基于Verilog和FPGA的记忆游戏设计 - 实验报告

- 学号：517021910653
- 姓名：王祖来

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

实验仪器和平台

- DE1-SoC实验板
- QuartusII 13.1
- ModelSim 10.1

实验内容和任务

在本次实验中，我基于Verilog和FPGA设计了一个记忆游戏。该游戏主要考验玩家短时间内的记忆能力。游戏开始后，玩家需要尽可能多的记住LED的显示序列。LED序列显示完毕后，玩家根据自己的记忆，通过开关输入对应的LED序列，如果该序列在显示阶段出现过，则玩家获得分数，否则失去分数。每一轮显示和输入过后，玩家可以自主选择继续游戏或者重启游戏。

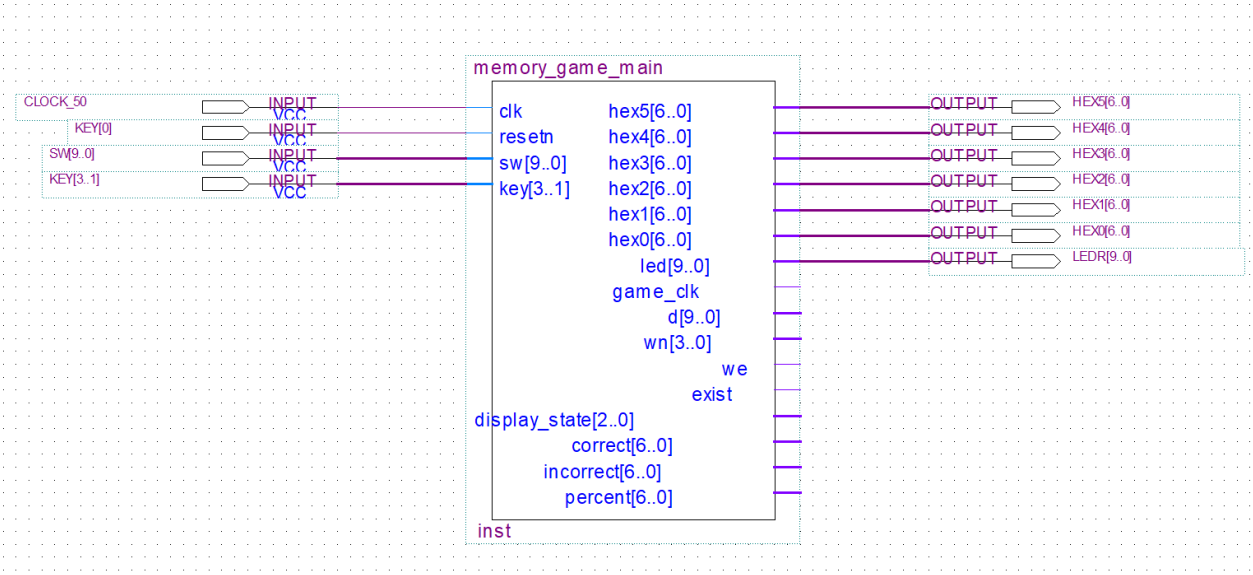
游戏操作说明

- key[3]：游戏开始按键
  - 长按该键1秒左右，游戏开始。七段数码管显示“GOING”字样，LED开始显示随机的序列
- key[2]：玩家确认输入按键
  - 此时10个开关的状态确认为玩家的输入，如果10个开关的排列在上一轮LED显示阶段出现过，则认为玩家记住了该序列，正确数加1，得分上升。
- key[0]：重启游戏按键
  - 分数清零，重启游戏
- sw[9..0]：玩家输入阶段，10个开关的排列表示为玩家的输入。10个开关的排列会对应的显示在LED上。

- led[9..0]：LED显示阶段，随机产生序列，要求玩家记忆尽可能多的序列。玩家输入阶段，显示玩家在开关上的对应输入。
- hex5/hex4：上一次开始游戏起玩家的正确个数
- hex3/hex2：上一次开始游戏起玩家的错误个数
- hex1/hex0：上一次开始游戏起玩家的得分
    - 正确个数上升时得分上升
    - 错误个数上升时得分下降

设计过程

顶层设计



板载50MHz时钟、3个按键和10个开关接入主模块。剩下的1个按键作为游戏中各个模块的复位按键。主模块输出到6个七段数码管和10个发光二极管。主模块的其他输出端口不接到输出的引脚上，只用于仿真验证时查看波形。

主模块

```
// key[3]: game start button
// key[2]: player input button
// key[1]: next stage button
// key[0]: resetn
// sw[9:0]: player input
// led[9:0]: random sequence generated, which should be remembered and inputed
by players
// hex: player scores (correct number, incorrect number, percent correct)
```

```
module memory_game_main(clk, resetn, sw, key, hex5, hex4, hex3, hex2, hex1,
hex0, led, game_clk,
                d, wn, we, exist, display_state, correct, incorrect, percent);
  input         clk, resetn;
  input  [9:0]  sw;
  input  [3:1]  key;
  output [6:0]  hex5, hex4, hex3, hex2, hex1, hex0;
  output [9:0]  led;
  output      game_clk, we, exist;
  output [9:0]  d;
  output [3:0]  wn;
  output [2:0]  display_state;
  output  [6:0] correct, incorrect, percent;

  wire      game_clk, display_clk, inc_point, dec_point, we, exist;
  wire [2:0]  display_state;
  wire [9:0]  d, led_flow, led_input, led;
  wire [3:0]  wn;
  wire [6:0]  hex5_cntdn, hex4_cntdn, hex3_cntdn, hex2_cntdn, hex1_cntdn,
hex0_cntdn;
  wire [6:0]  hex5_score, hex4_score, hex3_score, hex2_score, hex1_score,
hex0_score;

  assign display_state = 0;
  assign led = we == 1 ? led_flow : led_input;
  assign hex5 = we == 1 ? hex5_cntdn : hex5_score;
  assign hex4 = we == 1 ? hex4_cntdn : hex4_score;
  assign hex3 = we == 1 ? hex3_cntdn : hex3_score;
  assign hex2 = we == 1 ? hex2_cntdn : hex2_score;
  assign hex1 = we == 1 ? hex1_cntdn : hex1_score;
  assign hex0 = we == 1 ? hex0_cntdn : hex0_score;

  clock_generator game_clk_gen(clk, resetn, game_clk);
  game_start game_st(key[3], clk, game_clk, resetn, d, wn, we, led_flow,
              hex5_cntdn, hex4_cntdn, hex3_cntdn, hex2_cntdn, hex1_cntdn,
hex0_cntdn);
  number_mem num_mem(key[2], sw, d, wn, we, clk, resetn, exist, led_input);
  display_controller disp_cont(key[2], exist, display_state, clk, resetn,
                      hex5_score, hex4_score, hex3_score,
                      hex2_score, hex1_score, hex0_score,
                      correct, incorrect, percent);

endmodule
```

游戏设计主模块由4个子模块组成。clock_generator模块根据板载时钟输入产生低频率的游戏时钟。游戏时钟在其他模块中被用于显示玩家可以看清的LED序列。game_start模块为玩家按下游戏开始开关后显示LED序列并显示"GOING"字样的模块。number_mem模块用于显示阶段记忆出现过的LED序列以及玩家输入阶段比较玩家输入和曾经出现过的序列。display_controller为记录并显示玩家得分的模块。由于在游戏的不同阶段，发光二极管和七段数码管被用于不同模块显示不同内容，主模块通过三目运算符

实现了简单的二路选择器。

游戏时钟生成模块

```
module clock_generator (clock, resetn, game_clk);
  input   clock, resetn;
  output  game_clk;

  reg   game_clk;
  reg   display_clk;
  reg [31:0] game_cnt;
  reg [31:0] display_cnt;
  parameter GAME_CYCLE = 50000000;  // the number of cycles of 1 game_clk on
board clock

  initial begin
    game_clk <= 0;
    game_cnt <= 0;
  end

  always @ (posedge clock or negedge resetn)
    begin
      if (!resetn) begin
        game_clk <= 0;
        game_cnt <= 0;
      end
      else begin
        if (game_cnt >= GAME_CYCLE/2 - 1) begin
          game_cnt <= 0;
          game_clk <= ~game_clk;
        end
        else begin
          game_cnt <= game_cnt + 1;
        end
      end
    end
endmodule
```

该模块通过一个50M的计数器和板载时钟生成了一个周期为1s的游戏时钟并输出。

游戏开始模块

```
// display some numbers on led, and remember them
module game_start (start_key, clk, game_clk, resetn, d, wn, we, led,
            hex5, hex4, hex3, hex2, hex1, hex0);
  input       start_key, clk, game_clk, resetn;
```

```
output [9:0]  led, d;
output [3:0]  wn;
output        we;
output [6:0]  hex5, hex4, hex3, hex2, hex1, hex0;
reg  [6:0]  hex5, hex4, hex3, hex2, hex1, hex0;
reg  [3:0]  display_num;
parameter DISPLAY_CYCLE = 10;  // display times in one round of game

reg  [9:0]  led;
reg  [3:0]  wn;
reg  [9:0]  d;
reg         we, start, finish;
wire   [9:0]  randnum;

parameter code_G = 7'b1000010;
parameter code_O = 7'b1000000;
parameter code_I = 7'b1111001;
parameter code_N = 7'b1001000;
parameter code_none = 7'b1111111;

random_generator rand_gen(clk, game_clk, resetn, randnum);

initial begin
  display_num <= 0;
  wn <= 0;
  d <= 0;
  we <= 0;
  start <= 0;
  led <= 0;
end

// put number to led and memory
always @ (posedge game_clk)
  begin
    if (!resetn) begin
      display_num <= 0;
      wn <= 0;
      d <= 0;
      we <= 0;
      start <= 0;
    end
    else if (start == 1) begin
      if (display_num < DISPLAY_CYCLE) begin
        led <= randnum;
        wn <= display_num;
        d <= randnum;
        display_num <= display_num + 1;
      end
      else begin
```

```
            display_num <= 0;
            wn <= 0;
            d <= 0;
            we <= 0;
            start <= 0;
          end
        end
        else if (start == 0 && !start_key) begin
          hex5 <= code_none;
          hex4 <= code_G;
          hex3 <= code_O;
          hex2 <= code_I;
          hex1 <= code_N;
          hex0 <= code_G;
          start <= 1;
          we <= 1;
          display_num <= 0;
          wn <= 0;
          d <= 0;
        end
      end
endmodule
```

该模块在每一个游戏时钟上升沿判断玩家是否按下游戏开始按键。如果按下了游戏开始按键，则开始显示随机的LED序列并计数。同时显示“GOING“字样。当显示序列个数达到DISPLAY_CYCLE时，停止显示新的序列，游戏进入玩家输入阶段。该模块通过子模块random_generator生成随机序列。该模块输出we（写使能信号）、wn（写序号）、d（写入数据）到下一个模块number_mem中，存储这轮显示的各个LED序列，用于在玩家输入阶段时比较。

随机序列生成模块

```
// take least 10 bits of board clock as random number
module random_generator (clk, game_clk, resetn, randnum);
  input   clk, game_clk, resetn;
  output  [9:0] randnum;
  reg   [9:0] randnum;
  reg   [31:0] cnt;

  initial begin
    randnum <= 10'b0;
    cnt <= 0;
  end

  always @ (posedge clk or negedge resetn)
    begin
      if (!resetn) begin
        cnt <= 0;
```

```
      end
      else begin
        cnt <= cnt + 1;
      end
    end

  always @ (posedge game_clk or negedge resetn)
    begin
      if (!resetn) begin
        randnum <= 10'b0;
      end
      else begin
        randnum <= cnt[15:6];
      end
    end
endmodule
```

该模块维护一个32位计数器，计数的自增频率为板载时钟频率。在每一个游戏时钟上升沿，取计数器的第15位到第6位的10位数字作为生成的随机序列。

序列存储及输入对比模块

```
// exist: compare user input with saved leds value
// wb: if exist, delete the value in latch
// d, wn, we: write value when display leds flow
module number_mem (input_key, sw, d, wn, we, clk, clrn, exist, led);
   input  [3:0] wn;
   input  [9:0] d, sw;
   input        input_key, we, clk, clrn;
  output        exist;
  output [9:0] led;
  reg    [9:0] led;

   reg [9:0] register [0:9];

  assign exist = sw != 0 && ((register[0] == sw) || (register[1] == sw) ||
            (register[2] == sw) ||
            (register[3] == sw) || (register[4] == sw) || (register[5] == sw)
||
            (register[6] == sw) || (register[7] == sw) || (register[8] == sw)
||
            (register[9] == sw));

  initial begin: init
    integer i;
      for (i=0; i<10; i=i+1)
      register[i] <= 0;
```

```
  end

   always @(posedge clk or negedge clrn) begin
      if (clrn == 0) begin: reset // reset
      integer i;
         for (i=0; i<10; i=i+1)
            register[i] <= 0;
      end else begin
         if (we == 1) begin
            register[wn] <= d;
      end
      else if (!input_key) begin: inputkey
        integer i;
        for (i=0; i<10; i=i+1) begin
          if (register[i] == sw) begin
            register[i] <= 0;
          end
        end
      end
      else begin
        led <= sw;
      end
      end
   end
endmodule
```

该模块用于LED显示阶段时记忆所有出现的LED序列，玩家输入阶段时对比输入是否正确。其中10个10位的reg类型变量register用于存储一轮显示中的10个LED序列。在LED显示阶段，we（写使能信号）为高电平时，所有的序列都会被写到对应的register中。该模块异步判断若存在register与玩家输入的开关序列一样，则输出exist信号。在玩家输入阶段，每当玩家确认输入时，需要将对应register置0。这样可以使得每一个玩家输入的正确序列只被用来确认正确一次，防止玩家重复输入同一个正确序列刷分。

得分显示模块

```
module display_controller (input_key, exist, display_state, clk,
                  resetn, hex5, hex4, hex3, hex2, hex1, hex0, correct,
                  incorrect, percent);
  input       input_key, exist, clk, resetn;
  input   [2:0] display_state;
  output  [6:0] hex5, hex4, hex3, hex2, hex1, hex0;
  output  [6:0] correct, incorrect, percent;
  wire    [6:0] hex5, hex4, hex3, hex2, hex1, hex0;
  reg   [6:0] correct, incorrect, percent;
  wire        display_digit;
  wire    [6:0] hex5digit, hex4digit, hex3digit, hex2digit, hex1digit,
hex0digit;
```

```
  assign hex5digit = correct / 10;
  assign hex4digit = correct % 10;
  assign hex3digit = incorrect / 10;
  assign hex2digit = incorrect % 10;
  assign hex1digit = percent / 10;
  assign hex0digit = percent % 10;

  assign display_digit = display_state == 0;

  sevenseg hex5seg(hex5digit, hex5, display_digit);
  sevenseg hex4seg(hex4digit, hex4, display_digit);
  sevenseg hex3seg(hex3digit, hex3, display_digit);
  sevenseg hex2seg(hex2digit, hex2, display_digit);
  sevenseg hex1seg(hex1digit, hex1, display_digit);
  sevenseg hex0seg(hex0digit, hex0, display_digit);

  always @ (negedge input_key or negedge resetn)
    begin
      if (!resetn) begin
        correct <= 0;
        incorrect <= 0;
        percent <= 0;
      end
      else begin
        if (exist == 1) begin
          correct <= correct + 1;
          percent <= correct * 100 / (correct + incorrect);
        end
        else begin
          incorrect <= incorrect + 1;
          percent <= correct * 100 / (correct + incorrect);
        end
      end
    end
endmodule
```

该模块用于显示玩家的得分。在每一次玩家按下确认输入按钮（input_key）时，判断exist信号是否为1。若exist信号为1，玩家正确数（correct）加1，否则玩家错误数（incorrect）+1，同时计算玩家得分。该模块使用子模块sevenseg将得分转化为七段数码管对应的编码并输出。

七段数码管数字编码模块

```
module sevenseg (data, ledsegments, display_digit);
  input       display_digit;
  input   [6:0] data;
  output  [6:0] ledsegments;
  reg     [6:0] ledsegments;
```

```
  initial begin
    ledsegments = 7'b111_1111;
  end

  always @ (*)
    if (display_digit == 1) begin
      case(data)
                // gfe_dcba // 7段LED数码管的位段编号
                // 654_3210 // DE1-SOC板上的信号位编号
        0: ledsegments = 7'b100_0000; // DE1-SOC板上的数码管为共阳极接法。
        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; // 其它值时全灭。
      endcase
    end
endmodule
```

该模块判断输入的数字，并输出对应的七段数码管编码。

仿真验证

TestBench

```
`timescale 1ps/1ps            // 仿真时间单位/时间精度

module memory_game_sim;

    reg           resetn_sim;
    reg           clock_50M_sim;
   reg    [9:0] sw_sim;
   reg    [3:1] key_sim;

    wire   [6:0]  hex0_sim,hex1_sim,hex2_sim,hex3_sim,hex4_sim,hex5_sim;
   wire   [9:0] led_sim;
   wire       game_clk_sim, display_clk_sim, we_sim, exist_sim;
   wire   [9:0]  d_sim;
   wire   [3:0]  wn_sim;
```

```
    wire   [2:0]  display_state_sim;
    wire   [6:0]  correct_sim, incorrect_sim, percent_sim;

    initial
     begin
       key_sim <= 3'b111;
       sw_sim <= 10'b0101110110;
     end

    memory_game_main memory_game_instance(clock_50M_sim, resetn_sim, sw_sim,
key_sim,
                                hex5_sim, hex4_sim, hex3_sim, hex2_sim, hex1_sim,
                                hex0_sim, led_sim, game_clk_sim,
                                d_sim, wn_sim, we_sim, exist_sim,
display_state_sim,
                                correct_sim, incorrect_sim, percent_sim);
    initial
         begin
             clock_50M_sim = 1;
             while (1)
                 #1  clock_50M_sim = ~clock_50M_sim;
         end

    initial
         begin
             resetn_sim = 0;             // 低电平持续10个时间单位，后一直为1。
             while (1)
                 #5 resetn_sim = 1;
         end



    always @ (posedge game_clk_sim)
     begin
       if (led_sim != 0) begin
         sw_sim <= led_sim;
       end
       else begin
         sw_sim <= 10'b0;
       end
     end

    initial
     begin
       key_sim <= 3'b111;
       while (1) begin
         #3000 key_sim <= 3'b011;
         #3000 key_sim <= 3'b111;
         #3000 key_sim <= 3'b101;
```

```
        #3000 key_sim <= 3'b111;
      end
    end


    initial
        begin
          $display($time,"resetn=%b clock_50M=%b", resetn_sim, clock_50M_sim);
        end

endmodule
```

生成信号

- 周期为1个时间单位的clock_50M_sim信号作为CPU的clock信号
- resetn信号输入CPU，持续5个时间单位的低电平，然后改变为高电平并保持
- sw信号持续置为led信号，用于模拟在玩家正确输入时的开关输入
- key信号在011、111、101、111之间做周期为3000个时间单位的变换，作为玩家开始游戏和确认输入的模拟

验证结果

在ModelSim中进行仿真验证，sw信号和曾经出现的led信号相同时，输出exist并且correct信号持续自增；在key[3]按下时产生了10个随机序列，达到了预期的效果。对各阶段中信号输出的观察可知，游戏各阶段工作正常，数据能够按照设计正确的模块之间传播。对发光二极管和七段数码管的波形数据分析可得，显示数据根据玩家输入能够正确的输出。

实验总结

实验结果

标准测试程序代码在ModelSim中与其他实验代码经过编译，能正常完成仿真验证，能正确输出预期的波形。实验代码经过编译，载入到开发板后，能正常完成预期的游戏功能。通过玩家不同的操作能够产生正确的输出。经过游戏时间周期和显示字样的调试，游戏体验较好。

经验教训

- 在设计阶段，我划分了过多的子模块。在代码实现的过程中发现错误的划分子模块会导致各个模块之间产生比较严重的耦合，各个模块之间需要很多信号进行通信才能完成一个功能。经过重新设计，将功能相近的模块合并后，减少了模块之间的信号传递，增加了模块设计的合理性。
- 通过仿真验证可以提前发现并解决模块设计和程序实现中的问题。但是在实验过程中，我发现能在仿真验证中正确产生的波形，实际载入到开发板上后无法产生。通过放慢时钟周期，载入到开发板上观察具体的输出并调试，可以解决仿真验证无法解决的问题。
- 观察编译的警告，可以提前发现并解决实验设计和实现中所存在的问题。由于本次实验的信号是自主设计，在变量的wire和reg类型选择以及输入输出端口的声明上出现了不少问题。借助编译警告的帮助，经过仔细的检查解决了其中的问题。
- 在FPGA上设计游戏和之前设计CPU不同，必须考虑玩家的交互体验，因此需要在时间周期和显示字样上细致调试。

感受

作为一名软件工程专业大三的学生，在本次试验中我亲身体会到设计一个简单的游戏并运行在FPGA上的过程。对FPGA项目的设计流程，各个阶段的信号输入和输出，各个模块的功能和设计有了更加清晰的了解。同时在实验过程中查阅用户手册、verilog语法、使用仿真验证调试硬件需要很多耐心，在此过程中我定位问题、解决问题的能力得到了锻炼和提升。提前了解整体的设计思想以及充分理解各个部分的实现细节会使得实验的设计和实现的过程更加顺利。必要时和同学、老师的交流也带来了很多宝贵的经验和有效的思路。