 0]        din_data
);

// Counter and parameters
localparam FullT        = 867;//
localparam HalfT        = 433;//起始位中间时刻
localparam TOTAL_BITS   = 8;//总位数
reg [ 9 : 0] div_cnt;       // 分频计数器，范围 0 ~ 867
reg [ 3 : 0] din_cnt;       // 位计数器，范围 0 ~ 8

// Main FSM
localparam WAIT     = 0;
localparam RECEIVE  = 1;
reg current_state, next_state;
always @(posedge clk) begin
    if (rst)
        current_state <= WAIT;
    else
        current_state <= next_state;
end

always @(*) begin
    next_state = current_state;
    case (current_state)
        WAIT : begin
            if(div_cnt >= HalfT + 1) next_state = RECEIVE;
            else next_state = WAIT;
        end
        RECEIVE : begin
            if(din_cnt < 8) next_state = RECEIVE;
            else if(din_cnt >= 8 && div_cnt >= FullT)
                next_state = WAIT;
        end
```

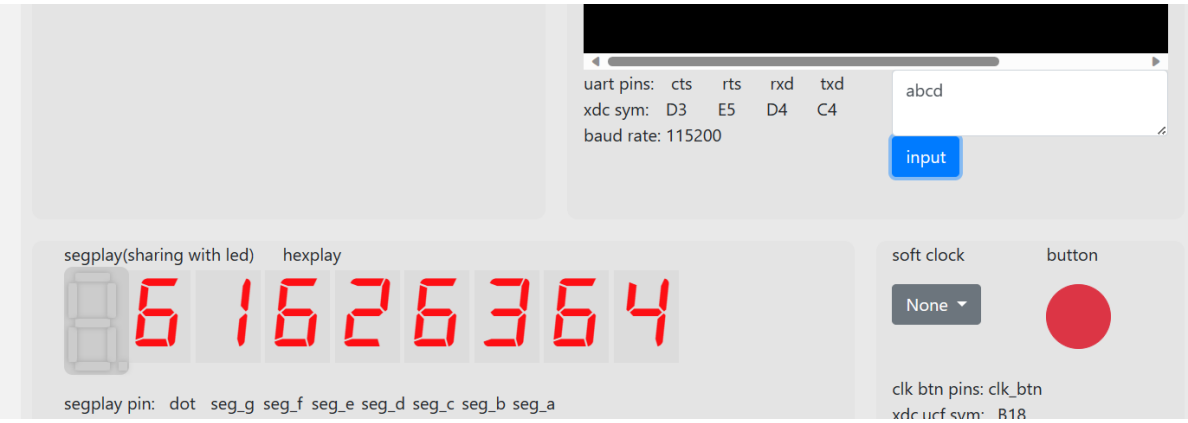
```
    endcase
end

// Counter
//分频计数器
always @(posedge clk) begin
    if (rst)
        div_cnt <= 10'D0;
    else if (current_state == WAIT) begin // STATE WAIT
        if(din == 0) begin
            if(div_cnt != HalfT + 1) div_cnt <= div_cnt + 1;
            else div_cnt <= 0;
        end
    end
    else begin  // STATE RECEIVE
        if(div_cnt != FullT) begin
            div_cnt <= div_cnt + 1;
        end else div_cnt <= 0;
    end
end
//位计数器
always @(posedge clk) begin
    if (rst)
        din_cnt <= 0;
    else begin
        if(div_cnt == FullT) begin
            if(din_cnt != TOTAL_BITS) din_cnt <= din_cnt + 1;
            else din_cnt <= 0;
        end
    end
end

// Output signals
reg [ 0 : 0] accept_din;    // 位采样信号
always @(*) begin
    //accept_din = 1'B0;
    accept_din = (div_cnt == FullT && next_state == RECEIVE);
end

always @(*) begin
    din_vld = (din_cnt == TOTAL_BITS && next_state == WAIT);
end

always @(posedge clk) begin
    if (rst)
        din_data <= 8'B0;
    else if (current_state == WAIT)
        din_data <= 8'B0;
    else if (accept_din)
        din_data <= din_data | (din << din_cnt);
end
endmodule
```

上板测试结果如下：

1. 测试输入`1234`



2. 测试输入`abcd`



> 根据ASCII表知程序正确

Part 2-A：猜数字小游戏

> 游戏介绍：
>
> 程序随机生成 3 个介于 0 ~ 5 之间的、互不相同的数码，由玩家进行猜测。玩家每次通过拨码开关输入自己猜测的数据序列，按下按钮表明输入完毕。随后，程序检查玩家最近的三次输入和生成的数码是否匹配，并通过 LED 给出对应的反馈结果。玩家在看到结果之后，会再次按下按钮以继续输入自己的猜测。在玩家操作同时，数码管（Segments）会显示一个倒计时，在倒计时结束之前如果匹配正确则视为用户胜利，否则会视为用户失败。之后，用户再次按下按钮，以开始一局新的游戏。

2-A-1：开关输入

> **要求：**
>
> 确定用户拨动开关的时机以及对应的编号

> **逻辑实现：**
>
> 运用三级寄存器边沿检测，通过按位异或`sw_reg_3`和`sw_reg_2`判断出`sw_change`,并根据`sw_change`判断出用户拨动了哪位。

代码如下：

```
module SW_Input(
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,
    input                   [ 7 : 0]            sw,

    output      reg         [ 3 : 0]            hex,//表示动了哪一位
    output                  [ 0 : 0]            pulse//表示开关是关闭还是开启
);
// 三级寄存器边沿检测
reg [7:0] sw_reg_1, sw_reg_2, sw_reg_3;
always @(posedge clk) begin
    if (rst) begin
        sw_reg_1 <= 0;
        sw_reg_2 <= 0;
        sw_reg_3 <= 0;
    end
    else begin
        sw_reg_1 <= sw;
        sw_reg_2 <= sw_reg_1;
        sw_reg_3 <= sw_reg_2;
    end
end
wire [7:0] sw_change = sw_reg_3 ^ sw_reg_2;// TODO: 检测上升沿
// TODO: 编写代码，产生 hex 和 pulse 信号。
// Hint: 这两个信号均为组合逻辑产生。
always @(*) begin
    case(sw_change)
     8'b0000_0001: begin
        hex = 0;
     end
     8'b0000_0010: begin
        hex = 1;
     end
     8'b0000_0100: begin
        hex = 2;
     end
     8'b0000_1000: begin
        hex = 3;
     end
     8'b0001_0000: begin
        hex = 4;
     end
     8'b0010_0000: begin
        hex = 5;
     end
     default: begin
        hex = 10;
     end
```

```
    endcase
end
//assign pulse = (sw_reg_3[hex] > sw[hex]);
assign pulse = ~(hex == 10);
endmodule
```

> 这里还需要用到一个左移模块来实现记录玩家波动开关的记录。相关代码实验文档中已给出。

2-A-2：结果对比

> **要求：**
>
> 确定当前用户的输入是否与游戏目标相符，并给出对应的检测结果。

> **逻辑实现：**
>
> 通过将`input_number`和`target_number`对比，并根据不同的情况判断出`check_result`。
>
> 这里运用嵌套的`if-else`语句，判断出不同情况的`check_result`。

代码如下：

```
module Check(
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,

    input                   [11 : 0]            input_number,
    input                   [11 : 0]            target_number,
    input                   [ 0 : 0]            start_check,

    output                  [ 5 : 0]            check_result
);
// 模块内部用寄存器暂存输入信号，从而避免外部信号突变带来的影响
reg [11:0] current_input_data, current_target_data;
always @(posedge clk) begin
    if (rst) begin
        current_input_data <= 0;
        current_target_data <= 0;
    end
    else if (start_check) begin
        current_input_data <= input_number;
        current_target_data <= target_number;
    end
end

// 使用组合逻辑产生比较结果
wire [3:0] target_number_3, target_number_2, target_number_1;
wire [3:0] input_number_3, input_number_2, input_number_1;
assign input_number_1 = current_input_data[3:0];
assign input_number_2 = current_input_data[7:4];
assign input_number_3 = current_input_data[11:8];
assign target_number_1 = current_target_data[3:0];
assign target_number_2 = current_target_data[7:4];
assign target_number_3 = current_target_data[11:8];

reg i1t1, i1t2, i1t3, i2t1, i2t2, i2t3, i3t1, i3t2, i3t3;
```

```
always @(*) begin
    i1t1 = (input_number_1 == target_number_1);//1，正确且位置正确
    i1t2 = (input_number_1 == target_number_2);//1，正但位置错误
    i1t3 = (input_number_1 == target_number_3);//1，正确但位置错误
    i2t1 = (input_number_2 == target_number_1);//1，
    i2t2 = (input_number_2 == target_number_2);//1，正确且位置正确
    i2t3 = (input_number_2 == target_number_3);//1，
    i3t1 = (input_number_3 == target_number_1);//1，
    i3t2 = (input_number_3 == target_number_2);//1，
    i3t3 = (input_number_3 == target_number_3);//1，正确且位置正确
end
reg [2:0] num_1,num_2;
always @(*) begin
    if(rst) begin
        num_1 = 0;
        num_2 = 0;
    end else if(start_check)begin
        if(i1t1)begin//已经有一个正确
            if(i2t2)begin//两个正确
                if(i3t3)begin
                    num_1 = 3'b100;
                    num_2 = 3'b000;
                end else if(i3t1 || i3t2)begin
                    num_1 = 3'b010;
                    num_2 = 3'b001;
                end else begin
                    num_1 = 3'b010;
                    num_2 = 3'b000;
                end
            end else if(i2t1 || i2t3)begin//001,001
                if(i3t3)begin
                    num_1 = 3'b010;
                    num_2 = 3'b001;
                end else if(i3t1 || i3t2)begin
                    num_1 = 3'b001;
                    num_2 = 3'b010;
                end else begin
                    num_1 = 3'b001;
                    num_2 = 3'b001;
                end
            end else begin//001,000
                if(i3t3)begin
                    num_1 = 3'b010;
                    num_2 = 3'b000;
                end else if(i3t1 || i3t2)begin
                    num_1 = 3'b001;
                    num_2 = 3'b001;
                end else begin
                    num_1 = 3'b001;
                    num_2 = 3'b000;
                end
                end
        end else if(i1t2 || i1t3) begin//000,001
            if(i2t2)begin//两个正确//001,001
                if(i3t3)begin
```

```
                num_1 = 3'b010;
                num_2 = 3'b001;
            end else if(i3t1 || i3t2)begin
                num_1 = 3'b001;
                num_2 = 3'b010;
            end else begin
                num_1 = 3'b001;
                num_2 = 3'b001;
            end
        end else if(i2t1 || i2t3)begin//000,010
            if(i3t3)begin
                num_1 = 3'b001;
                num_2 = 3'b010;
            end else if(i3t1 || i3t2)begin
                num_1 = 3'b000;
                num_2 = 3'b100;
            end else begin
                num_1 = 3'b000;
                num_2 = 3'b010;
            end
        end else begin//000,001
            if(i3t3)begin
                num_1 = 3'b001;
                num_2 = 3'b001;
            end else if(i3t1 || i3t2)begin
                num_1 = 3'b000;
                num_2 = 3'b010;
            end else begin
                num_1 = 3'b000;
                num_2 = 3'b001;
            end
            end
    end else begin
        if(i2t2)begin//两个正确
            if(i3t3)begin
                num_1 = 3'b010;
                num_2 = 3'b000;
            end else if(i3t1 || i3t2)begin
                num_1 = 3'b001;
                num_2 = 3'b001;
            end else begin
                num_1 = 3'b001;
                num_2 = 3'b000;
            end
        end else if(i2t1 || i2t3)begin
            if(i3t3)begin
                num_1 = 3'b001;
                num_2 = 3'b001;
            end else if(i3t1 || i3t2)begin
                num_1 = 3'b000;
                num_2 = 3'b010;
            end else begin
                num_1 = 3'b000;
                num_2 = 3'b001;
            end
```

```
            end else begin
                if(i3t3)begin
                    num_1 = 3'b001;
                    num_2 = 3'b000;
                end else if(i3t1 || i3t2)begin
                    num_1 = 3'b000;
                    num_2 = 3'b001;
                end else begin
                    num_1 = 3'b000;
                    num_2 = 3'b000;
                end
                end
        end
        end  else begin
            num_1 = 3'b000;
            num_2 = 3'b000;
        end
end
assign check_result = {{num_1},{num_2}};
endmodule
```

2-A-3：计时器

> **要求：**
>
> 控制倒计时的开启与关闭

> **逻辑实现：**
>
> 根据所给的使能信号，补全实验文档中的代码，并根据状态机的状态，实现倒计时的开启与关闭。

代码如下：

```
module Timer(
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,

    input                   [ 0 : 0]            set,//置1
    input                   [ 0 : 0]            en,//开始倒数

    output                  [ 7 : 0]            minute,
    output                  [ 7 : 0]            second,
    output                  [11 : 0]            micro_second,

    output                  [ 0 : 0]            finish
);

reg current_state, next_state;
localparam ON = 1;
localparam OFF = 0;

always @(posedge clk) begin
    if (rst)
        current_state <= OFF;
```

```
    else
        current_state <= next_state;
end
// TODO: Finish the FSM
always @(*) begin
    case (current_state)
    ON: begin
        if(en && ~finish) begin
            next_state = ON;
        end else next_state = OFF;
    end
    OFF:begin
        if(en) begin
            next_state = ON;
        end else next_state = OFF;
    end
    endcase
end


localparam TIME_1MS = 100_000_000 / 1000;
//localparam TIME_1MS = 2;
reg [31 : 0] counter_1ms;
// TODO: Finish the counter
always @(posedge clk) begin
    if(rst) begin
        counter_1ms <= 0;
    end else if(current_state == ON) begin
        if(counter_1ms != TIME_1MS) begin
            counter_1ms <= counter_1ms + 1;
        end else counter_1ms <= 0;
    end
end
wire carry_in[2:0];
Clock # (
    .WIDTH                  (8)    ,
    .MIN_VALUE              (0)    ,
    .MAX_VALUE              (59)   ,
    .SET_VALUE              (1)
) minute_clock (
    .clk                    (clk),
    .rst                    (rst),
    .set                    (set),
    .carry_in               (carry_in[2]),
    .carry_out              (finish),
    .value                  (minute)
);
Clock # (
    .WIDTH                  (8)   ,
    .MIN_VALUE              (0)   ,
    .MAX_VALUE              (59)  ,
    .SET_VALUE              (0)
) second_clock (
    .clk                    (clk),
    .rst                    (rst),
```

```
    .set                    (set),
    .carry_in               (carry_in[1]),
    .carry_out              (carry_in[2]),
    .value                  (second)
);
Clock # (
    .WIDTH                  (12)   ,
    .MIN_VALUE              (0)    ,
    .MAX_VALUE              (999)  ,
    .SET_VALUE              (0)
) micro_second_clock (
    .clk                    (clk),
    .rst                    (rst),
    .set                    (set),
    .carry_in               (carry_in[0]),
    .carry_out              (carry_in[1]),
    .value                  (micro_second)
);
// TODO: what's carry_in[0] ?
assign carry_in[0] = (counter_1ms == TIME_1MS);

endmodule
```

2-A-4：控制单元

> **要求：**
>
> 控制整个电路的正常工作。

> **逻辑实现：**
>
> 共四个状态，等待输入，判断结果，结果正确，结果错误。根据实验文档中的状态图写出状态机每个状态的应该输出的使能信号，包括时间使能、LED灯和数码管的显示情况（用到数据选择器来实现）。

代码如下：

```
module Control (
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,
    input                   [ 0 : 0]            btn,

    input                   [ 5 : 0]            check_result,
    output      reg         [ 0 : 0]            check_start,
    output      reg         [ 0 : 0]            timer_en,
    output      reg         [ 0 : 0]            timer_set,
    input                   [ 0 : 0]            timer_finish,

    output      reg         [ 1 : 0]            led_sel,
    output      reg         [ 1 : 0]            seg_sel
);
localparam WAIT = 2'b00;//等待输入
localparam CHECK = 2'b01;//判断出结果
localparam RIGHT = 2'b10;//结果正确
localparam WRONG = 2'b11;//结果错误
```

```
reg [1:0] current_state,next_state;
initial begin
    current_state = WAIT;
end
always @(posedge clk) begin
    if(rst)
        current_state <= WAIT;
    else
        current_state <= next_state;
end
always @(*) begin
    case (current_state)
    WAIT: begin
        if(timer_finish != 1) begin
            if(btn)begin
                if(check_result == 6'b100_000) begin
                    next_state = RIGHT;
                end else next_state = CHECK;
            end else begin
                next_state = WAIT;
            end
        end else begin
            next_state = WRONG;
        end
    end
    CHECK: begin
        if(timer_finish != 1) begin
            if(btn) begin
                next_state = WAIT;
            end else next_state = CHECK;
        end else begin
            next_state = WRONG;
        end
    end
    RIGHT: begin
        if (btn) begin
            next_state = WAIT;
        end else next_state = RIGHT;
    end
    WRONG: begin
        if(btn) begin
            next_state = WAIT;
        end else begin
            next_state = WRONG;
        end
    end
    endcase
end
always @(posedge clk) begin
    if(rst) begin
        timer_en <= 0;//时间不开始计时
        timer_set <= 1;//时间重置
    end else begin
    if(current_state == WAIT) begin
        timer_en <= 1;
```

```
        timer_set <= 0;
        check_start <= 1;
        led_sel <= 2'b00;
        seg_sel <= 2'b00;
    end
    if(current_state == CHECK) begin
        check_start <= 0;
        timer_en <= 1;
        timer_set <= 0;
        led_sel <= 2'b01;
        seg_sel <= 2'b01;
    end
    if(current_state == WRONG) begin
        check_start <= 0;
        timer_en <= 0;
        timer_set <= 1;
        led_sel <= 2'b11;
        seg_sel <= 2'b11;
    end
    if(current_state == RIGHT) begin
        check_start <= 0;
        timer_en <= 0;
        timer_set <= 1;
        led_sel <= 2'b10;
        seg_sel <= 2'b10;
    end
end
end
endmodule
```

2-A-5：组装

> **逻辑实现：**
>
> 将上述模块进行组装，并写一个Top模块实现连接。
>
> **这里定义`target_number`为`12'H123`。**

代码如下：

```
module Top (
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            btn,
    input                   [ 7 : 0]            sw,

    output                  [ 7 : 0]            led,
    output                  [ 2 : 0]            seg_an,
    output                  [ 3 : 0]            seg_data
);

wire rst = sw[7];
//btn边缘检测
reg sig_r1, sig_r2;
wire pos_btn;
always @(posedge clk) begin
```

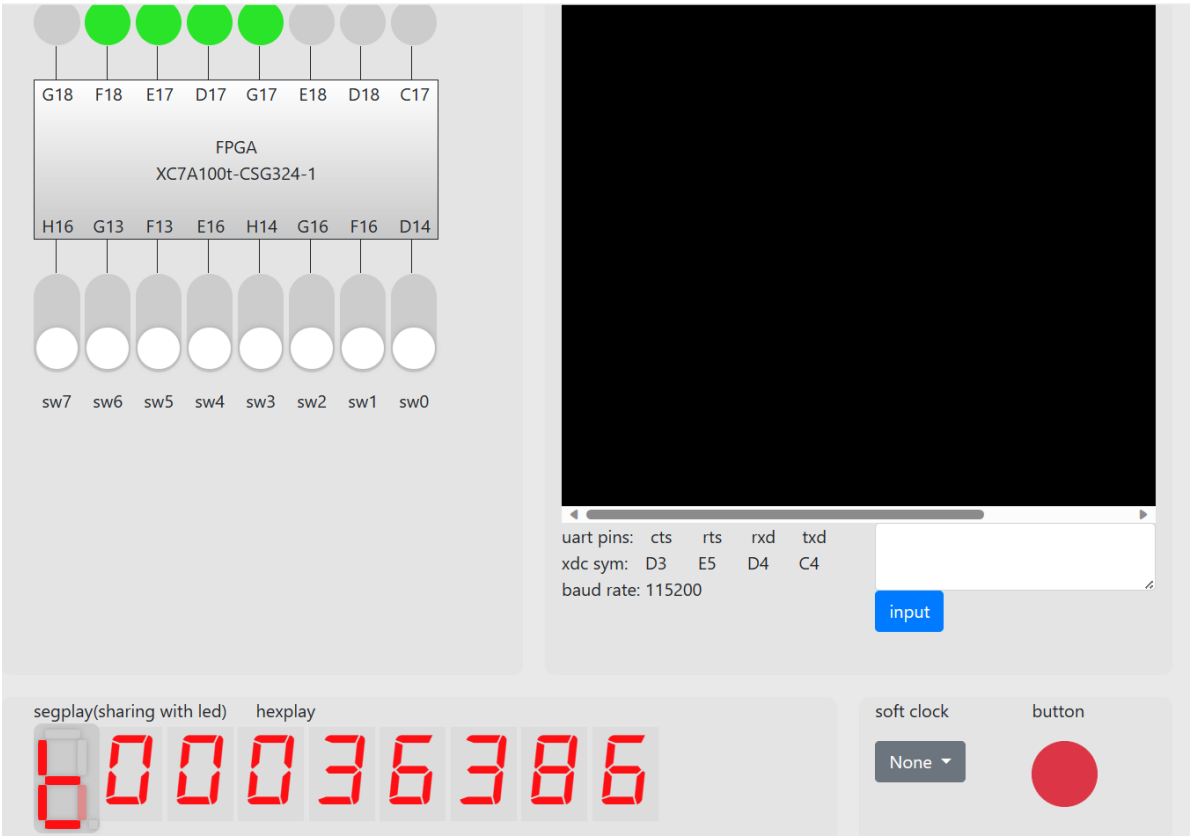
```
    if (rst) begin
        sig_r1 <= 0;
        sig_r2 <= 0;
    end
    else begin
        sig_r1 <= btn;
        sig_r2 <= sig_r1;
    end
end
assign pos_btn = (sig_r1 && ~sig_r2) ? 1 : 0;
reg [11:0] target_number = 12'H123;
reg [11:0] input_number;
wire [3:0]hex;
wire pulse;
wire [31:0] dout;
//SW_Input
SW_Input in(
    .clk(clk),
    .rst(rst),
    .sw(sw),
    .hex(hex),
    .pulse(pulse)
);
ShiftReg shiftreg(
    .clk(clk),
    .rst(rst),
    .hex(hex),
    .pulse(pulse),
    .dout(dout)
);
always @(*) begin
    input_number = dout[11:0];
end

//Check
reg start_check;
wire [5:0] check_result;
Check check(
    .clk(clk),
    .rst(rst),
    .input_number(input_number),
    .target_number(target_number),
    .start_check(start_check),
    .check_result(check_result)
);
reg [5:0] result;
always @(posedge clk) begin
    if(check_result)begin
        result <= check_result;
    end
end
//Timer
reg set,en;
wire [7:0] minute;
wire [7:0] second;
```

```
wire [11:0] micro_second;
wire finish;
Timer timer(
    .clk(clk),
    .rst(rst),
    .set(set),
    .en(en),
    .minute(minute),
    .second(second),
    .micro_second(micro_second),
    .finish(finish)
);

//MUX
wire [7:0] ledflow;
wire [1:0] led_sel;
MUX4 #(
    .WIDTH(8)
) mux_led(
    .src0(ledflow),//流水灯
    .src1({{2'b00},{result}}),//显示结果
    .src2(8'b1111_1111),//正确
    .src3(8'b0000_0000),//错误
    .sel(led_sel),
    .res(led)
);
LED_Flow led_flow(
    .clk(clk),
    .rst(rst),
    .led(ledflow)
);
wire [1:0] seg_sel;
wire [31:0] output_data;
MUX4 #(
    .WIDTH(32)
) mux_seg(
    .src0({{4'b0},{minute},{second},{micro_second}}),
    .src1({{4'b0},{minute},{second},{micro_second}}),
    .src2(32'H8888_8888),
    .src3(32'H4444_4444),
    .sel(seg_sel),
    .res(output_data)
);

Segment segment(
    .clk(clk),
    .rst(rst),
    .output_data(output_data),
    .output_valid(8'Hff),
    .seg_data(seg_data),
    .seg_an(seg_an)
);
wire timer_en,timer_set;
wire check_start;
Control control(
```

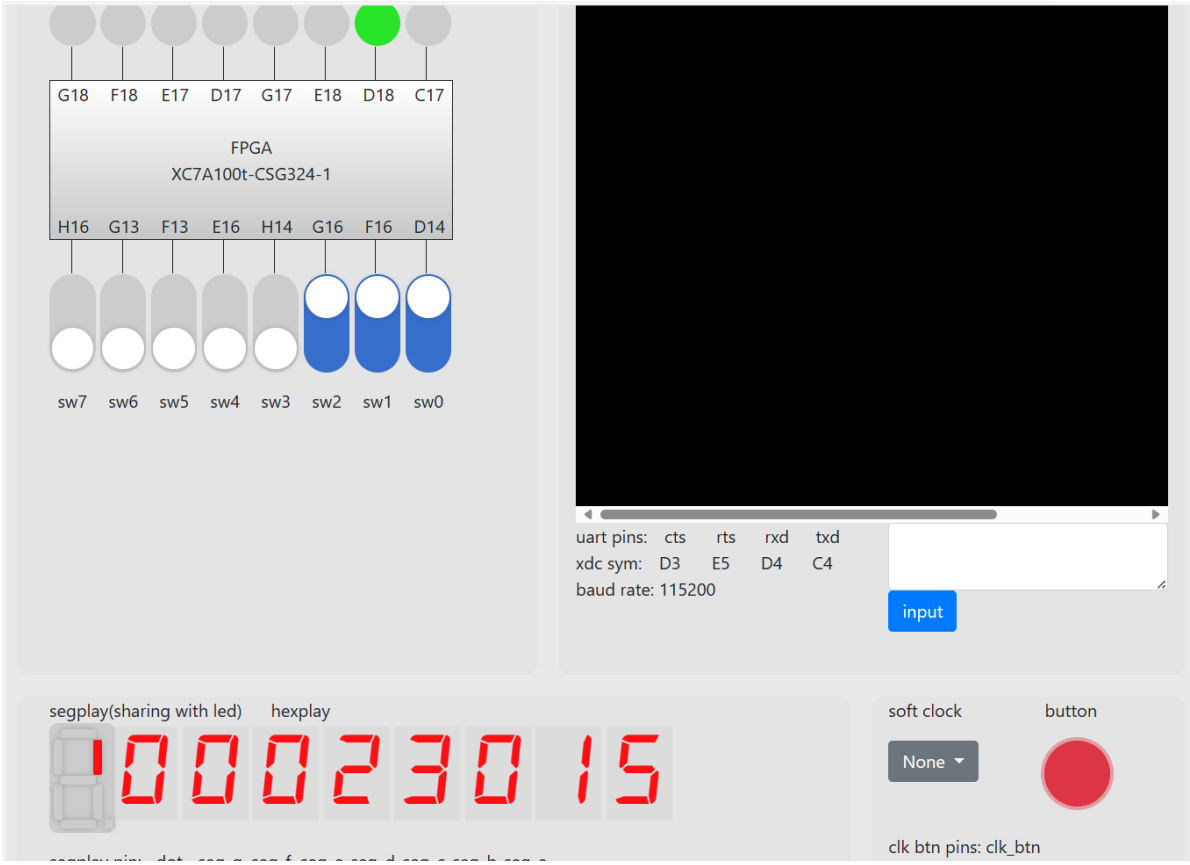
```
    .clk(clk),
    .rst(rst),
    .btn(pos_btn),
    .check_result(check_result),
    .check_start(check_start),
    .timer_en(timer_en),
    .timer_set(timer_set),
    .timer_finish(finish),
    .led_sel(led_sel),
    .seg_sel(seg_sel)
);
always @(*) begin
    en = timer_en;
    set = timer_set;
    start_check = check_start;
end
endmodule
```

上板测试结果如下：

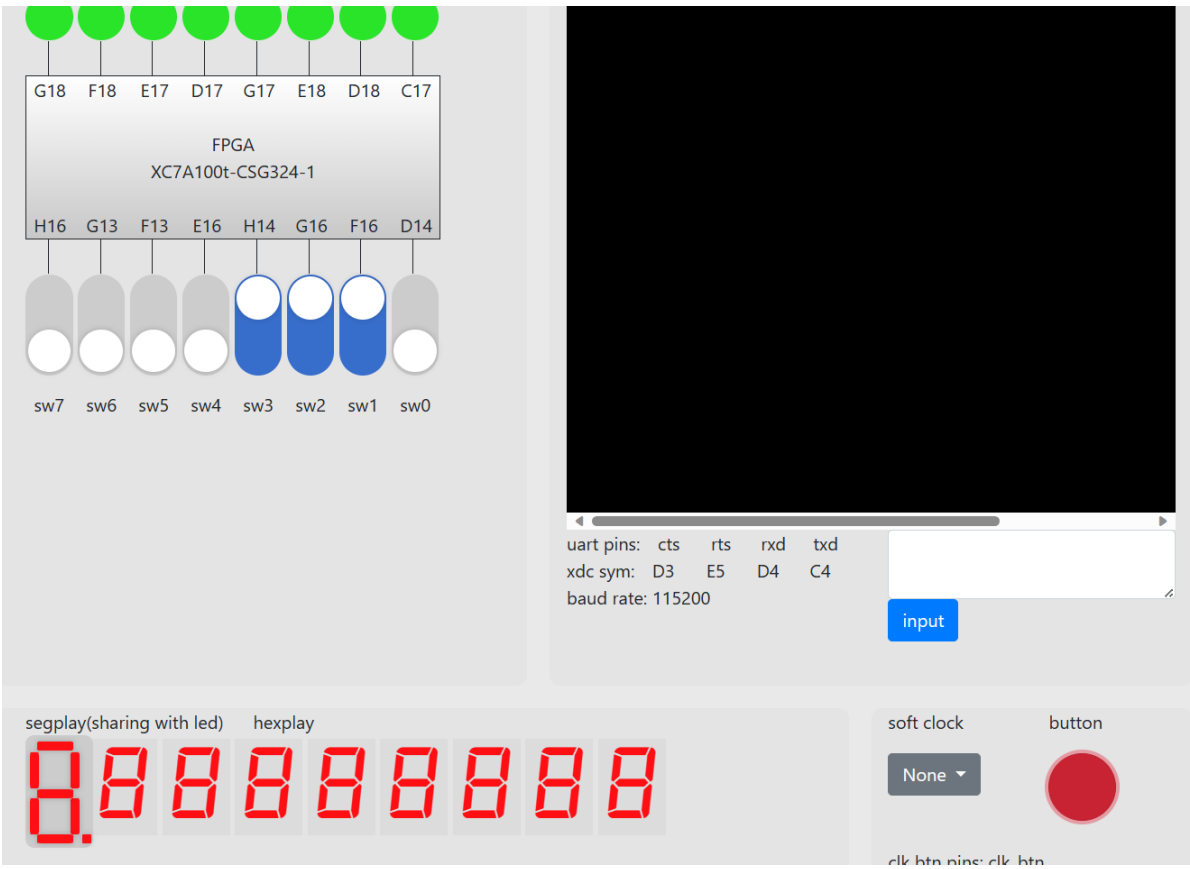
1. 开始界面，LED流水灯，数码管显示倒计时。



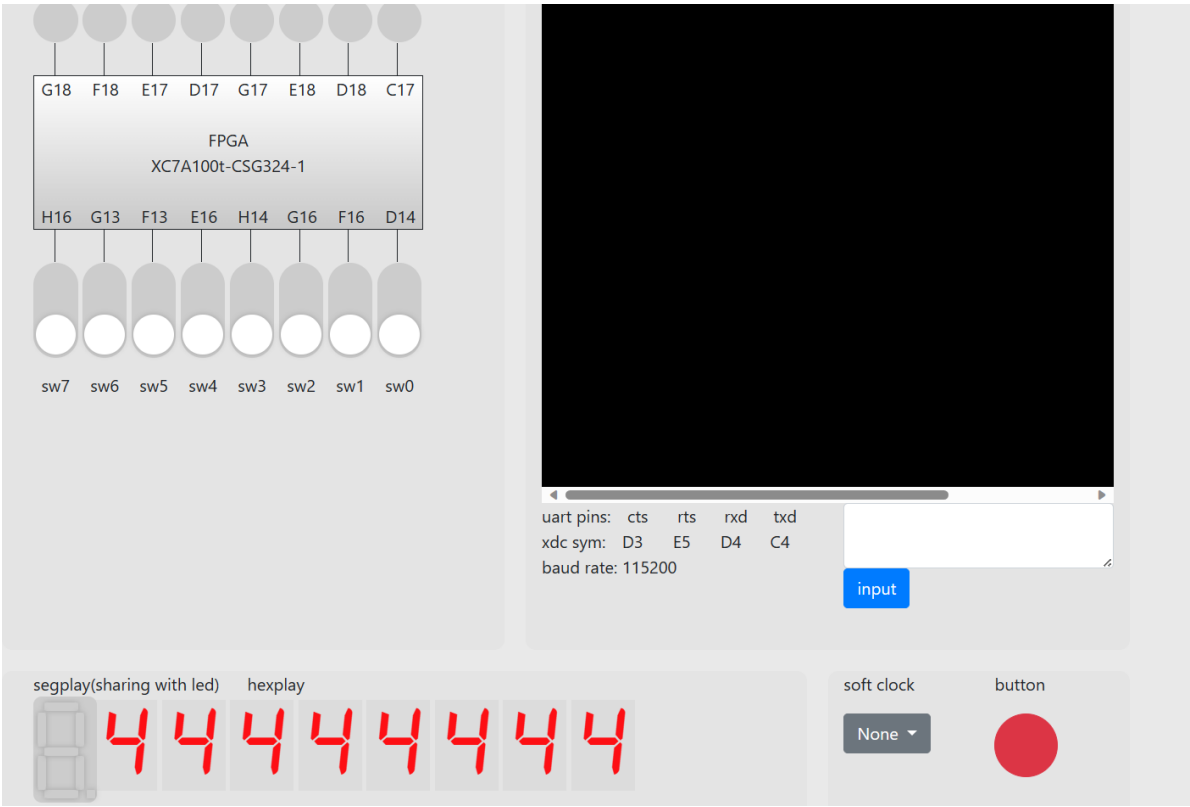
2. 按下012，按下按钮，显示两个数字正确，位置错误（正确数字为123）。



3. 按下123，按下按钮，正确



4. 时间结束



2-A-6：更高级的猜数字小游戏

> **要求：**
>
> 在基础版本上，增加闪烁倒计时，随机生成数字，BCD码显示，串口命令这四个模块。

> **逻辑实现：**
>
> - 闪烁倒计时：通过判断`Timer`模块产生的时间，并通过运用写过的数码管的掩码功能，通过增加一个计数器来计时，实现数码管的闪烁效果。
> - 随机数字生成器，需要在所给的代码上增加使能信号，通过`control`模块的状态判断是否生成随机数字。这里的随机是通过每个周期在数字中循环得到的，即不同时刻产生的数字不同即可实现随即效果。
> - BCD码显示：根据所给的代码，并通过`Timer`模块给出的数字实现BCD码转化。
> - 串口命令：使用了状态机，根据实验文档中所给的状态图，举一反三，共设置了7个状态，用于判断语句并给出相应的信号，传到`control`模块中，进而输出使能信号用于控制程序。

1. 闪烁倒计时（在`Timer`模块中加入使能信号并输出到`Control`模块中）

核心代码如下：

```
assign finish1 = (second > 10);
assign finish2 = (second > 3 && second <= 10);
assign finish3 = (second <= 3);
```

2. 随机数字生成器

核心代码如下：

```
reg [7 : 0] index;
always @(posedge clk) begin
    if (rst)
        index <= 0;
    else if (generate_random) begin
        if (index < 119)
            index <= index + 1;
        else
            index <= 0;
    end
end
always @(posedge clk) begin
    if(random_en)begin
        random_data <= targets[index];
    end
end
```

3. BCD码显示（套用实验文档中的链接所给的代码，运用状态机不断左移判断得出结果），由于代码链接中已经给出，这里不再给出。
4. 串口命令

核心代码如下：

```
localparam WAIT = 4'b0000;
localparam a1 = 4'b0001;
localparam a2 = 4'b0010;
localparam n1 = 4'b0011;
localparam n2 = 4'b0100;
localparam p1 = 4'b0101;
localparam p2 = 4'b0110;
localparam p3 = 4'b0111;
localparam p4 = 4'b1000;
reg [3:0] current_state,next_state;
always @(posedge clk) begin
    if(rst)begin
        current_state <= WAIT;
    end else begin
        current_state <= next_state;
    end
end
always @(*) begin
    case (current_state)
    WAIT:begin
        if(din_vld && din_data == "a")begin
            next_state = a1;
        end else if(din_vld && din_data == "n")begin
            next_state = n1;
        end else if(din_vld && din_data == "p")begin
            next_state = p1;
        end else begin
            next_state = WAIT;
        end
```

```
    end
    a1:begin
        if(din_vld && din_data == ";")begin
            next_state = a2;
        end else if(din_vld && din_data != ";")begin
            if(uart_pause)begin
                next_state = p2;
            end else begin
                next_state = WAIT;
            end
        end else begin
            next_state = a1;
        end
    end
    a2:begin
        if(uart_pause) begin
            next_state = p2;
        end else begin
            next_state = WAIT;
        end
    end
    n1:begin
        if(din_vld && din_data == ";")begin
            next_state = n2;
        end else if(din_vld && din_data != ";")begin
            next_state = WAIT;
        end else begin
            next_state = n1;
        end
    end
    n2:begin
        next_state = WAIT;
    end
    p1:begin
        if(din_vld && din_data == ";")begin
            next_state = p2;
        end else if(din_vld && din_data != ";")begin
            next_state = WAIT;
        end else begin
            next_state = p1;
        end
    end
    p2:begin
        if(din_vld && din_data == "p") begin
            next_state = p3;
        end else if(din_vld && din_data == "a")begin
            next_state = a1;
        end else begin
            next_state = p2;
        end
    end
    p3:begin
        if(din_vld && din_data == ";")begin
            next_state = p4;
        end else if(din_vld && din_data != ";")begin
```

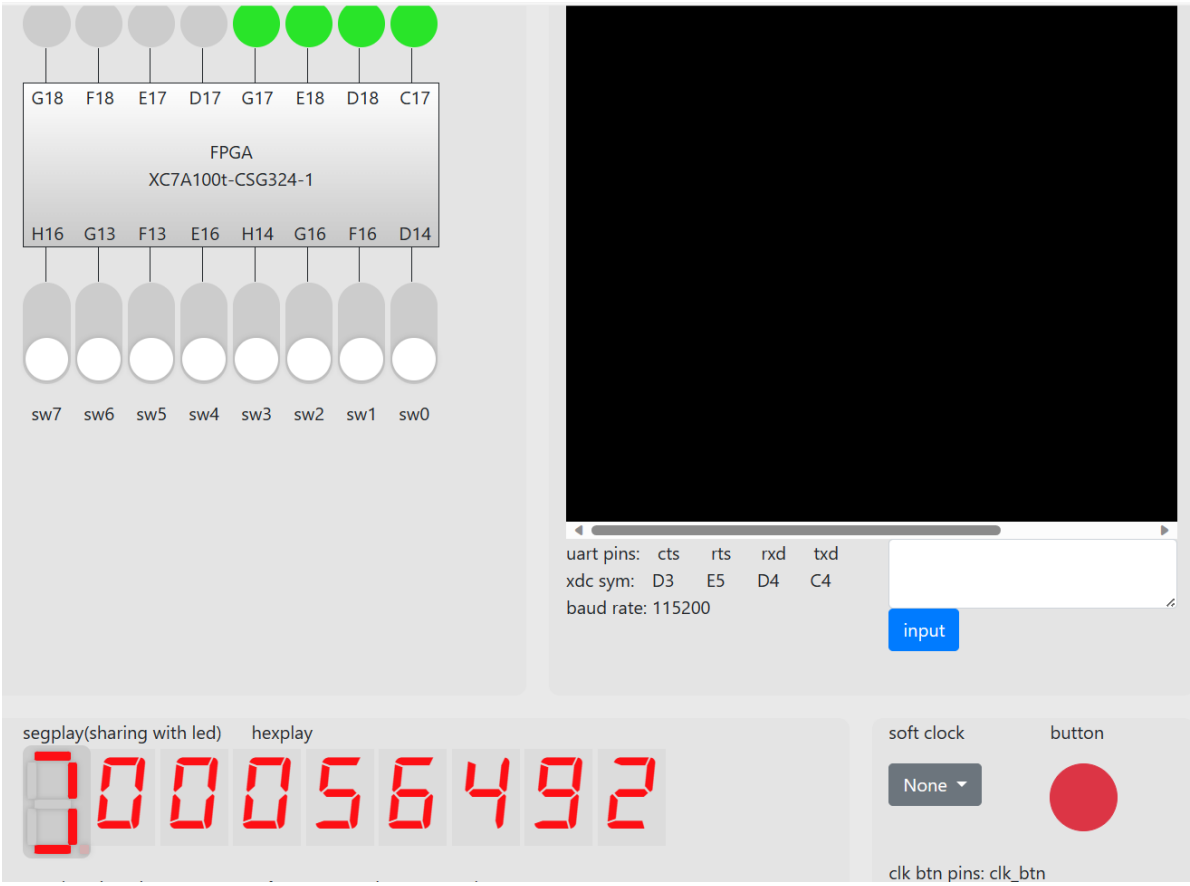
```
            next_state = p3;
        end else begin
            next_state = p3;
        end
    end
    p4:begin
        next_state = WAIT;
    end
    default: next_state = WAIT;
    endcase
end
always @(posedge clk) begin
    case (current_state)
    WAIT:begin
        uart_answer <= 0;
        uart_pause <= 0;
        uart_new <= 0;
    end
    a1:begin
        uart_answer <= 0;
        uart_new <= 0;
        if(uart_pause)begin
            uart_pause <= 1;
        end else uart_pause <= 0;
    end
    a2:begin
        uart_answer <= 1;
        if(uart_pause)begin
            uart_pause <= 1;
        end else uart_pause <= 0;
        uart_new <= 0;
    end
    n1:begin
        uart_answer <= 0;
        uart_pause <= 0;
        uart_new <= 0;
    end
    n2:begin
        uart_answer <= 0;
        uart_pause <= 0;
        uart_new <= 1;
    end
    p1:begin
        uart_answer <= 0;
        uart_pause <= 0;
        uart_new <= 0;
    end
    p2:begin
        uart_answer <= 0;
        uart_pause <= 1;
        uart_new <= 0;
    end
    p3:begin
        uart_answer <= 0;
        uart_pause <= 1;
```

```
            uart_new <= 0;
        end
        p4:begin
            uart_answer <= 0;
            uart_pause <= 0;
            uart_new <= 0;
        end
        endcase
end
```

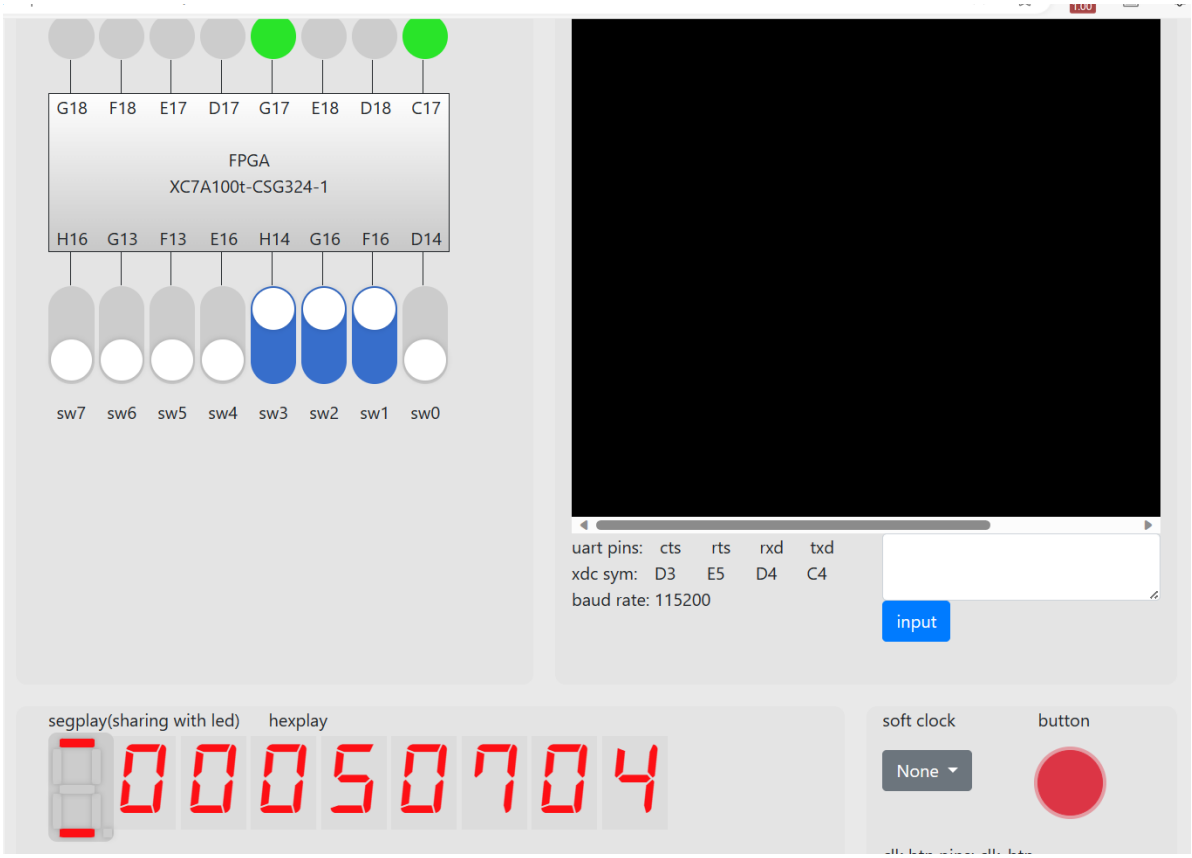
> 还需要在`control`模块中做出相应的改变，相关文件已放进src文件夹中。

上板测试结果如下：

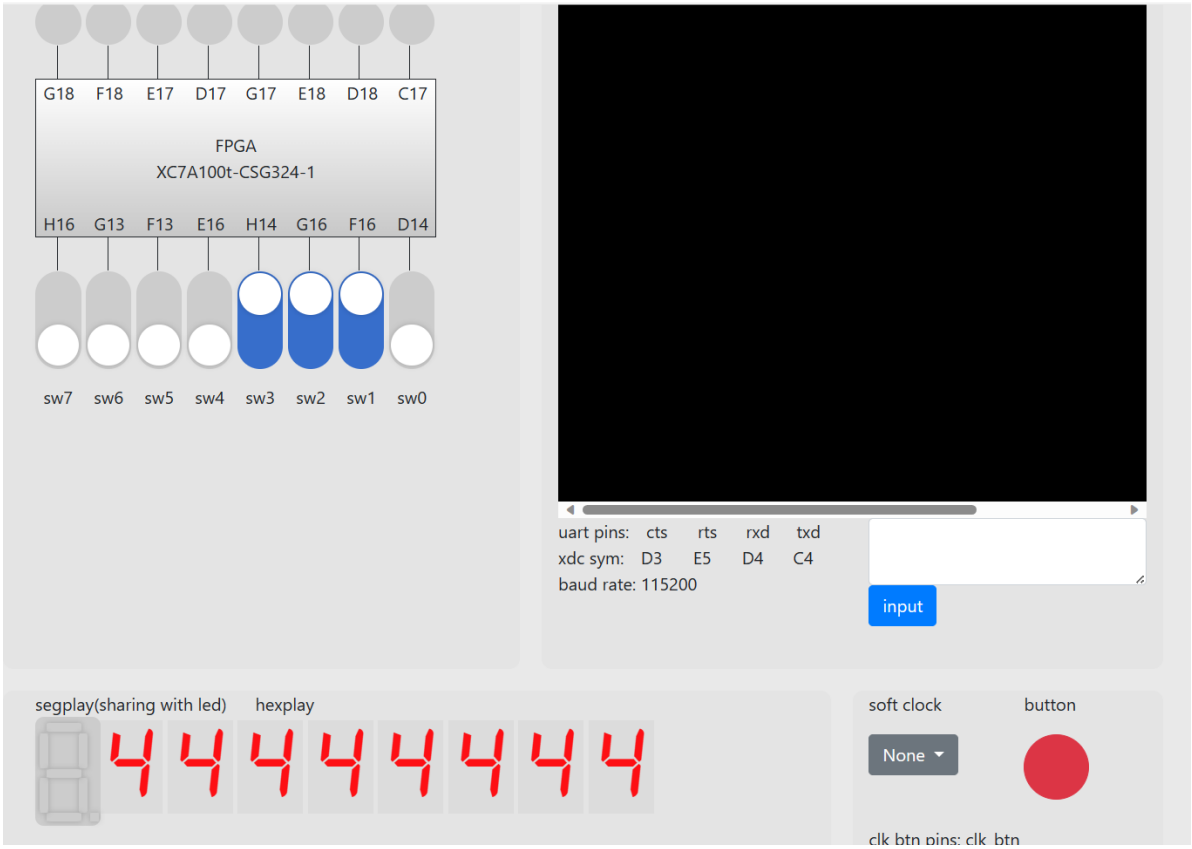
1. 初始页面，正常倒计时，流水灯



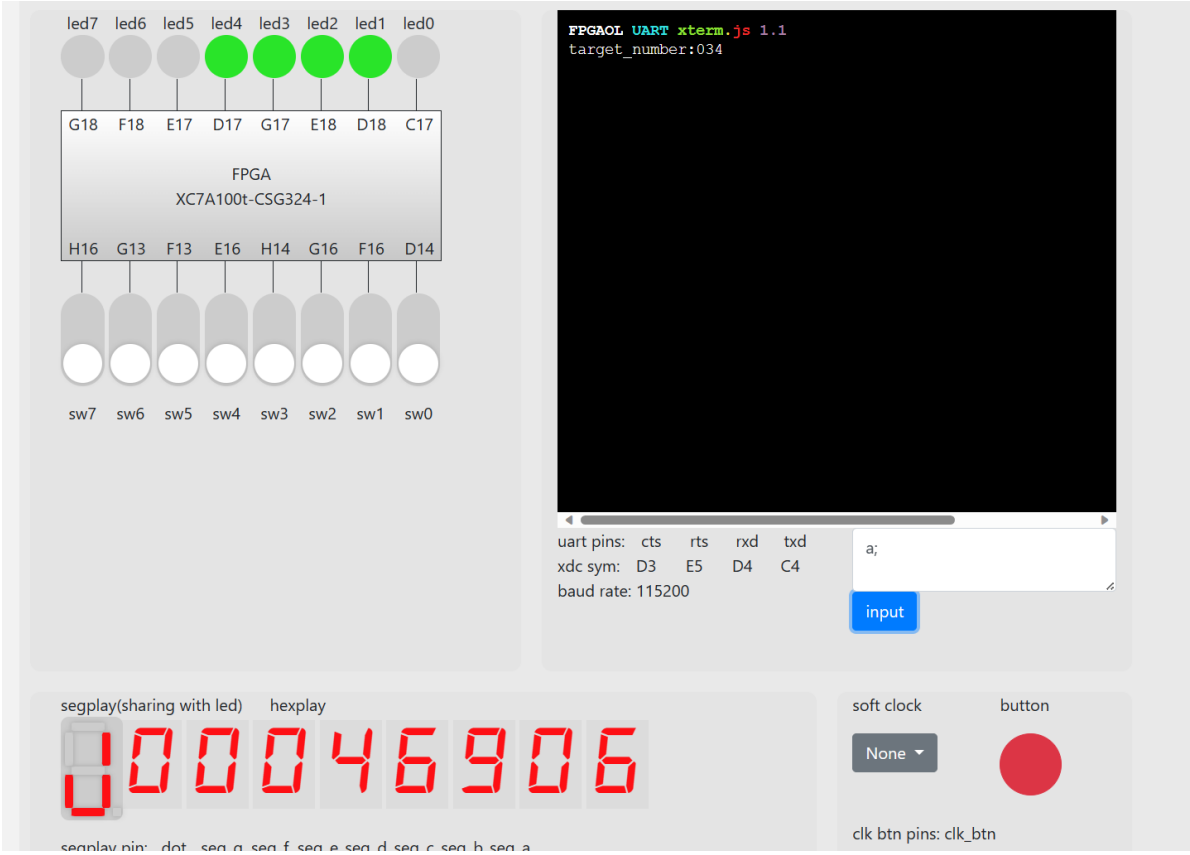
2. 输入数字，显示结果（这是第二轮，即目标是随机生成的数字）



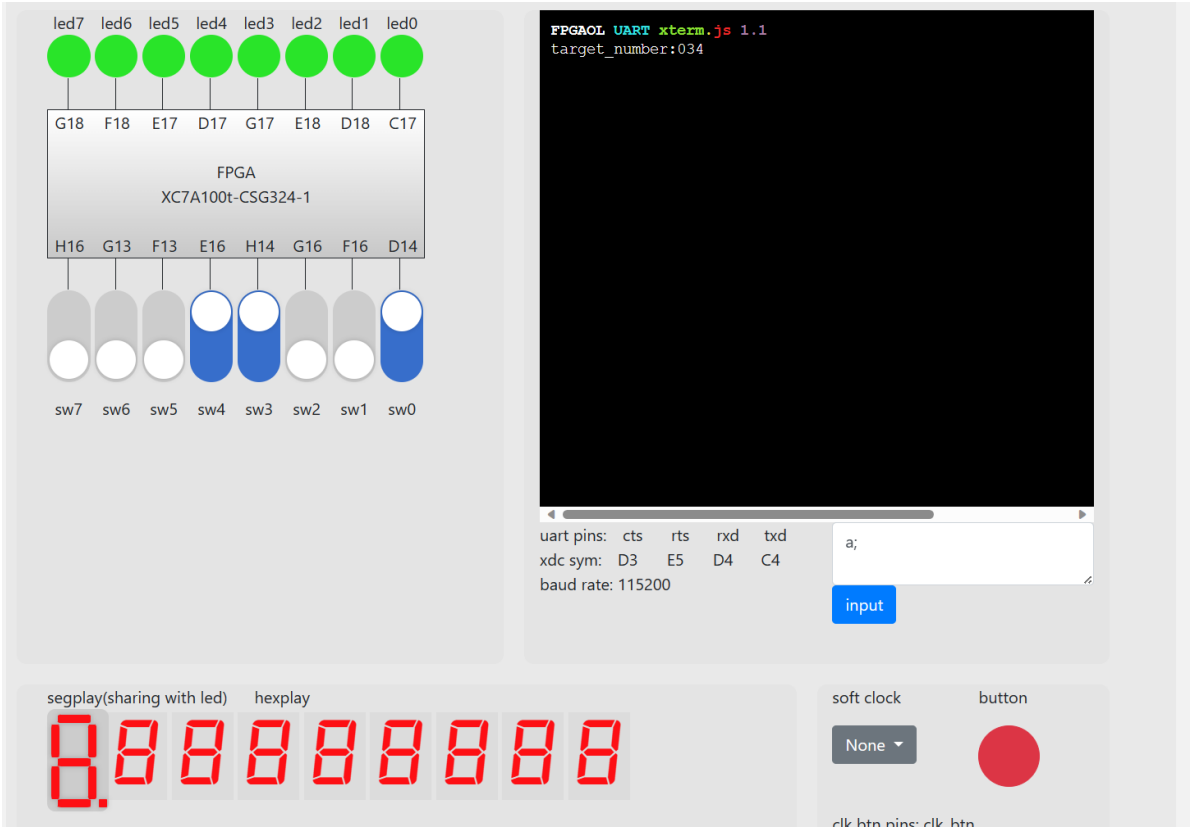
3. 时间耗尽，显示失败



4. 串口命令（其他串口命令已经给助教检查完，由于这里不能做动态展示，所以不做演示）



5. 输入正确，成功



总结与反思

1. 本次实验，对于`verilog`语言有了更深入的了解，同时学会串口通信协议并进行整体协调程序。
2. 对于实验部分，可以根据之前写过的代码，已经实现的功能组装整合形成一个不算简单的小游戏，收获颇丰。

> 3. 可以说，这是我目前学过的课程中收获最多的实验课之一，学会根据仿真调代码，虽然经常濒临崩溃，但是每一次做完实验都有一种满满的成就感，浅浅期待下之后的手搓cpu。最后感谢助教的耐心讲解与解答，一直在帮同学们答疑解惑，助教们辛苦哩!