复旦大学微电子学院

• 专用集成电路设计方法

2025
ASIC

FPGA 打地鼠游戏机实现

日期：2025-05-10

开发者：祝彦翔 22307130073

目录

一、设计规划

1.1 设计要求

本项目是基于复旦大学 2025 春季学期专用集成电路设计方法课程项目要求设计，完成简易打地鼠游戏机的 FPGA 实现。本设计在基础要求上自行发挥，加入多个附加模块，力求设计更高的趣味性和丰富度。本打地鼠游戏的核心要求是将 FPGA 板上的 8 个 LED 灯作为 8 个地鼠，8 个按键分别对应锤击不同位置的 8 个地鼠，在游戏过程中共有三轮游戏，难度递增，每轮游戏中有 8 小局，三轮中每局的时间分别是 6 秒、4 秒和 2 秒。如果打到正确的地鼠会立即有新的地鼠出现，分数也会增加，三轮打中地鼠加的分数分别是 1 分、2 分、3 分。在此基础上，我们考虑用一个七段数码管来实时记录玩家的分数，一个 textLCD 来记录（没有重置下）多局的最高分数，以及一个无源蜂鸣器来营造游戏的氛围。

1.2 设计思路

本设计采用的是 Top Down 的设计流程。首先先完成系统级设计，将这个复杂电路进行模块化的拆分。由设计要求中可以分解出不同的模块，一个 Top 级的调度模块和一个逻辑控制模块，这两个模块是系统的核心。顶层模块的输入输出包括了不同按键（rst_n, start, “敲击”按键等）以及数码管、LED、蜂鸣器 buzzle、TextLCD 的输出控制信号。而逻辑控制模块（GameControl module）则是控制游戏按设计要求实现，如亮灯时间等。这个模块需要的输入依旧是多个按键，输出则是实时的得分，游戏状态机状态等等。

另外，由设计要求中我们考虑不同的附加模块，包括数码管显示的 ScoreDisplay module，无源蜂鸣器的 SoundPlayer module 以及 1602 型 LCD 显示屏的 TextLCD module。它们相对于主逻辑而言是相对更独立的模块，可以进行独立测试开发。（这三种外设的具体使用呈现于 3.2 节）

拆分完不同的模块后可进行模块级的设计，将各个模块的逻辑功能在输入输出的要求下填充完整。本项目的主体逻辑在设计要求中基本都已规范。而外设部分，数码管记录当前游戏的实时分数（每次按 start 为一次游戏），而 TextLCD 则记录的是多次游戏的最高分数，初始情型下会显示”Max Score: 000”，随着游戏的进行，可以不断刷新最高纪录（在

FPGA 游戏重置后，每次游戏完成可以再按 start 键开启下一次游戏），如果按复位键则把最高分重置为 0，类似于游戏厅中投篮机原理。

再者是逻辑级和电路级的设计，这两个设计分别依赖于 Verilog 设计语言以及 EDA 的设计平台，这里不做展开。

1.3 设计方式

本项目的设计方式中包含对开发板、EDA 工具等的介绍：

1.3.1 设计平台

本项目为个人项目，软件平台为 Vivado，硬件实现平台为 FPGA，通过 FPGA 的设计来代替 ASIC 芯片的流片。相比于 ASIC 流片，FPGA 有着更高的便捷性和更低成本，更加适合于本项目中初步的芯片学习开发的场景。另外在这个项目中用到了少量的 Python 进行信息的前期处理等，代码没有复杂的库依赖，在开发者的环境下可以顺利运行。

1.3.2 开发板和 EDA 工具使用说明

本项目使用的开发板型号是正点原子达芬奇 pro 开发板，这是一款较高阶的学习性质的 FPGA 开发板，它的核心板是 Xilinx XC7A200T。由于这款开发板有众多外接自由引脚，所以使用这款 FPGA，它的具体参数可见 3.1 节。

本项目选取的 EDA 工具为 Vivado. Vivado 是由 Xilinx 公司开发的一款 FPGA 开发工具，主要用于基于 Xilinx FPGA 芯片的 数字电路设计、仿真、综合和实现。Vivado 相比于更高阶的 EDA 工具更加简洁，适合小规模项目的开发。这里开发者也使用了更为复杂但精细的上板流程，使用了 Synopsys 公司的 DC 和 IC compiler 等工具进行了相同的流程进行了逻辑综合和布局布线等，但是相较于 Vivado 更加复杂，对于这种小项目的开发使用 Vivado 平台是足够的。

开发平台是 Vivado 2023.2 版本，如使用不同版本验证可能会有版本冲突问题。

1.3.3 设计流程图

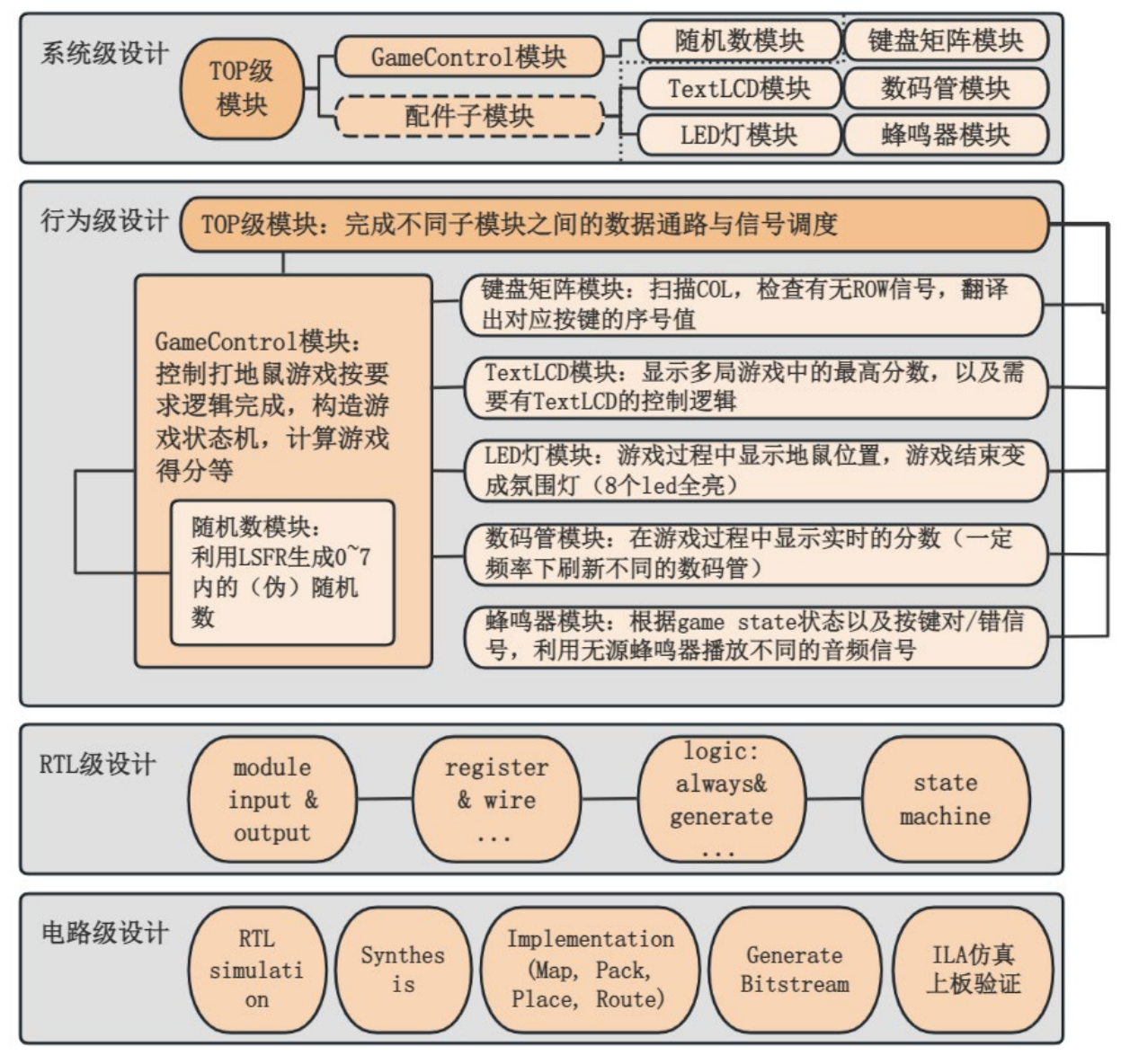

图 1：设计流程图

二、设计实现

2.1 框图介绍

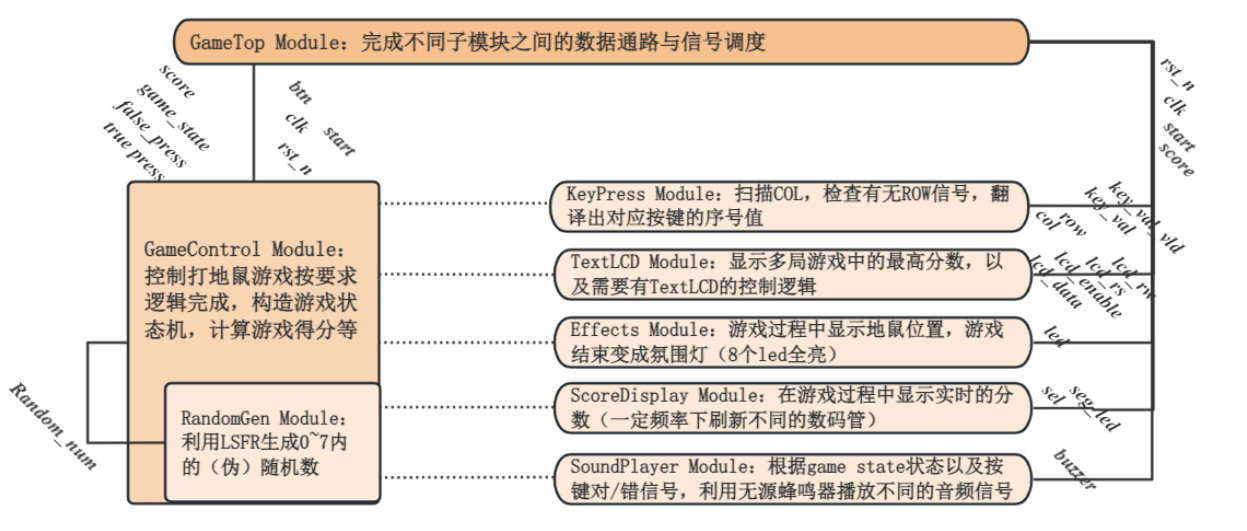

图 2：模块框图及不同模块之间的（主要）联系

2.2 各模块设计和验证（仿真激励和结果波形说明）

各个模块的设计思路已经在第 1 节中分析，下面会结合仿真来详细讲解不同模块的逻辑实现方式。

2.2.1 顶层调度模块：GameTop module

```
`timescale 1ns / 1ps

module GameTop(
    input clk,
    input rst_n,
    input start,              // Start button
    input [3:0] row,          // 4 rows for key matrix
    output [3:0] column,      // 4 columns for key matrix
    output [7:0] led,         // 8 LED outputs
    output [7:0] seg_led,         // 7-segment display segments
    output [5:0] sel,      // Select signal for 7-segment display
    output buzzer,            // Buzzer output
    output [7:0] lcd_data,    // LCD data output
    output lcd_enable,        // LCD enable signal
    output lcd_rs,            // LCD register select signal
    output lcd_rw             // LCD read/write signal
    );
```

图 3：GameTop 模块的输入输出

```
// Lighting and sound effects module
Effects effects(
    .clk(clk),
    .rst_n(rst_n),
    .game_state(game_state),
    .target_led(target_led),
    .led(led)
);

// Score display module
ScoreDisplay score_display(
    .sclk(clk),
    .s_rst_n(rst_n),
    .score(score),
    .sel(sel),
    .seg_led(seg_led)
);

// Sound player module
SoundPlayer sound_player(
    .clk(clk),
```

图 4：GameTop 主体为子模块的实例化

GameTop 作为项目的最顶层模块，起到的是串联起不同子模块的作用，里面例化了所有的二级模块，让这些子模块的输入输出在顶层模块上交互。

2.2.2 游戏逻辑控制模块：GameControl module

```
module GameControl(
    input clk,
    input rst_n,
    input start,
    input [7:0] btn,
    input [7:0] random_num,
    output reg [1:0] game_state,
    output reg [2:0] round,
    output reg [47:0] max_time,
    output reg [7:0] target_led,
    output reg [5:0] score,
    output reg false_press,
    output reg true_press
    );
```

图 5：GameControl 模块输入输出

```
IDLE: begin
    if (!start) begin
        game_state <= PLAY;
        round <= 3'b001;
        score <= 6'b000000; // Reset score
        max_time <= ROUND1_TIME;
        timer <= 32'b0;
        count <= 4'b0;
        target_led <= 1 << (random_num % 8);
        false_press <= 0;
        true_press <= 0;
    end
end
```

图 6：Gamestate=IDLE 状态

```
PAUSE: begin
    pause_cnt <= pause_cnt + 1;
    if (pause_cnt > PAUSE_TIME) begin
        restart <= 1;
    end
    if (restart && !start) begin
        game_state <= PLAY;
        pause_cnt <= 0;
    end
end
```

图 7：Gamestate=PAUSE 状态

```
PLAY: begin
    // press to pause the game
    if (!start && !restart) begin
        game_state <= PAUSE;
    end
    // back from PAUSE state
    if (restart) begin
        if (start) begin
            restart <= 0;
        end
    end

    timer <= timer + 1;
    if (true_press || false_press) begin
        true_press <= 0;
        false_press <= 0;
    end

    if (timer >= max_time) begin
        timer <= 32'b0;
        next_led <= 1 << (random_num[2:0]);
        if (next_led == target_led) begin
            next_led <= 1 << (~random_num[2:0]);
        end
        target_led <= next_led;
        count <= count + 1;
    end else if (btn == target_led) begin
```

图 8：Gamestate=PLAY 状态（部分）

```
GEND: begin
    if (!start) begin
        game_state <= IDLE;
    end
end
```

图 9：Gamestate=GEND 状态

GameControl 模块是控制游戏的核心部分，上图展示了 GameControl 的状态机（部分）。在本模块中设计了有四个状态的状态机，分别是 IDLE：游戏机开启而未按下 start 开始键的状态；PAUSE：游戏中途按下 start 键进入的暂停状态；PLAY：游戏状态；GEND：游戏结束状态。

首先是游戏时间的设置，本 FPGA 的默认时钟频率是 50MHz，这也是常用 FPGA 时钟频率这里采用默认值。由此可以计算出 2 秒、4 秒、6 秒分别对应的时钟周期数，作为游戏

状态下的计数目标。在 PLAY 状态下，如果时间到了最长时间，或是 btn 信号与 led 信号一致的话，则通过 RandomGen 生成的随机数生成下一个地鼠 led 位置，在这里添加了一个补充逻辑：如果下一个 led 位置与当前位置相同的话，则把这个随机数（共 3bit）取反，保证再生成的数肯定不是先前的位置。

在游戏的过程中，分数不断的被加到 score 寄存器中，进而被转移给数码管模块等进行显示。

关于游戏暂停逻辑，规定要求游戏过程中按下 start 键可以暂停游戏，这里考虑到实际情况下暂停的时间肯定不会小于 10ms，于是以 10ms 作为 start 按键的反应时间，只有当 10 ms 后再次按下 start 按键才可以重新恢复游戏，以防止按键连触导致游戏暂停失败。

这个状态机的流程是非常简洁的，在整体的线性循环流程下，插入 PLAY 和 PAUSE 之间的相互跳转，控制逻辑清晰避免产生游戏 bug。

2.2.3 随机生成模块：RandomGen module

```
module RandomGen(
    input clk,
    input rst_n,
    output reg [7:0] random_num
    );

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            random_num <= 8'b10101010;
        end else begin
            random_num <= {random_num[6:0], random_num[7] ^ random_num[5] ^ random_num[4] ^ random_num[3]};
        end
    end

endmodule
```

图 10：RandomGen 模块的工作逻辑

在 FPGA（或数字电路）中难以生成真随机数，需要引入器件才能做到真实随机。关于伪随机，最常见的就是这 LFSR 生成伪随机数。这里用的也正是这个方法。在本项目中我们需要的随机数范围是 0~7，构造一个 8bit 的 LFSR 序列，最后一位多比特异或构造一个近似的随机 0 或 1，前面的序列通过不断前推生成。在一个长时间周期下，如真实游戏下系统时钟周期 50MHz，1 秒为例就生成了 50MHz 个随机数，可以认为是接近真实随机。为了得到 0~7 下的随机数只要取这个序列的末三位即可。

进一步考虑 LFSR 伪随机数生成方法，其有一定的局限性，即生成不同的随机数的概率及分布不太随机。如果进一步增加随机模块的随机性，可以考虑 Sobol 序列（构造低差异性

序列）再使用 LFSR，可以有更好的随机性。但这里考虑项目的简易程度，还是使用简单 LFSR 完成。

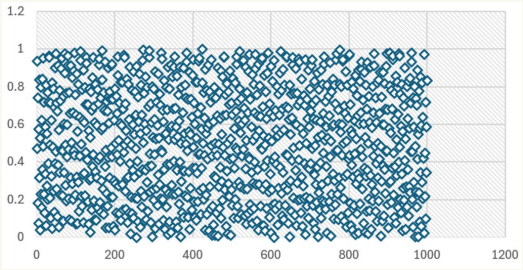

图 11：SOBOL 低差异序列生成
（横轴为生成顺序，纵轴为序列以 0~1 的浮点数进行表示的分布）

图 12：随机数生成模块波形图（每周期生成一个伪随机序列）

2.2.4 LED 灯工作模块：Effects module

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        led <= 8'b11111111;
    end else begin
        case (game_state)
            2'b01: begin
                led <= ~target_led;
            end
            2'b10: begin
                led <= 8'b00000000;
            end
            default: begin
                led <= 8'b11111111;
            end
        endcase
    end
end
```

图 13：Effects module 控制逻辑

Effects module 控制的是八个 led 灯的亮与灭。Gamestate 对其进行控制：在游戏进行之前或是游戏暂停时，板上的八个 led 灯均是全灭状态，在游戏进行过程中，我们用 led 灯来模拟地鼠，此时总会有一个地鼠出现也就是八个灯总有一个会亮，具体是由 GameControl 中控制的。另外当游戏结束时，这八个灯会全亮表示结束，可以开启下一局游戏。

Effects module 可视为另一个基于 gamestate 的状态机附加控制。

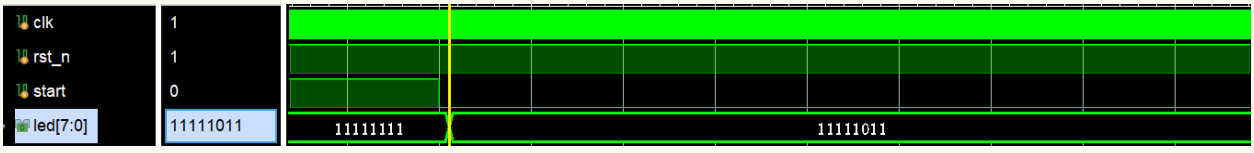

图 14：LED 灯模块（图中是游戏开始过程，灯从全灭变换到 3 号灯亮）

2.2.5 七段数码管记分牌模块：ScoreDisplay module

```
always @(*) begin
    if (score >= 40) begin
        tens = 4;
        ones = score - 40;
    end else if (score >= 30) begin
        tens = 3;
        ones = score - 30;
    end else if (score >= 20) begin
        tens = 2;
        ones = score - 20;
    end else if (score >= 10) begin
        tens = 1;
        ones = score - 10;
    end else begin
        tens = 0;
        ones = score;
    end
end
```

图 15：分数的个位与十位

```
always  @(posedge sclk or negedge s_rst_n) begin
    if(s_rst_n == 1'b0)
        sel <= 6'b11_1110;
    else if(cnt_1ms == (DELAY_1MS-1))
        sel <= {sel[4:0], sel[5]};
end
```

图 16：数码管显示位置 select 信号刷新逻辑

```
always  @(*) begin
    case(sel)
        6'b01_1111: seg_led_temp = ones;
        6'b10_1111: seg_led_temp = tens;
        6'b11_0111: seg_led_temp = 4'd0;
        6'b11_1011: seg_led_temp = 4'd0;
        6'b11_1101: seg_led_temp = 4'd0;
        6'b11_1110: seg_led_temp = 4'd0;
    endcase
end

always @ (posedge sclk or negedge s_rst_n) begin
    if (s_rst_n == 1'b0)
        seg_led <= SEG_ZERO;
    else case(seg_led_temp)
        4'd0:    seg_led <= SEG_ZERO;
        4'd1:    seg_led <= SEG_ONE  ;
        4'd2:    seg_led <= SEG_TWO  ;
        4'd3:    seg_led <= SEG_THREE;
        4'd4:    seg_led <= SEG_FOUR ;
        4'd5:    seg_led <= SEG_FIVE ;
        4'd6:    seg_led <= SEG_SIX  ;
        4'd7:    seg_led <= SEG_SEVEN;
        4'd8:    seg_led <= SEG_EIGHT;
        4'd9:    seg_led <= SEG_NINE ;
        default: seg_led <= SEG_ZERO;
    endcase
end
```

图 17：数码管显示信号控制逻辑

七段数码管的控制是由 select 信号和 seg_led 信号共同完成的，其工作显示也是快速刷新不同位置产生视觉停留。Select 信号的刷新周期是 1ms，短于视觉上分辨时间。这里用到的数码管共六位，由于最高分数只有 48 分，所以前 4 位被直接设成了 0，最后两位分别记为 tens 和 ones。Tens 和 ones 两位并没有用直接求除求余进行计算，而是如图采取了 case 进行计算，目的是避免 FPGA 求除法操作下的不稳定性。

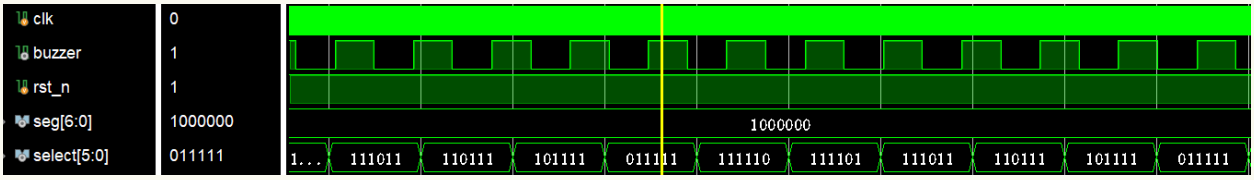

图 18：数码管模块波形图（数码管的 select 在不断刷新 ，图中正显示数字 0）

2.2.6 无源蜂鸣器氛围声模块：SoundPlayer module

```
function [31:0] get_freq;
    input [2:0] seq_val;
    begin
        case (seq_val)
            3'd0: get_freq = FREQ_C5;
            3'd1: get_freq = FREQ_D5;
            3'd2: get_freq = FREQ_E5;
            3'd3: get_freq = FREQ_F5;
            3'd4: get_freq = FREQ_G5;
            3'd5: get_freq = FREQ_A5;
            3'd6: get_freq = FREQ_B5;
            3'd7: get_freq = FREQ_C6;
            default: get_freq = FREQ_C5;
        endcase
    end
endfunction
```

图 19：不同频率的生成器

```
IDLE: begin
    sequence[0] <= IDLE_SEQ0;
    sequence[1] <= IDLE_SEQ1;
    sequence[2] <= IDLE_SEQ2;
    seq_length <= 3;
end
PLAY: begin
    if (false_press) begin
        sequence[0] <= FALSE_SEQ0;
        sequence[1] <= FALSE_SEQ1;
        sequence[2] <= 0;
        seq_length <= 2;
    end else if (true_press) begin
        sequence[0] <= TRUE_SEQ0;
        sequence[1] <= TRUE_SEQ1;
        sequence[2] <= TRUE_SEQ2;
        seq_length <= 3;
```

图 20：不同状态控制不同声音产生

SoundPlayer 模块主要用于根据游戏运行状态控制无源蜂鸣器发声。通过输入游戏状态（IDLE、PLAY、SOUNDEND）以及用户操作信号（true_press、false_press），模块可驱动蜂鸣器输出不同的提示音，增强玩家的体验。

模块内部设计了一个音符序列控制机制，根据不同的状态或事件，选择预设的音符序列。每个音符均以方波形式输出，频率由映射表 get_freq 动态生成。播放逻辑基于计数器控制音符持续时间和频率，实现多个音符连续发声，从而构成完整的提示音效果。

模块引入了状态变化检测机制，当检测到游戏状态或按键信号发生变化时，即触发新的音符序列播放。音符序列通过 sequence 数组保存，播放时按照顺序依次输出，对应频率通过查表获得，结合 tone_half_period 控制输出频率；音符切换由 tone_duration_counter 控制，每个音符持续 0.5 秒。

模块以 50MHz 系统时钟为基础，通过分频实现音符频率控制。每个频率周期分为两个半周期，分别对应蜂鸣器电平的高低切换。通过 tone_counter 控制方波翻转时机，输出连续稳定的频率信号，确保蜂鸣器能够准确发声。

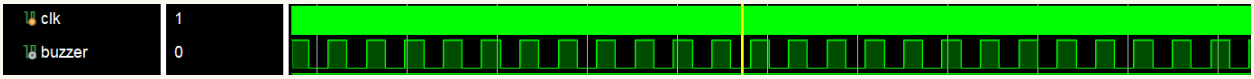

图 21：无源蜂鸣器的波形（在短时间范围下看 buzzer 频率是不变的）

2.2.7 1602LCD 最高纪录录模块：TextLCD module

```
module TextLCD (
    input wire clk,             // 50MHz clock
    input wire rst_n,           // Active low reset
    input wire [7:0] max_score, // 0-255 max score
    output reg [7:0] lcd_data,
    output reg lcd_enable,
    output reg lcd_rs,
    output wire lcd_rw          // always 0: write mode
);
```

图 22：TextLCD module 输入输出

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        char_buffer[0]  <= "M";
        char_buffer[1]  <= "a";
        char_buffer[2]  <= "x";
        char_buffer[3]  <= " ";
        char_buffer[4]  <= "S";
        char_buffer[5]  <= "c";
        char_buffer[6]  <= "o";
        char_buffer[7]  <= "r";
        char_buffer[8]  <= "e";
        char_buffer[9]  <= ":";
        char_buffer[10] <= " ";
        char_buffer[11] <= "0";
        char_buffer[12] <= "0";
        char_buffer[13] <= "0";
        char_buffer[14] <= " ";
        char_buffer[15] <= " ";
```

图 23：字符存取，最高分数实时替换

```
WAIT_POWER_ON: begin
    if (power_on_cnt < 20'd750_000)
        power_on_cnt <= power_on_cnt + 1;
    else
        state <= INIT_0;
end
INIT_0: begin
    lcd_rs <= 0;
    lcd_data <= 8'h38; // Function set: 2 lines, 5x8 font
    state <= DELAY;
    delay_flag <= INIT_1;
end
INIT_1: begin
    lcd_rs <= 0;
    lcd_data <= 8'h0C; // Display ON
    state <= DELAY;
    delay_flag <= INIT_2;
end
INIT_2: begin
    lcd_rs <= 0;
    lcd_data <= 8'h06; // Entry mode set
    state <= DELAY;
    delay_flag <= INIT_3;
end
INIT_3: begin
    lcd_rs <= 0;
    lcd_data <= 8'h01; // Clear display
    state <= DELAY;
    delay_flag <= INIT_4;
end
INIT_4: begin
    lcd_rs <= 0;
    lcd_data <= 8'h80; // Set DDRAM address to 0x00
    state <= DELAY;
    delay_flag <= IDLE;
end
IDLE: begin
    if (slow_clk) begin
        char_index <= 0;
        state <= WRITE;
    end
    lcd_enable <= 0;
end
WRITE: begin
    lcd_rs <= 1; // Data mode
    lcd_data <= char_buffer[char_index];
    lcd_enable <= 1;
    state <= DELAY;
    delay_flag <= (char_index < 31) ? WRITE : IDLE;
    char_index <= char_index + 1;
end
DELAY: begin
    case (enable_phase)
        2'd0: begin
            lcd_enable <= 1;
            enable_phase <= 2'd1;
        end
        2'd1: begin
            lcd_enable <= 0;
            enable_phase <= 2'd2;
        end
        2'd2: begin
            if (delay_cnt < 16'd5000) begin
                delay_cnt <= delay_cnt + 1;
            end else begin
                delay_cnt <= 0;
                enable_phase <= 0;
                state <= delay_flag;
            end
        end
    endcase
end
```

图 24：TextLCD 控制状态机

上图代码中主要展示了控制 TextLCD 的状态机，其余部分可以参考完整代码。TextLCD 有其独特的控制逻辑和引脚规范，其原理写于 3.2.1 节。在这个项目中，其显示目标是“Max Score: *xxx*”这个数值初始下是 000，如果当前局刷新了这个纪录它会把最高分显示在上面。

这个状态机有 9 个状态，前几个都是初始状态，实现的是亮屏、清屏、写状态等不同的设置，然后进入写状态，包括了写操作和延迟操作，每次写一个实际字符都要进行延迟后再进入下一个字符的写，如果几个字符写完则重新进行这个状态机的循环主完成下一次的写循环。

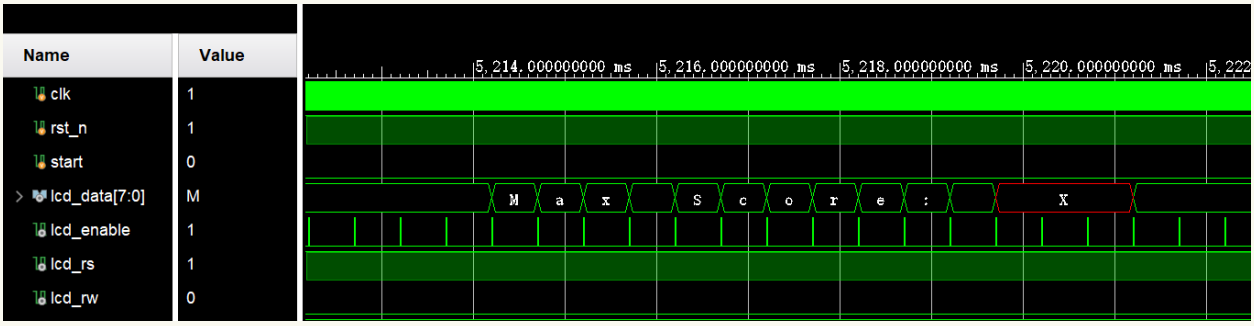

图 25：TextLCD 模块的波形图（一次写刷新过程）

2.2.8 键盘矩阵控制模块：KeyPress module

```
module KeyPress(
    input sclk,
    input s_rst_n,
    output reg [3:0] col,
    input [3:0] row,
    output reg [3:0] key_val,
    output reg key_val_vld
);
```

图 26：KeyPress 模块的输入输出

```
always @(posedge sclk or negedge s_rst_n) begin
    if(s_rst_n == 1'b0)
        col <= 4'b1110;
    else if(cnt_1ms >= DELAY_1MS)
        col <= {col[2:0], col[3]};
end
```

图 27：COL 扫描信号的生成

```
always @(posedge sclk or negedge s_rst_n) begin
    if(s_rst_n == 1'b0)
        key_val <= 'd0;
    else if(cnt_10ms == DELAY_10MS && key_and_d2 == 1'b0)
        case({row, col})
            8'hee:  key_val <= 'd0;
            8'hed:  key_val <= 'd1;
            8'heb:  key_val <= 'd2;
            8'he7:  key_val <= 'd3;
            8'hde:  key_val <= 'd4;
            8'hdd:  key_val <= 'd5;
            8'hdb:  key_val <= 'd6;
            8'hd7:  key_val <= 'd7;
            8'hbe:  key_val <= 'd8;
            8'hbd:  key_val <= 'd9;
            8'hbb:  key_val <= 'd10;
            8'hb7:  key_val <= 'd11;
            8'h7e:  key_val <= 'd12;
            8'h7d:  key_val <= 'd13;
            8'h7b:  key_val <= 'd14;
            8'h77:  key_val <= 'd15;
            default:key_val <= 'd0;
        endcase
end
```

图 28：按键序列号信号转换

KeyPress 控制的是键盘矩阵的按键信号，其输入值为 ROW 而 COL 为输出值，键盘矩阵的工作原理见 3.2.2 节。在这个模块中构造了两个子频率，一个周期是 1ms，一个周期是 10ms。1ms 的时钟用于 COL 扫描信号上，用于扫描检测哪一列的键位上发生了按下操作。而 10ms 的时钟则用于键盘矩阵的防抖操作，在键盘被按下的瞬间可以会发生快速的信号跳变，于是这里将检测信号的时间分辨率调为 10ms，避免信号切换时的短暂跳变，使按键信号稳定。

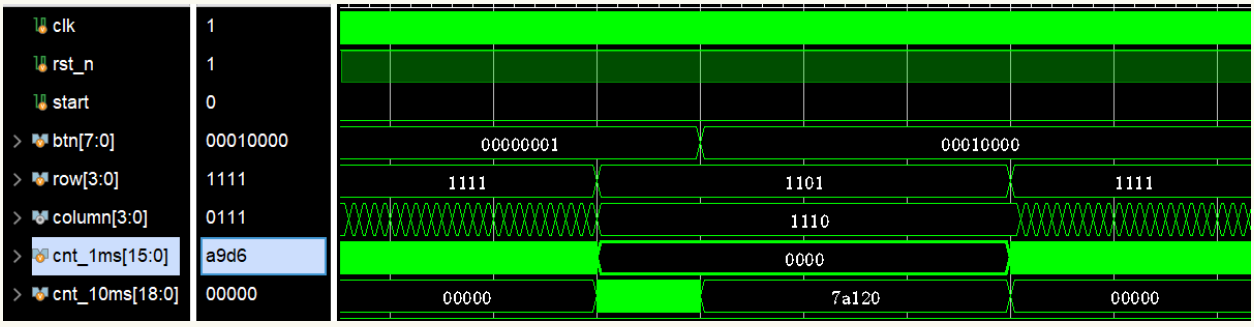

图 29：键盘矩阵逻辑波形图（按下第 5 号按键事件附近）

2.3 综合与实现

2.3.1 综合条件设定和结果分析

在综合条件约束中，由于设计较为简易，并没有考虑面积约束（只是对于 XC7A200T 核心板面积为最大值）、功耗约束等，只考虑时序约束。在约束文件中，我们规定 FPGA 的工作频率为 50Mhz 指定时钟频率，I/O 延迟也不予考虑。在 RTL 级包括 GameControl 等多个位置都用到了时钟频率如亮灯间隔为 6 秒或其它时间，都基于这个时钟频率规定。对于 Vivado 综合结果进行分析来看：

Resource	Utilization	Available	Utilization %
LUT	3709	134600	2.76
FF	4203	269200	1.56
BRAM	225	365	61.64
IO	45	285	15.79

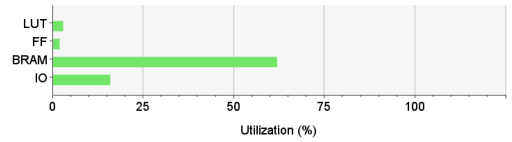

图 30：Report Utilizaiton 简要报告

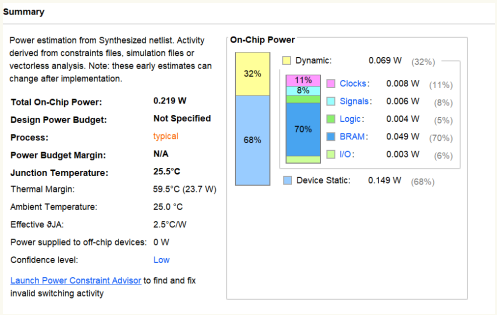

图 31：Report Power 简要报告

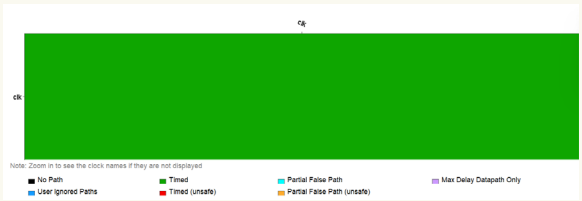

图 32：Report Clock Network 简要报告

根据综合报告中的资源利用率、功耗估算及时钟网络信息，整体设计情况良好。资源利用方面，查找表使用了 3709 个，占总资源的 2.76%；触发器使用了 4203 个，占 1.56%；IO 利用率为 15.79%。而块 RAM 使用了 225 个，占比达 61.64%，本设计对片上存储资源有着较高的依赖。功耗估算方面，芯片总功耗为 0.219 W，其中静态功耗为 0.149 W，占比 68%，动态功耗为 0.069 W，占比 32%。动态功耗中，BRAM 消耗最高，为 0.049 W，占动态功耗的 70%，这与 BRAM 的高利用率相吻合。其他模块如时钟、信号、逻辑和 IO 功耗较低，分别为 11%、8%、5% 和 6%。

所以从综合结果来看，本设计整体资源利用率较低，功耗等控制也符合要求。

2.3.2 静态时序结果分析

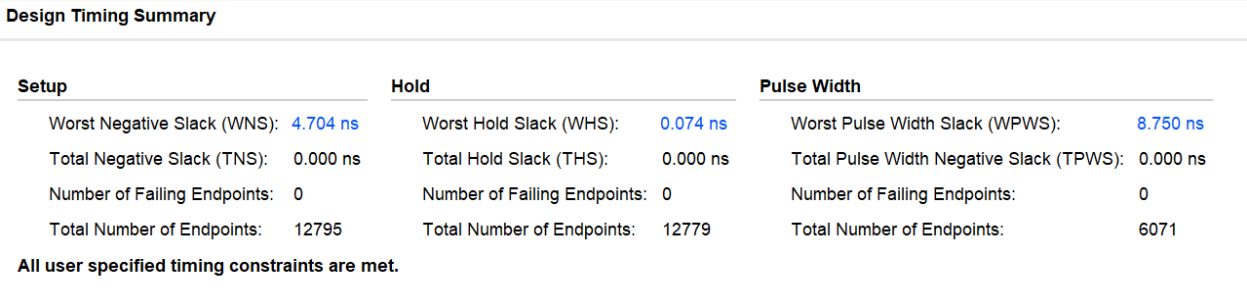

图 33：Timing Summary(Vivado)

静态时序分析（STA）是数字电路 设计中的关键验证方法，用于在不运行电路的情况下评估电路是否能在预定时钟频率下可靠运行。在 Vivado 中可以检查静态时序的 setup time, hold time, clock skew 和 slack.

本次静态时序分析结果表明，设计在 Setup、Hold 和 Pulse Width 三个方面均满足时序要求。其中，Setup 时序的最坏负裕度（WNS）为 4.704ns，Total Negative Slack（TNS）为 0，表明所有路径均未违反 Setup 约束；Hold 时序的最坏裕度（WHS）为 0.074ns，虽较小但为正值，同样没有路径违例；脉冲宽度方面，最坏裕度（WPWS）为 8.750ns，裕度充足。所有类型的时序检查中，失败路径数量均为 0，最终工具提示“所有用户指定的时序约束均已满足”，说明当前设计的时序性能良好，不再需要进一步优化。

三、FPGA 系统介绍、下载实现和调试，测试游戏过程

3.1 FPGA Artix-7 XC7A200T

这款 FPGA 是正电原子公司基于 Xilinx Artix-7 型号的开发板。目前 Xilinx7 系列 FPGA 芯片有四个子系列，他们分别是 Spartan7 系列、Artix7 系列、Kintex7 系列以及 Virtex7 系列。根据 Xilinx 提供的官方手册，这四款不同系列的芯片有不同的参数和应用场景。

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM(1)	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance(2)	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU(3)	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	–	16	32	96
Transceiver Speed	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	–	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	–	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

图 34：Xilinx7 系列资源一览表

相比于其它的系列，本项目中用到的 Artix-7 核心板有着低成本，应用场景广且接口多（包括了 DDR 硬核和高速的收发器）的特点，适合用作这种简易项目的 FPGA 学习。而对于正点原子达芬奇 PRO 底板而言，有着丰富的资源。该 FPGA 配有有源蜂鸣器，不过为了让音频更有趣味性，有源蜂鸣器并不能带来声音音调上的变化，所以考虑外接无源蜂鸣器，再加上 TextLCD 的多输入端口和外设数码管和矩阵键盘等，初步计算需要 30+ 的引脚拓展口，而该开发板符合此需求。

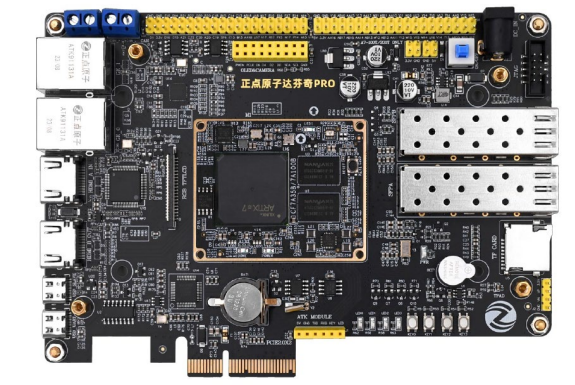

图 35：正点原子达芬奇 PRO 底板

图 36：Xilinx ARTIX-7 核心板

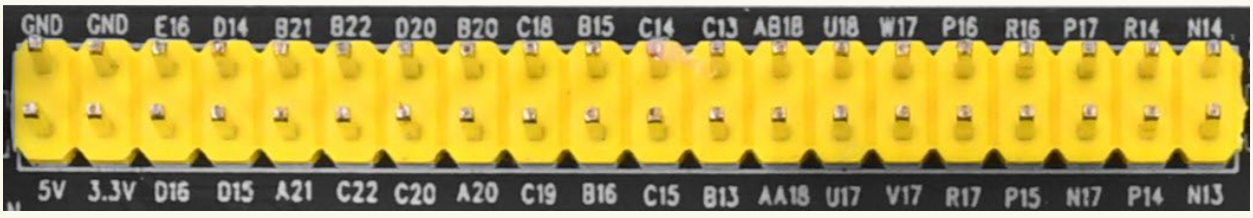

图 37：底板的拓展口引脚图

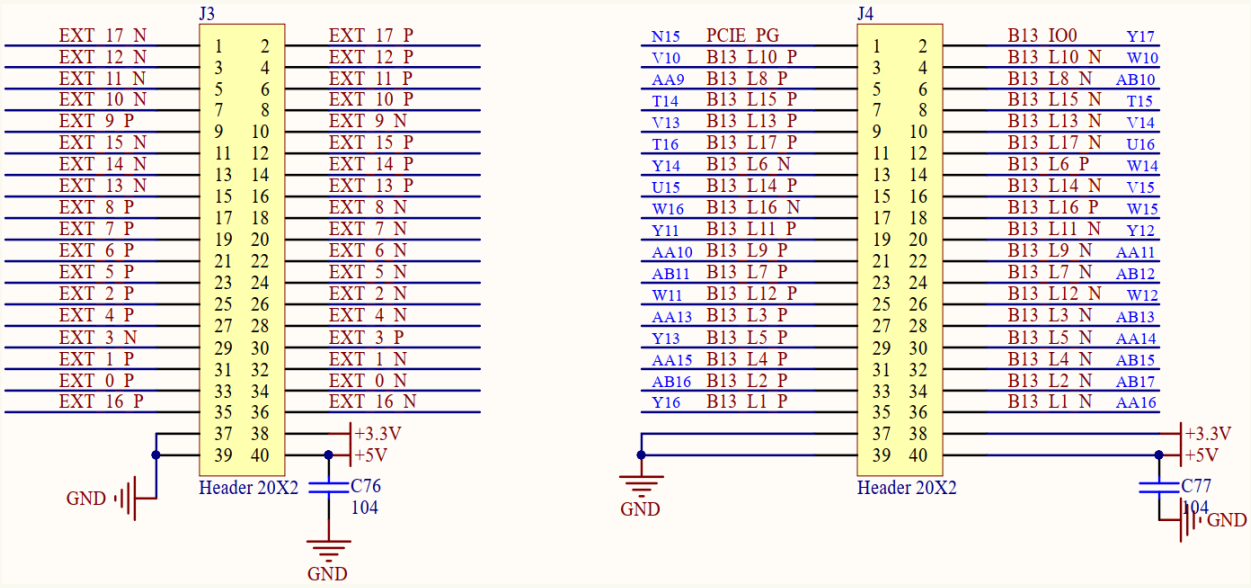

图 38：引脚接口

另外，开发板上配有复位按键，和少量的按键，我们可以把 rst_n 和 start 按键设为板上按键。

3.2 外连设备介绍

本项目中使用到的外连设备包括以下四种，但实际项目中七段数码管、矩阵键盘和 8 个 LED 灯集成在了一个外接设备中。

3.2.1 1602 TextLCD

图 39：1602 型号 TextLCD

1602 型号的 LCD 是最为普通的 LCD，可以显示 16*2（32）个字符。该型号的 LCD 共有 16 个引脚，分别为 VSS（接地），VDD（接高电平），V0（不同的 V0 带来不同的分辨度，这里取 V0 为 0 即 GND，即获得最为明显的分辨度），LED_A 和 LED_K 分别也接高电平和 GND，此外的 11 个引脚是需要接引脚的，RS 是数据命令，RW 是读写操作，E 是使能信号，其余的 D0~D7 共 8 位输入的是显示字符的 ASCII 码。

在这个项目中，TextLCD 显示的是“Max Score: *xxx*”其中 *xxx* 代表的是多次游戏的最高分数，这个分数是实时更新的。这里只用到了 16 个字符，为了防止其余位置上显示混乱，其余位置上都输出空格。

关于 TextLCD 的控制，RS 和 RWj 有着不同的意义：

RS 高电平的时候对应了数据寄存器，也即显示数据，而 RS 低电平的时候对应了指令寄存器（在此情形下不同的 Data 对应了不同的 LCD 指令，如下图所示）。而 RW 高电平代表了读操作，低电平代表了写操作。

序号	指令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	0	0	1	x
3	输入方式设置	0	0	0	0	0	0	0	1	I/D	S
4	显示开关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位控制	0	0	0	0	0	1	S/C	R/L	x	x
6	功能设置	0	0	0	0	1	DL	N	F	x	x
7	字符发生存储器地址设置	0	0	0	1	字符发生存储器地址					
8	数据存储器地址设置	0	0	1	显示数据存储器地址						
9	读忙标志或地址	0	1	BF	计数器地址						
10	写入数据至CGRAM或DDRAM	1	0	要写入的数据内容							
11	从CGRAM或DDRAM中读取数据	1	1	读取的数据内容							

图 40：LCD1602 控制指令

3.2.2 矩阵键盘

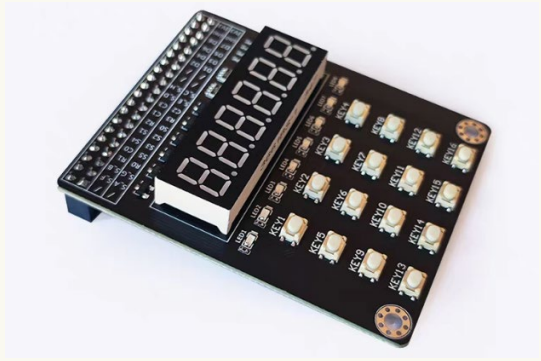

图 41：项目中使用的七段数码管、LED 和矩阵键盘一体外设

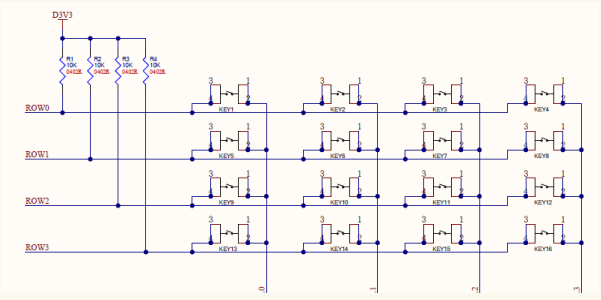

图 42：矩阵键盘的控制方式

由右图矩阵键盘的控制原理图可以知道，矩阵键盘并非像 FPGA 上的已有按键一样直接通过端口号进行独立的控制，而是需要考虑行号和列号。如图的 4*4 矩阵键盘一共连接 8 个引脚，分别 是 ROW0~ROW3，及 COL0~COL3。

其中需要注意的是 ROW 信号是输入信号而 COL 信号是输出信号。为了识别具体按下的键位置，COL 信号是一个扫描输出信号给到按键矩阵中，如果扫描信号扫到的 COL 列下发现了有按下的操作，再把 ROW 信号读回来，最终 COL 和 ROW 两个信号共同组成了当前按键的位置。

例如按下二号按键，COL 作扫描信号，当其扫描到 1101（COL[1]==0）时，检测到了按键，此时输入信号 ROW 为 1110（ROW[0]==0）。经过 case 语句将 ROW，COL 的位置信号“翻译”成按键的序号 0~15，并在后续代码中再将其转为 btn 信号与 led 信号进行对比。

在实际操作中，当按下特定按键的时候，按键并不一定是简单的单个上升或下降沿，而可能在变化的瞬间产生一些毛刺信号，这需要我们对按键矩阵进行消抖，这里采取了 10ms 的防抖时间。具体消抖的逻辑可见设计实现的部分。

3.2.3 七段数码管

在本实例中七段数码管与矩阵键盘是一体的，但是控制它们的引脚相互独立，

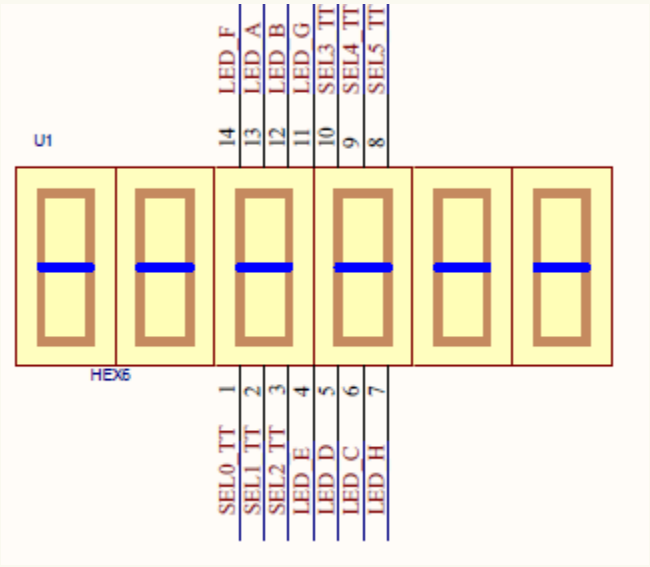

图 43：七段数码管的引脚控制

七段数码管由 14 个引脚控制，其中有 8 个是控制单个数码管的显示（低电平有效，从最上面的数码管到小数点分别是最高位 seg[7] 到最低位 seg[0]），如 seg：1001111 代表着数字 1。

而剩余的 6 位为 SEL 信号，即选择第几位数码管进行显示，图中共有 6 个七段数码管，但实际上由于单次游戏最高只会到 1*8+2*8+3*8=48 分，故只需用到末 2 位，前面的数码管保持全空。SEL 信号在 111110 和 111101 之间以很高的频率刷新，在人眼视觉上来看是同时显示两位数码管。

3.2.4 无源蜂鸣器

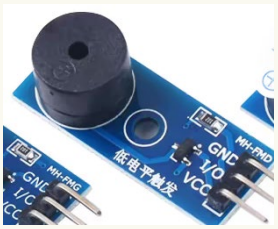

图 44：5V 低电平触发无源蜂鸣器

蜂鸣器分为有源蜂鸣器和无源蜂鸣器两种类型。有源蜂鸣器只会产生哔哔的声音由于有震荡源的干扰几乎难以发出音乐，创意性不大，所以这里采用了无涯蜂鸣器（5V 低电平触发）。

虽然只有一个 buzzle 的信号端口，但是无源蜂鸣器的原理较为复杂。无源蜂鸣器的发声主要是通过 PWM 波来进行控制的，进一步说是通过 PWM 波决定其频率和占空比，占空比即高电平时间的比例，其决定的是音量（如静音就是要让占空比为 0），而频率则决定音调。

本设计原先的目标是发出提示词，预先将四段音频 mp3 转换成 wav 文件，再将 wav 型文件转换成 coe 文件（FPGA 中 BRAM 可导入文件），这一部分由 Python 实现（代码也附于项目）。由游戏状态机的不同状态控制播放不同的音频。开始游戏前播放“欢迎来到……按 start 按键开始游戏”，如果游戏过程中按错播放“按错啦”，如果打中地鼠则播放“被你打中啦”，游戏结束后也会播放“游戏结束……查看分数”不同的播放词。

```
memory_initialization_radix=16;
memory_initialization_vector=
8000,
8003,
7FFF,
8002,
8001,
8003,
7FFE,
```

图 45：MusicIdle 的 coe 文件开头部分

但是经过实际操作后发现由 mp3 导出的音频波形难以恢复其频率特性，无源蜂鸣器生成的声音会产生很多杂音难以分辨发出的声音。（其代码及文件也保留在了总文件中）

于是在本项目设计过程中将目标改为了不同音调的比较有意思的音乐，如 IDLE 状态下播放从中音 E 到 G 的上行三音阶，如果游戏中按正确的时候会播放 F5、G5、A5 逐渐上升的音调，显示积极情绪。

3.3 下载实现

编译工程并生成比特流文件后，将下载器一端连接电脑（本达芬奇 PRO 开发板的上板需要使用下载器）。通过 Vivado 的 Open Hardware Manager 下检测到目标 FPGA。然后可以 Program Device 也即将比特流文件烧录至 FPGA 中。

3.4 调试

FPGA 的调试有多种，最为基础的调试是通过 testbench 文件进行的 RTL 级的调试。通过 RTL 调试可以通过波形判断是否与理想波形一致。

本设计中使用的 testbench 文件模拟了按键操作，这里考虑理想状态并没有模拟按钮的毛刺信号。另外由于生成的是随机数（在 testbench 固定了按键时间的情况下其实不是随机的，但是考虑到真实情况下的游戏每次按按键的时间是不会完成一样的而远大于随机数生成的频率）在 testbench 中仅仅模拟了随便按的情型，也即不断轮流按 1~8 八个按钮。

具体的调试波形可以见第 2 节的波形图，可见 RTL 级的逻辑与开发逻辑是一致的。

另外，除了 RTL 级的调试外，还有利用 ILA IP 核进行上板验证。Program Device 之后可以看到 ILA 的波形图。通过在线调试观察可知，上板验证结果与仿真结果是一致的，这也说明了游戏机的功能正确性。

比如在本项目过程中，对于这按键矩阵外设的使用比较困惑，起始以为 COL 和 ROW 信号均为输入信号，通过 ROW 和 COL 的交点值确认按键的位置。RTL 仿真中结果也与构想中一致，但是上板验证却不能实现对应的要求，于是使用 ILA 进行了在线调试。

通过 ILA 调试发现 ROW 和 COL 的端口有异常，始终没有逻辑变动，于是重新检查发现了键盘矩阵的正确使用方式，进而完成了这个 module。

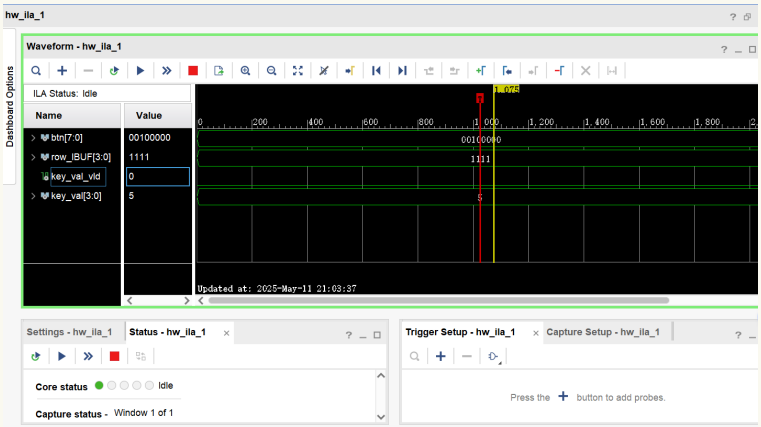

图 46：用 ILA 调试 ROW 和 COL 信号

3.5 测试游戏过程

这里上板后 FPGA 游戏机的使用。

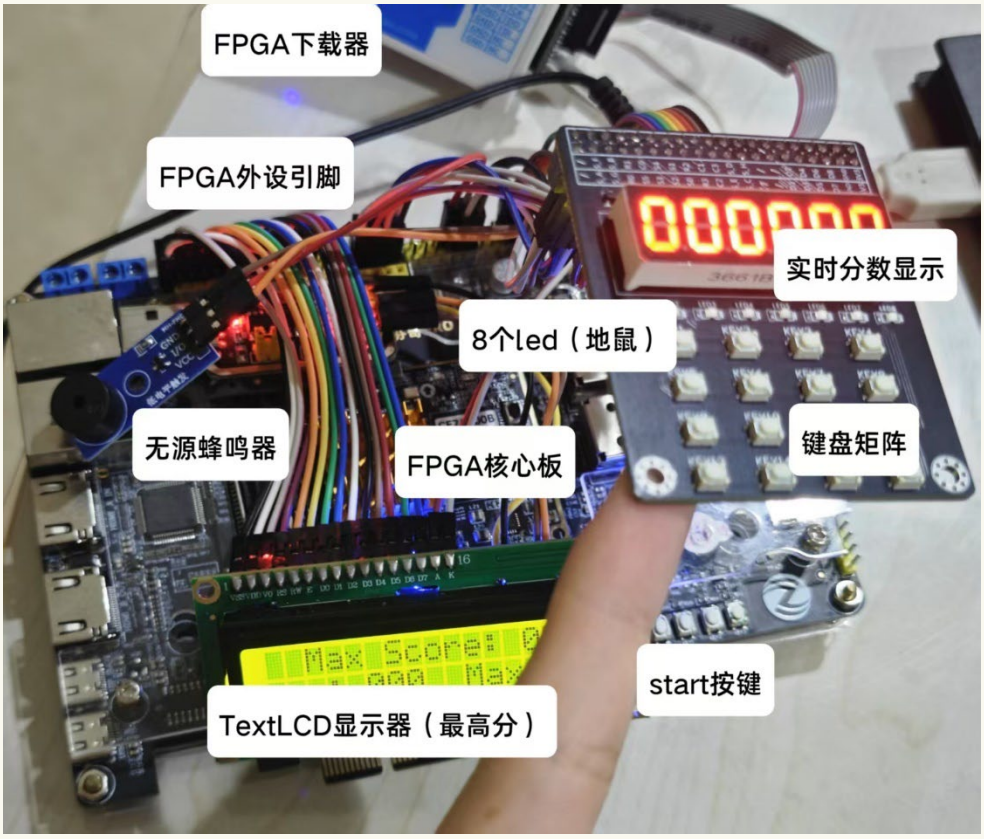

图 47：FPGA 及所有配件连接情况

本项目在 FPGA 板上有上板验证，详情可见视频附件，这里展示中间关键步骤。上图展示了不同模块配件的连接：FPGA 板需要通过下载器与电脑连接，将比特流从电脑上下载下来，另外还要单独再连电源。我们利用板上的复位键和一个按键作为 start 按键。

开始游戏时先按下复位键，可以看到 Max Score 为 000，分数数码管也是 000000，灯全灭。按下 start 按键，如下图 8 个 led 灯有一个亮，按对按键有加分灯会转换到不同的地方，但如果按错并不会有反应。同时无源蜂鸣器也会工作，根据状态发出不同的声音，符合要求。

一局游戏结束后，本局得分 25 分，由于这是第一局所以 25 分就是游戏的最高分。此时 Max Score: 25。LED 灯也是变成氛围灯的模式，8 个灯全亮，无源蜂鸣器也会播放一小段游戏结束的音效。

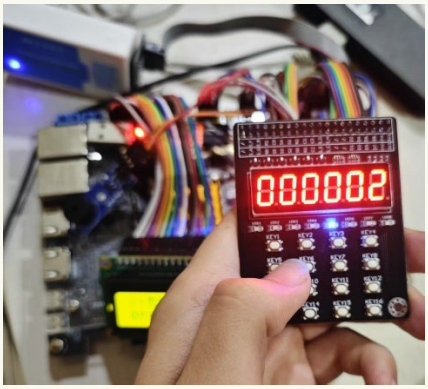

图 48：游戏进行过程中

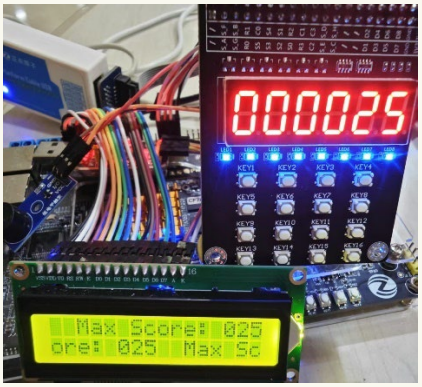

图 49：第一局游戏结束（得分 25 分）

下面再进行第二、三局，第二局结束后得到了 33 分，此时得分比上一局 25 分高，于是此时 Max Score 变成了 33 分。同理第三局又刷新了高分 43 分，Max Score 又变成了 43 分。如果这时按 rst_n 键重新复位，Max Score 恢复为 0。

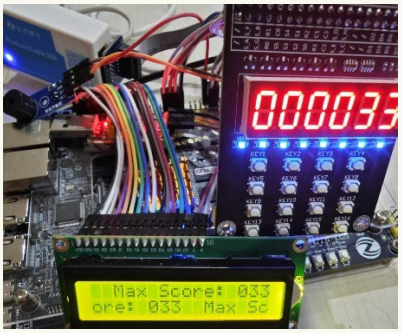

图 50：第二局游戏结束（得分 33 分）

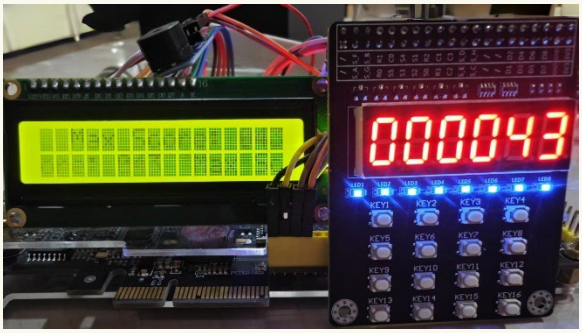

图 51：第三局游戏结束（得分 43 分）

另外，游戏中暂停也可正常使用，按下 start 键后，8 个 led 灯全灭，此时其它计时等操作都不会进行。再次按下 start 键后，led 灯恢复，计时也恢复任何工作与之前无打断情况是一致的。

四、设计总结

本设计利用了 Top Down 的设计流程，从产品目标到模块划分和设计，再到这具体逻辑的搭建、RTL 级的实现进行了一个完整的产品开发流程。在此基础上不断修改纠错并在基础要求上继续添加附加模块，在不过度修饰的情况下让用户的体验有更好的提升。

整个设计的流程是非常复杂的，既要包括 EDA 软件平台的使用也要综合硬件的设计。本项目用的 Vivado 设计平台和 Xilinx Artix-7 系列 FPGA 核心板有着非常好的设计匹配度，大大方便了从 RTL 级到比特流的过程（简化了使用不同设计软件的过程），从行为级到 RTL 级再到物理级、比特流级，一步步完成 FPGA 的设计。

另外在设计的过程中，同时为进行各级的调试，包括 RTL 级的调试和上板的动态调试等，在调试的过程中经常可以发现各种不同的问题，包括高低电平触发条件不对，游戏逻辑控制不对等等问题。这些问题都在调试的过程中被纠正。

五、个人体会

就我而言，本项目是我完成的最为完整和复杂的一个 FPGA 开发项目，一个人从零到有设计出了一个可以玩的游戏机出来，收获感是很大的。

另外，通过这个项目的开发我对于很多不同的芯片外接逻辑也有了清晰的认知，尤其是 TextLCD 和无源蜂鸣器的设计。这两个重要的外接设备以前都是只知道使用的效果而不知道具体如何控制，通过项目以做代学深入了解到了它们的控制方式，对于这种小插件的使用有了更多认知。

再者，我本身也喜欢这类趣味性的小设计。在一年前我就自己购买了正点原子的这款开发板但是并没有用过很多次，感觉单纯的研究有些较为枯燥。但这种项目让我有了更多的兴趣去钻研 FPGA 更加复杂的开发去设计更多更有意思的东西。

附：主要文件说明

在邮件中仅附不同模块的 verilog 代码以供参考（即下表中 game_machine\game_machine.srcs\source_1\New 文件夹），由于 BRAM IP 核和演示视频等文件内容较大，均放于 github 及百度网盘中以供查看。

文件结构如下（粗体文件代表项目中的工作模块，其余是 Vivado 自动生成的模块或文件夹名等）：

Whack-a-Mole-GameMachine
- *game_machine*
- *game_machine.srcs*
- *constrs_1\new**constraints.xdc**：*FPGA 约束文件
- *music*

- ***idle.mp3 / true.mp3 / false.mp3 / end.mp3：*** 不同状态下播放的音频信息
- ***idle.wav / true.wav / false.wav / end.wav：*** 音频信息对应的波形信号
- ***idle.coe / true.coe / false.coe / end.coe：*** 音频信息的 FPGA 可存储文件
- ***wav_to_coe.py：*** wav 向 coe 文件的转换代码

- *sim_1\new**GameTop_tb.v：*** 项目的 testbench 测试文件
- *source_1*
  - *ip：* 利用 Vivado 中的 Generate Bram 自动生成 IP 核
    - *MusicIdle：* 存储 idle.coe 的 Block RAM
    - *MusicTrue：* 存储 true.coe 的 Block RAM
    - *MusicFalse：* 存储 false.coe 的 Block RAM
    - *MusicEnd：* 存储 end.coe 的 Block RAM
  - *New：* 存放 RTL 代码的文件夹（本项目为逻辑清晰一个模块放在一个代码中）
    - ***GameTop.v：*** 顶层调度模块
    - ***GameControl.v：*** 游戏逻辑控制模块
    - ***RandomGen.v：*** 随机数生成模块
    - ***ScoreDisplay.v：*** 数码管显示实时分数模块
    - ***SoundPlayer.v：*** 无源蜂鸣器工作模块
    - ***TextLCD.v：*** 1602 型 LCD 工作模块
    - ***Effects.v：*** LED 灯工作模块
    - ***KeyPress.v:*** 键盘矩阵工作模块
- *other folder* (Vivado 自动生成，各种调试布局布线信息等)

- ***演示视频.mp4***
- ***Readme.md：*** 代码的解释性文件

本项目已于 github 上开源，可通过链接 https://github.com/Yanxiang-ZHU/Whack-a-Mole-GameMachine.git 查看源码（包括解释性文件 readme 及演示视频等）。另外也可通过百度网盘链接: https://pan.baidu.com/s/1uYaHy95xZXPiYyR3n_VMdg?pwd=asic （提取码: asic）拉取源码。