（自选项目）基于 Verilog 和 FPGA 的数字游戏设计 - 实验报告

- 学号：
- 姓名：

实验目的

1. 理解和掌握有限状态机的原理和设计方法。
2. 掌握数字电路中时序控制的技巧。
3. 发挥个人能力和兴趣，综合本学期所学知识，完成自选的一个题目设计。

实验仪器与平台

- 硬件：DE1-SoC 实验板
- 软件：Altera Quartus Ⅱ 13.1

实验内容和任务

在本次实验中我实现了一个数字游戏，主要考验玩家熟练地进行十进制数和二进制数之间的转换的能力。玩家需要拨动十个开关，以使得开关状态所表示的二进制数与数码管上显示的十进制数相等，从而通过关卡。游戏主要包括以下特性：

- 可自行设定目标分数值和游戏的难度系数
- 每轮游戏需在规定时间内完成（由玩家设定的难度系数决定），超时视为失败
- 高分榜显示游戏纪录
- 设计了各种情形下的灯光效果，提升游戏愉悦感，降低压力

游戏介绍

游戏流程

玩家首先设置目标分数值（即期望完成的关卡数，范围 1 - 99），按下“下一步”，然后设置难度系数（决定每轮允许的游戏时间*，范围 1 - 10），再次按下“下一步”，进入 5 秒钟倒计时准备时间，然后游戏开始。玩家需要拨动开关，使得开关状态所表示的二进制数与右边 4 个数码管上显示的十进制随机数（范围 0 - 1023）相等，然后按下“下一步”提交答案，方可视为正确作答。若超时未正确作答，则游戏失败。若正确作答但未达到目标分数，则显示 `-PASS-` 闪烁动画，然后继续下一关卡；若已经达到目标分数，则游戏成功结束，显示成功动画。游戏完成后可按下“高分榜”按钮查看最高分纪录。

*难度系数如何决定游戏时间：记难度系数为 `d`，则一轮游戏必须在 `10 * d` 秒内完成。难度系数越大，难度实际上越小。

按键功能

各按键的功能如下：

- key3: 下一步
- key2: 显示高分榜
- key1: 返回（会终止正在进行的游戏）
- key0: 重置（重置一切状态，上电时需先按此按钮进行重置）

状态

游戏整体上可以看作一个有限状态机，共有 8 种状态，分别如下：

- 0: 初始状态（欢迎），等待玩家设置目标分数
- 1: 目标分数已设置，等待玩家设置难度系数
- 2: 游戏即将开始（准备状态）
- 3: 游戏进行中，左边两个数码管显示当前分数，右边四个数码管显示本关的随机数
- 4: 通过一个关卡，显示 `-PASS-` 闪烁动画
- 5: 成功完成目标
- 6: 游戏失败
- 7: 高分榜显示

各个状态之间的转换如下：

- 0: 当按下 key3 时，转到状态 1；当按下 key2 时，转到状态 7
- 1: 当按下 key3 时，转到状态 2
- 2: 倒计时 5 秒后自动转到状态 3
- 3:
  - 若开关值与随机数相等，并且 key3 被按下（视为提交操作），则分数 + 1，更新高分纪录
    - 若分数已达到目标，则转到状态 5
    - 若分数未达到目标，则转到状态 4
  - 若超时未正确作答，则转到状态 6
- 4: 显示 `-PASS-` 闪烁动画 3 秒，然后自动转到状态 3
- 任意状态按下 key1 或者 key0 都将转到状态 0

灯光效果

- 在状态 0、状态 1 及状态 7 下，LED 显示为每 0.5 s 向右移动一次的亮点（跑马灯）
- 在状态 2 下，LED 全亮，提醒玩家为游戏做好准备
- 在状态 3 下，LED 是玩家的“血条”，会随着时间的流逝逐渐向左减少，消失时便是超时
- 在状态 4 下，LED 和数码管显示的 `-PASS-` 一起闪烁 3 s（周期 1 s），提示玩家已通过一关
- 在状态 5 下，LED 的显示以 1 s 的周期在 `1010101010` 和 `0101010101` 之间交替，类似于“拉拉队员”的灯光效果，提示玩家成功达到目标
- 在状态 6 下，LED 全灭，提示玩家游戏失败
- 数码管在部分状态下会显示一些英文字母（如 `GOAL`、`HIGH` 等），用于提示输入和展示结果

随机数生成

每个关卡中的数是随机产生的。采用的随机数生成方法如下：存储自上次重置以来已经过去的时钟周期数，然后在按键按下时刻取其 14 ~ 5 位作为 0 ~ 1023 之间的均匀分布随机数。由于时钟周期的流逝相对于用户的操作和感知来说要快得多，用户的按键操作可以认为是在随机的时刻发生的，所以我们可以认为在此时刻取到的是一个（真）随机数。

倒计时

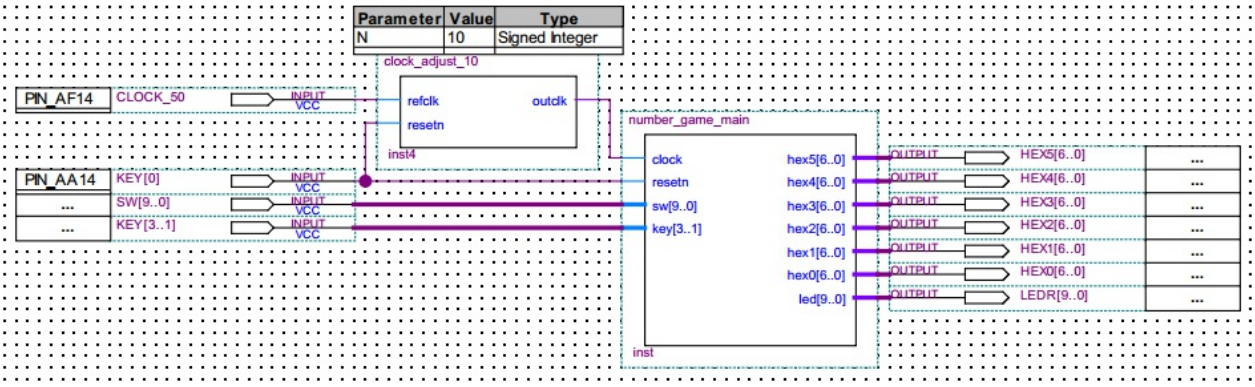
由主模块向倒计时计数器发起一个异步的重置信号来启动倒计时，然后主模块轮询计数器的值，当值变为 0 时说明倒计时已经结束。

调试模式

当调试模式开启时（将 `enable_debug` 参数设置为 1，详见后文代码），直接按下 key3 即可通过关卡（无需开关值正确），方便程序调试。

详细设计

顶层设计图



板载时钟通过 `clock_adjust_10` 调节器将频率调慢 10 倍后作为游戏的主时钟，用于不断地扫描输入和更新状态。

设计代码

```
// Main module of the game.
// States:
//   0 -> initial (welcome), waiting for goal settings
//   1 -> goal set, waiting for difficulty settings
//   2 -> downcounting, will enter game after 5 seconds (preparing)
//   3 -> game in process
//   4 -> pass one level (score + 1), show "-PASS-" flash effect and proceed to next level
//   5 -> successfully finished goal
//   6 -> failed to finish goal
//   7 -> scoreboard display
// Key usage:
//   key3: next (enabled in states 0, 1 and 3)
//   key2: show scoreboard (only enabled in state 0)
//   key1: back (return to state 0, will terminate current game)
//   key0: reset (reset all states including high score record, return to state 0)
module number_game_main(clock, resetn, sw, key, hex5, hex4, hex3, hex2, hex1, hex0, led);
    input         clock, resetn;
    input  [9:0]  sw;
    input  [3:1]  key;
    output [6:0]  hex5, hex4, hex3, hex2, hex1, hex0;
    output [9:0]  led;

    wire          second_clock, flash_clock, pf_status;
    wire   [2:0]  prepare_downcount, pass_downcount;
    wire   [4:0]  progress;
    wire   [6:0]  hex1_goal, hex0_goal;
    wire   [6:0]  hex1_diff, hex0_diff;
    wire   [6:0]  hex3_prepare, hex2_prepare;
    wire   [6:0]  hex5_score, hex4_score, hex3_number, hex2_number, hex1_number, hex0_number;
    wire   [6:0]  hex5_pass, hex4_pass, hex3_pass, hex2_pass, hex1_pass, hex0_pass;
    wire   [6:0]  hex1_sb, hex0_sb;
    wire   [9:0]  random;
    wire   [9:0]  led_wlcm, led_progress, led_pass, led_success;
    wire   [31:0] new_goal, new_difficulty;
    reg           resetn_downcount, downcount_started, key3_last, key2_last;
    reg    [2:0]  state;
    reg    [3:0]  difficulty;
    reg    [6:0]  goal, score, highscore;
    reg    [9:0]  number;

    // Clock ratio for 1s clock and 0.5s clock.
    parameter secclk_N = 5000000;
    parameter halfsecclk_N = 2500000;

    // Sevenseg and LED codes.
    parameter code_off = 7'b1111111; // Sevenseg all off.
    parameter code_hf = 7'b0111111; // '-'
    parameter code_A = 7'b0001000;
    parameter code_C = 7'b1000110;
    parameter code_D = 7'b0100001;
    parameter code_F = 7'b0001110;
    parameter code_G = 7'b1000010;
```

```
    parameter code_H = 7'b0001001;
    parameter code_I = 7'b1111001;
    parameter code_L = 7'b1000111;
    parameter code_O = 7'b1000000;
    parameter code_P = 7'b0001100;
    parameter code_S = 7'b0010010;
    parameter code_U = 7'b1000001;
    parameter led_off = 10'b0000000000; // LED all off.
    parameter led_on = 10'b1111111111; // LED all on.

    // Debug mode switch.
    parameter enable_debug = 0;

    wire key3 = key[3];
    wire key2 = key[2];
    wire key1 = key[1];

    // Generate 1s clock and 0.5s clock.
    clock_adjust sec_clk(clock, resetn_downcount, secclk_N, second_clock);
    clock_adjust halfsec_clk(clock, resetn, halfsecclk_N, flash_clock);

    // Random number generator.
    rand10 rand_gen(clock, resetn, random);

    // Welcome LED. (For states 0, 1 and 7)
    display_welcome disp_welcome(flash_clock, resetn, led_wlcm);

    // Module instances for state 0.
    min_max_threshold input_goal(1, 99, {22'b0, sw}, new_goal); // Goal should be in range [1, 99].
    display_goal disp_goal(new_goal, hex1_goal, hex0_goal);

    // Module instances for state 1.
    min_max_threshold input_diff(1, 10, {22'b0, sw}, new_difficulty); // Difficulty should be in range [1, 10].
    display_difficulty disp_diff(new_difficulty, hex1_diff, hex0_diff);

    // Module instances for state 2.
    downcount5 prep_downcount(second_clock, resetn_downcount, prepare_downcount);
    display_prepare disp_prep(prepare_downcount, hex3_prepare, hex2_prepare);

    // Module instances for state 3.
    display_score disp_score(score, hex5_score, hex4_score);
    display_number disp_num(number, hex3_number, hex2_number, hex1_number, hex0_number);
    downcount_difficulty game_downcount(second_clock, resetn_downcount, difficulty, progress);
    display_progress disp_prog(progress, led_progress);

    // For state 4. (pf_status: turn sevensegs and LEDs on/off)
    downcount3_flash pass_flash_downcount(flash_clock, resetn_downcount, pass_downcount, pf_status);
    assign hex5_pass = pf_status ? code_hf : code_off;
    assign hex4_pass = pf_status ? code_P : code_off;
    assign hex3_pass = pf_status ? code_A : code_off;
    assign hex2_pass = pf_status ? code_S : code_off;
    assign hex1_pass = pf_status ? code_S : code_off;
    assign hex0_pass = pf_status ? code_hf : code_off;
    assign led_pass = pf_status ? led_on: led_off;

    // For state 5.
    display_success led_succ(flash_clock, resetn, led_success);

    // For state 7.
    display_scoreboard disp_sb(highscore, hex1_sb, hex0_sb);

    // Multiplexers to select output for sevensegs and LEDs according to current state.
    mux8x7 select_hex5(code_G, code_D, code_hf, hex5_score, hex5_pass, code_hf, code_hf, code_H, state, hex5);
    mux8x7 select_hex4(code_O, code_I, code_hf, hex4_score, hex4_pass, code_S, code_F, code_I, state, hex4);
    mux8x7 select_hex3(code_A, code_F, hex3_prepare, hex3_number, hex3_pass, code_U, code_A, code_G, state, hex3);
    mux8x7 select_hex2(code_L, code_F, hex2_prepare, hex2_number, hex2_pass, code_C, code_I, code_H, state, hex2);
    mux8x7 select_hex1(hex1_goal, hex1_diff, code_hf, hex1_number, hex1_pass, code_C, code_L, hex1_sb, state, hex1);
    mux8x7 select_hex0(hex0_goal, hex0_diff, code_hf, hex0_number, hex0_pass, code_hf, code_hf, hex0_sb, state, hex0);
    mux8x10 select_led(led_wlcm, led_wlcm, led_on, led_progress, led_pass, led_success, led_off, led_wlcm, state, led);

    always @(posedge clock or negedge key1 or negedge resetn) begin
        if (!resetn || !key1) begin // reset or back
            state <= 0;
            score <= 0;
```

```
            goal <= 1;
            difficulty <= 1;
            downcount_started <= 0;
            resetn_downcount <= 1;
            key3_last <= 0;
            key2_last <= 0;

            if (!resetn)
                highscore <= 0;
        end else begin // posedge of clock
            case (state) // state machine
                0: begin
                    if (!key3 && key3_last) begin // key3 pressed, store goal and proceed to state 1
                        goal <= new_goal[6:0];
                        state <= 1;
                    end else if (!key2 && key2_last) // key2 pressed, show scoreboard (goto state 7)
                        state <= 7;
                end
                1: begin
                    if (!key3 && key3_last) begin // key3 pressed, store difficulty and proceed to state 2
                        difficulty <= new_difficulty[3:0];
                        state <= 2;
                    end
                end
                2: begin
                    // The following 5 lines of code generates a reset signal to start the prepare downcounter.
                    if (!resetn_downcount) begin
                        resetn_downcount <= 1;
                    end else if (!downcount_started) begin
                        downcount_started <= 1;
                        resetn_downcount <= 0;
                    end else if (prepare_downcount == 0) begin // Downcount ended, launch game.
                        state <= 3;
                        number <= random; // Fetch a random number from the random number generator.
                        downcount_started <= 0;
                    end
                end
                3: begin
                    // Start the life progress downcounter.
                    if (!resetn_downcount) begin
                        resetn_downcount <= 1;
                    end else if (!downcount_started) begin
                        downcount_started <= 1;
                        resetn_downcount <= 0;
                    end else if (progress == 0) begin // Life dropped to 0, goto state 6 (fail).
                        state <= 6;
                    end else if (!key3 && key3_last && (sw == number || enable_debug)) begin
                        // Pass current level, if key3 is pressed (to submit), and switch equals the number.
                        // When debug mode is enabled, pressing key3 will pass current level directly.
                        score <= score + 1; // Increment score.
                        highscore <= score + 1 > highscore ? (score + 1) : highscore; // Record highscore.
                        downcount_started <= 0;
                        state <= score + 1 >= goal ? 5 : 4; // If finished goal, goto state 5, else goto state 4.
                    end
                end
                4: begin
                    // Start the pass flash downcounter.
                    if (!resetn_downcount) begin
                        resetn_downcount <= 1;
                    end else if (!downcount_started) begin
                        downcount_started <= 1;
                        resetn_downcount <= 0;
                    end else if (pass_downcount == 0) begin // Downcount ended, goto state 3 (new level).
                        state <= 3;
                        number <= random; // Fetch a new random number.
                        downcount_started <= 0;
                    end
                end
            endcase

            // Store last states of key3 and key2 (resolve long pressing problem).
            key3_last <= key3;
            key2_last <= key2;
        end
```

```
    end
endmodule
```

```
// Generate a clock whose period is N times the period of the reference clock.
module clock_adjust(refclk, resetn, N, outclk);
    input             refclk, resetn;
    input      [31:0] N;
    output reg        outclk;
    reg        [31:0] counter;

    initial begin
        counter <= 0;
        outclk <= 0;
    end

    always @(posedge refclk or negedge resetn) begin
        if (!resetn) begin
            counter <= 0;
            outclk <= 0;
        end
        else begin
            if (counter >= N / 2 - 1) begin
                counter <= 0;
                outclk <= ~outclk;
            end
            else
                counter <= counter + 1;
        end
    end
endmodule

// Generate a clock whose period is 10 times the period of the reference clock.
module clock_adjust_10(refclk, resetn, outclk);
    input             refclk, resetn;
    output reg        outclk;
    reg        [31:0] counter;
    parameter N = 10;

    initial begin
        counter <= 0;
        outclk <= 0;
    end

    always @(posedge refclk or negedge resetn) begin
        if (!resetn) begin
            counter <= 0;
            outclk <= 0;
        end
        else begin
            if (counter >= N / 2 - 1) begin
                counter <= 0;
                outclk <= ~outclk;
            end
            else
                counter <= counter + 1;
        end
    end
endmodule
```

```
// Store the number of cycles elapsed from last reset.
module cycles_counter(clock, resetn, counter);
    input             clock, resetn;
    output reg [31:0] counter;

    initial begin
        counter <= 0;
    end
```

```
    always @(posedge clock or negedge resetn) begin
        if (!resetn)
            counter <= 0;
        else
            counter <= counter + 1;
    end
endmodule
```

```
// Generate a 10bit random number.
module rand10(clock, resetn, random);
    input         clock, resetn;
    output [9:0]  random;
    wire   [31:0] counter;

    assign random = counter[14:5]; // Strip last 5 bits to get a more evenly distributed result.

    cycles_counter cc(clock, resetn, counter);
endmodule
```

```
// Show a decimal digit on a sevenseg.
module sevenseg_decimal(data, ledsegments);
    input      [3:0] data;
    output reg [6:0] ledsegments;

    always @(data)
        case (data)
            // gfe_dcba -> 654_3210
            // 1 -> off, 0 -> on
            0: ledsegments = 7'b100_0000;
            1: ledsegments = 7'b111_1001;
            2: ledsegments = 7'b010_0100;
            3: ledsegments = 7'b011_0000;
            4: ledsegments = 7'b001_1001;
            5: ledsegments = 7'b001_0010;
            6: ledsegments = 7'b000_0010;
            7: ledsegments = 7'b111_1000;
            8: ledsegments = 7'b000_0000;
            9: ledsegments = 7'b001_0000;
            default: ledsegments = 7'b111_1111; // All off on other conditions.
        endcase
endmodule
```

```
module display_welcome(flash_clock, resetn, led);
    input        flash_clock, resetn;
    output [9:0] led;
    reg    [3:0] counter;

    // A red spot moving right.
    assign led = 1 << (9 - counter);

    always @(posedge flash_clock or negedge resetn)
        if (!resetn)
            counter <= 0;
        else
            counter <= counter >= 9 ? 0 : counter + 1;
endmodule

module display_goal(goal, hex1, hex0);
    input  [6:0] goal;
    output [6:0] hex1, hex0;

    sevenseg_decimal disp_g_1(goal / 10, hex1);
    sevenseg_decimal disp_g_0(goal % 10, hex0);
endmodule
```

```
module display_difficulty(difficulty, hex1, hex0);
    input  [3:0] difficulty;
    output [6:0] hex1, hex0;

    sevenseg_decimal disp_d_1(difficulty / 10, hex1);
    sevenseg_decimal disp_d_0(difficulty % 10, hex0);
endmodule

module display_prepare(downcount, hex3, hex2);
    input  [2:0] downcount;
    output [6:0] hex3, hex2;

    sevenseg_decimal disp_p_3(0, hex3);
    sevenseg_decimal disp_p_2(downcount, hex2);
endmodule

module display_score(score, hex5, hex4);
    input  [6:0] score;
    output [6:0] hex5, hex4;

    sevenseg_decimal disp_s_5(score / 10, hex5);
    sevenseg_decimal disp_s_4(score % 10, hex4);
endmodule

module display_number(number, hex3, hex2, hex1, hex0);
    input  [9:0] number;
    output [6:0] hex3, hex2, hex1, hex0;

    sevenseg_decimal disp_n_3(number / 1000, hex3);
    sevenseg_decimal disp_n_2(number % 1000 / 100, hex2);
    sevenseg_decimal disp_n_1(number % 100 / 10, hex1);
    sevenseg_decimal disp_n_0(number % 10, hex0);
endmodule

module display_progress(progress, led);
    input  [4:0] progress;
    output [9:0] led;

    // A life progress bar decaying from 10 to 0 (moving left).
    assign led = ~((1 << (10 - progress)) - 1);
endmodule

module display_success(flash_clock, resetn, led);
    input            flash_clock, resetn;
    output reg [9:0] led;

    // LED flashing between '1010101010' and '0101010101'.
    always @(posedge flash_clock or negedge resetn)
        if (!resetn)
            led <= 10'b1010101010;
        else
            led <= ~led;
endmodule

module display_scoreboard(score, hex1, hex0);
    input  [6:0] score;
    output [6:0] hex1, hex0;

    sevenseg_decimal disp_sb_1(score / 10, hex1);
    sevenseg_decimal disp_sb_0(score % 10, hex0);
endmodule
```

```
// Adjust a value to [min, max] range.
module min_max_threshold(min, max, vin, vout);
    input  [31:0] min, max, vin;
    output [31:0] vout;

    assign vout = vin > min ? (vin < max ? vin : max) : min;
endmodule
```

```
// Downcount 3 seconds and flash sevensegs and LEDs every second.
module downcount3_flash(halfsecclk, resetn, counter, status);
    input            halfsecclk, resetn;
    output reg [2:0] counter;
    output           status;

    // Turn on/off sevensegs and LEDs.
    assign status = !(counter % 2);

    always @(posedge halfsecclk or negedge resetn) begin
        if (!resetn)
            counter <= 6;
        else if (counter > 0)
            counter <= counter - 1;
    end
endmodule

// Downcount 5 seconds.
module downcount5(secclk, resetn, counter);
    input            secclk, resetn;
    output reg [2:0] counter;

    always @(posedge secclk or negedge resetn) begin
        if (!resetn)
            counter <= 5;
        else if (counter > 0)
            counter <= counter - 1;
    end
endmodule

// Downcount 10 * difficulty seconds, output life progress (10 - 0).
module downcount_difficulty(secclk, resetn, difficulty, progress);
    input             secclk, resetn;
    input      [3:0]  difficulty;
    output reg [4:0]  progress;
    reg        [31:0] counter;

    always @(posedge secclk or negedge resetn) begin
        if (!resetn) begin
            counter <= 10 * difficulty;
            progress <= 10;
        end else if (counter > 0) begin
            counter <= counter - 1;
            progress <= (counter - 1 + difficulty - 1) / difficulty; // ceil((counter - 1) / difficulty)
        end
    end
endmodule
```

```
module mux8x7(a0, a1, a2, a3, a4, a5, a6, a7, s, y);
    input      [6:0] a0, a1, a2, a3, a4, a5, a6, a7;
    input      [2:0] s;
    output reg [6:0] y;

    always @(*)
        case (s)
            3'b000: y = a0;
            3'b001: y = a1;
            3'b010: y = a2;
            3'b011: y = a3;
            3'b100: y = a4;
            3'b101: y = a5;
            3'b110: y = a6;
            3'b111: y = a7;
        endcase
endmodule

module mux8x10(a0, a1, a2, a3, a4, a5, a6, a7, s, y);
    input      [9:0] a0, a1, a2, a3, a4, a5, a6, a7;
    input      [2:0] s;
    output reg [9:0] y;
```

```
    always @(*)
        case (s)
            3'b000: y = a0;
            3'b001: y = a1;
            3'b010: y = a2;
            3'b011: y = a3;
            3'b100: y = a4;
            3'b101: y = a5;
            3'b110: y = a6;
            3'b111: y = a7;
        endcase
endmodule
```

实验总结

实验结果

实验代码经过编译综合，载入到开发板后，能正常完成预期的数字游戏功能，各项功能均实测通过。

一些总结和拓展

1. 在本次实验中我借鉴了 CPU 设计中对多路选择器的巧妙使用。对于多路选择器没有选择的那些值，不管其对应的模块（与当前状态无关）在进行何种胡乱的计算，我们都不用关心，因为多路选择器按照当前状态选择对应的值，其他值对输出结果是没有影响的。代码中对数码管和 LED 输出的选择就体现了这一点。
2. 实际上本次实验所编写的 `cycles_counter` 计数器和 `downcount` 倒计时模块也可用于 CPU 的 I/O 扩展。我们可以把它们分别看作是获取当前时间的设备和完成时延的设备，在虚拟地址空间中为它们分配地址，这样汇编代码便可用 `load` 和 `store` 指令访问它们，以在汇编程序中实现获取时间和完成时延的功能。汇编层甚至可以把它们封装成“系统调用” `time()` 和 `sleep()`。
3. 产生（伪）随机数还可使用 LFSR（线性反馈移位寄存器）或者更好的梅森旋转算法等，不过本实验中直接使用时钟周期数也是一种行之有效的方法。

经验教训

1. 若有信号不稳定的情况，可能是一个时钟周期内组合逻辑部分的信号还没有稳定地扩散完成，可以调慢时钟来解决这种问题。
2. 为了使按键的“按下”（即边沿）触发操作而不是按键的“按住”（即值）触发操作，我们应当同时检查按键的当前状态和上次的状态。

感受

在本次实验中，我综合本学期所学知识，发挥个人能力和兴趣，实现了一个基于 Verilog 和 FPGA 的数字游戏，在实际操作中更清晰地理解了有限状态机的设计以及硬件的时序控制问题，并联系 CPU 设计的知识进行了进一步的思考，为本课程的实验设计画上了一个圆满的句号。