数字逻辑与数字系统设计

实验大作业报告

（2023年）

课程名称：数字逻辑与数字系统设计

任课教师：张彦航

作业题目：象棋快棋赛电子裁判计时器设计

完成人：张宇杰

学号：2022113573

班级：2203101

报告日期：　2023 年　　11 月　　29 日

报告成绩	□ 优秀：8 分；　　□良好：6.5 分；　□中等：5.5 分； □ 及格及其他：≤ 4.8 分；　　□雷同报告：0 分
评判标准	□ 报告格式规范；（2 分） □ 各部分原理讲述清晰，逻辑图准确，仿真结果正确；（2 分）， □ 有调试过程的说明，尤其是有对存在问题及解决方法的详细说明（要有截图）；（2 分） □ 设计结论客观准确，参考文献、设计心得与总结及附录等内容齐全。小组成员分工明确；（2 分）
评阅人	

目 录

报告正文

1 设计要求

1.1 基本要求

①甲乙双方的计时器为一个秒时钟，双方均用 3 位数码管显示，预定的初值均为三分钟，采用倒计时方式。通过按扭启动，由本方控制对方，比如甲方走完一步棋后必须按一次甲方的按键，该按键启动乙方倒计时。同理，乙方走完一步棋后必须按一次乙方的按键，该按键启动甲方倒计时。

②超时能发出报警判负用（可以用 F1 灯灯亮）表示。

③累计时间设置可以修改。

④比赛采用 3 局 2 胜机制，若某队员胜利，可以用 F2 灯亮表示，且双方的数码管显示各自得分（每局 3 分）

1.2 附加要求

当某方时间到时，该方侧向的灯应循环闪烁

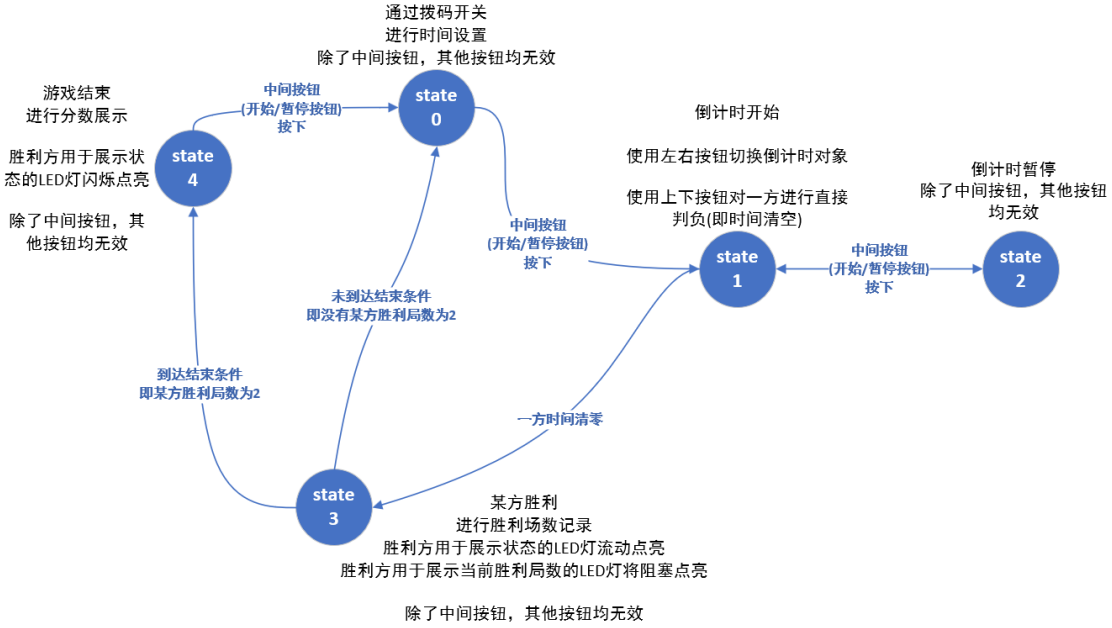
有暂停功能

在计时中也有累计胜局数显示

2 工作原理及系统方框图

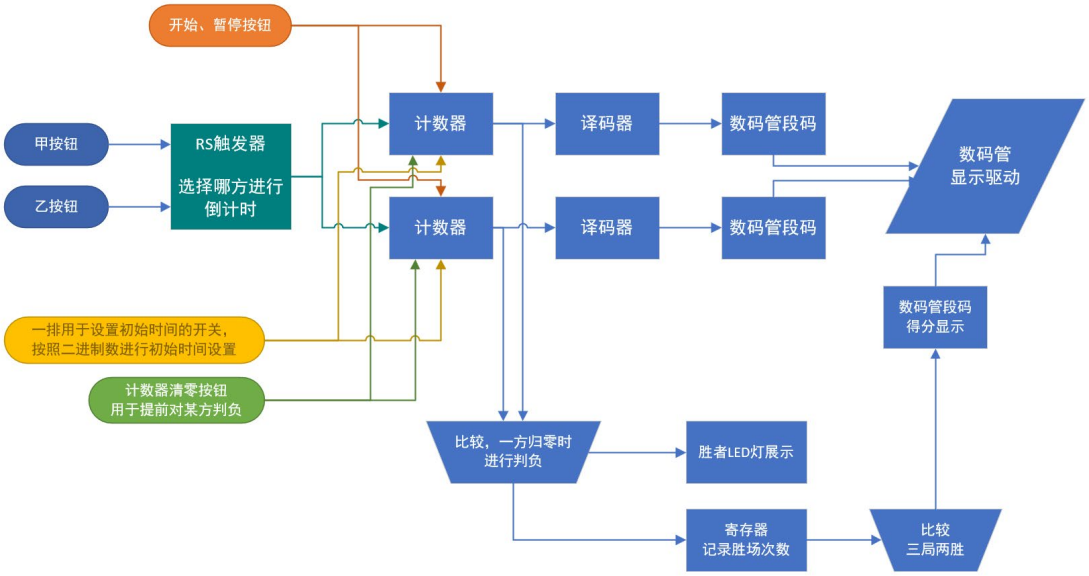
2.1 工作原理

工作过程中的状态转移过程如下图



2.2 系统方框图

各模块之间的耦合流程大致见下



3 各部分模块具体功能及设计思路

3.1 RS 锁存器

有一个，其 R 端与 S 端被连接至 A、B 两方的切换计时按钮上，用于对倒计时方的选择

模块 Verilog 定义如下：

```
module RS_Trigger(
    input wire set,
    input wire reset,
    input wire clock,
    output reg out = 0,
    output wire out_reverse
);
```

使用 case 语句来根据不同的{set, reset}组合来对 out 进行切换

3.2 计数器

有两个，用于双方时间的记录与倒计时功能

模块 Verilog 定义如下：

```
module Counter(
    input wire set_zero,
    input wire set_val,
    input wire [15:0] val,
    input wire clock,
    input wire freeze,
    input wire is_ascend,
    output reg [15:0] out = 1
);
```

带有同步清零 set_zero 端与同步置数 set_val 端，以及暂停计时的 freeze 端与切换正计时与倒计时的 is_ascend 端。

set_zero 端用于在倒计时中对其直接清零，对应实际场景就是：虽然双方时间都没有归零，但某方被将死，这时可以通过 set_zero 端直接将其时间清零。

set_val 端用于设置倒计时时长。

freeze 端用于在暂停时停止计数器的工作。

is_ascend 在本例中始终给 0 值，即倒计时功能。

3.3 总秒数转 8421BCD 码译码器

有两个，用于将总秒数转换为可用的 8421BCD 码

模块 Verilog 定义如下：

```
module Decode_16bits_to_8421(
    input wire [15:0] bits, //  总秒数
    output reg [15:0] digits
);
```

输出 digits 的含义：

模块内部把代表总秒数的 bits 转换为“分钟+秒钟”的格式，其中

digits[15:12]为分钟数的十位对应的 8421BCD 码，

digits[11:8]为分钟数的个位对应的 8421BCD 码，

digits[7:4]为秒钟数的十位对应的 8421BCD 码，

digits[3:0]为秒钟数的个位对应的 8421BCD 码

3.4 四位二选一数据选择器

有两个，用于切换数码管时间与分数的显示

模块 Verilog 定义如下：

```
module Mux_8421_2_to_1(
    input wire [3:0] num0,
    input wire [3:0] num1,
    input wire addr,
    output reg [3:0] out
);
```

3.5 8421BCD 转数码管段码译码器

有 8 个，将 8421BCD 码转成 8 位段码。有两个版本，A 为带零显示版本，而 B 为无零显示版本

两个模块 Verilog 定义如下：

```
module Nixie_Tube_Decode_typeA( //  带零显示
    input wire [3:0] Code_8421,
    input wire work,
    output reg [7:0] LEDs // . g f e d c b a
);
module Nixie_Tube_Decode_typeB( //  不带零显示
    input wire [3:0] Code_8421,
    input wire work,
    output reg [7:0] LEDs // . g f e d c b a
);
```

3.6 数码管显示驱动器

有两个，每个驱动器可控制四个数码管的显示

模块 Verilog 定义如下：

```
module Nixie_Show_Driver( //  数码管显示驱动
    input wire [7:0] num0,
```

```
    input wire [7:0] num1,
    input wire [7:0] num2,
    input wire [7:0] num3,
    input wire clk,
    output reg [3:0] addr,
    output reg [7:0] select_num
);
```

驱动器切换位码的频率为 1000Hz

3.7 状态 LED 显示驱动

有两个，控制的 A、B 双方的状态 LED 灯的点亮，有流动状态(当前游戏胜利)，以及闪烁模式(三局两胜胜利)

模块 Verilog 定义如下：

```
module LED_flow(
    input wire clk,
    input wire work,
    input wire type, // 0  为闪烁模式，  1  为流动模式
    output reg [3:0] led
);
```

LED 状态的切换频率为 5Hz

3.8 累计胜局 LED 显示驱动

有两个，控制的 A、B 双方的累计胜局 LED 灯的点亮，显示各方的当前胜利局数

模块 Verilog 定义如下：

```
module LED_score(
    input wire work,
    input wire [1:0] value,
    output reg [2:0] led
);
```

3.9 整体综合模块

模块 Verilog 定义如下：

```
module Big_Project(
    input wire clk,
    input wire [5:0] initial_minutes,
    input wire set_zero_A, //  清空对应时钟
    input wire set_zero_B,

    input wire btn_A,
    input wire btn_B,
    input wire btn_start_and_pause,

    output wire [7:0] nixie_seg0,
    output wire [3:0] nixie_addr0,
    output wire [7:0] nixie_seg1,
    output wire [3:0] nixie_addr1,
```

```
                output wire [3:0] state_led_A,
                output wire [3:0] state_led_B,
                output wire [2:0] score_led_A,
                output wire [2:0] score_led_B
            );
```

模块内部使用了以下寄存器变量：

```
reg [2:0] state = 0;
// 0 为正在设置时间，1 为正在倒计时，2 为暂停中，3 为本小局游戏结束，4 为本局游戏结束
reg [1:0] win_round_A = 0, win_round_B = 0;
// 各自的胜利局数
reg pause = 0;
// 是否正在暂停
reg set_time = 0;
// 是否设置时间
reg [31:0] btn_sp_counter = 32'd100_000_000;
// btn_start_and_pause 两次响应时间间隔为 1 秒
reg [7:0] nixie_work_state = 8'b11111111;
// 8 个数码管的工作状态
reg led_flow_type = 0;
// 状态 led 的流动模式
reg led_flow_work_A = 0, led_flow_work_B = 0;
// 各个状态 led 的工作使能
reg nixie_model = 0;
// 数码管输出模式，0 为时间输出，1 为分数输出
```

模块中除了各元件的例化，还有一个主体 always 语块，其伪代码见下：

```
// 根据当前状态进行相应操作
always @(posedge clk) begin
    …
    case (state)
        3'd0: // 正在设置时间
            ……
            # 在设置时间时按下了 btn_start_and_pause 按钮
                state = 1; // 计时开始
        3'd1: // 正在倒计时
            ……
            # 在计时时按下了 btn_start_and_pause 按钮
                state = 2; // 计时暂停
            # 一方时间清零
                state = 3; // 计时结束
        3'd2: // 正在暂停
```

```
                # 在暂停时按下了 btn_start_and_pause 按钮
                    state = 1; // 计时继续
            3'd3: // 一方时间清零
                # 对胜利方胜利次数进行累计
                # 三局两胜
                    state = 4; // 本局游戏结束
                # 三局未两胜
                    state = 0; // 本局游戏尚未结束
            3'd4: // 本局游戏结束
                # 按下了 btn_start_and_pause 按钮
                    state = 0;
        endcase
    end
```

4 调试过程

4.1 数码管显示数值不正确，全为 0

在 Vivado 综合时发现以下错误：

[Synth 8-6859] multi-driven net on pin P[11] with 1st driver pin 'set_seconds0/P[11]'

报错处原代码为：

```
assign set_seconds = initial_minutes * 60;
```

分析得知，set_seconds 为 16 位变量，但 initial_minutes 为 6 位变量，位宽不匹配，导致计算溢出

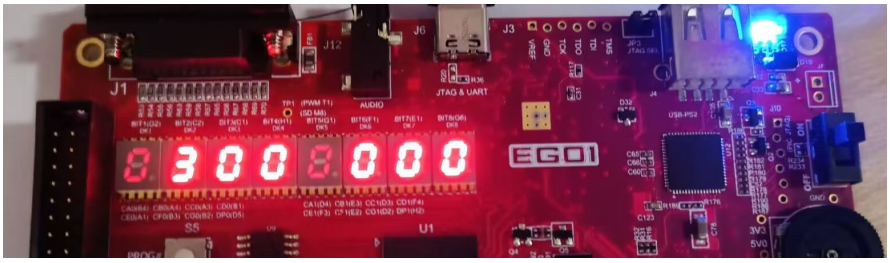
将其修改为：

```
assign set_seconds = {{10{1'b0}}, initial_minutes[5:0]} * 16'd60;
```

后有正确的数值显示。

4.2 数码管显示数值正确，但只有一方获得了初值

如下图，只有左侧的数值随着拨码开关变化，右侧始终是零：



分析得知，先前对于两个计数器模块的连接见下：

```
Counter u_counter_A(…,   clk & rs_out_reverse,   pause,…);
Counter u_counter_B(…,   clk & rs_out,   pause, …);
```

原本的想法是将 RS 锁存器的输出与计数器的时钟端相与，从而达到控制一个计数器不动，而另一个计数器倒计时的效果。

但分析发现，当没有了时钟信号的变化，Counter 模块中的 always @(posedge clock)根本不会执行，从而导致内部的 set_val 相关语句不被执行，从而计数器始终显示初值 0。

这是逻辑错误，将模块连接改为如下后，数码管就表现正常了：

Counter u_counter_A(…, clk, pause | rs_out,…);

Counter u_counter_B(…, clk, pause | rs_out_reverse, …);

4.3 中间按钮的表现不尽人意，出现抖动

解决方法，两次按钮按下的时长需大于 0.5 秒才能被响应

4.4 锁存器错误

综合时 Vivado 发现以下错误：

[Synth 8-327] inferring latch for variable 'LEDs_reg'

查阅资料后得知，这是在组合逻辑中被综合出锁存器的警告

解决方法，if 后应该有 else，case 内部要有 default，在每个分支都对输出进行赋值，防止锁存器的出现

4.5 计数器只增不减

原相关代码为：

```
……
        if (!freeze) begin
            if (is_ascend)
                if (out != 16'hFFFF)
                    out = out + 16'b1; //正计数
            else
                if (out != 16'b0)
                    out = out - 16'b1; //倒计数
        end
……
```

其中加粗斜体下划线的 else 优先被最近的 if (out != 16'hFFFF)捕获，导致运行结果异常，表现为 out = out - 16'b1 永远不在 is_ascend 为 0 时被执行

修改如下：

```
……
            if (is_ascend) begin
                if (out != 16'hFFFF)
                    out = out + 16'b1; //正计数
            end
            else
                if (out != 16'b0)
                    out = out - 16'b1; //倒计数
……
```

这告诉我们别偷懒不写单行 if 中的 begin end

5 设计结论

本次设计的象棋快棋赛电子裁判计时器，将拨码开关(SW0-R1~SW5-P3)作为初始时间分钟数的 6 位二进制输入，支持时长为 0 至 63 分钟。

将 8 个数码管分为两组，左边 4 个为玩家 A 的剩余时间及分数显示，右边 4 个为玩家 B 的剩余时间及分数显示。使用数据选择器来复用了数码管模块，让其既能显示时间也能显示得分。此外，每组(4 个)数码管的最高位(即分钟数的十位)被设计为无零显示，使得当分钟数小于零时让数码管有更好的可读性。

使用了左右两个按钮来作为计时方的切换，左方按钮对应 A 玩家的按钮，A 按

下时即可暂停 A 的计时，开启 B 的计时。右方按钮反之。

使用了上下两个按钮来将 A，B 方提前判负，符合实际情况也便于调试。

当一方单局胜利时，其方侧的 LED 将会流动显示，作为胜方的展示。

在进行游戏的过程中，板卡中央偏右下的紧凑 LED 灯用于记录双方的累计胜利次数，一方侧亮了几个灯该方侧就赢了几局。

当一方累计赢局达到 2 时，游戏结束，数码管显示各自的得分，胜方侧的 LED 灯将闪烁显示，作为胜方的展示。

本人觉得，本次的设计无论是在功能上还是在可视化上都已经算是比较的完善了，算是一个比较满意的设计。

6 设计心得与总结

6.1 模块化的重要性

将整个系统划分成若干子模块是非常重要的。模块化的过程是对整个问题分析、理解的过程，也有助于理清整个问题的实现思路。

模块化最大的好处是，在出现问题时，我们可以对逐个模块进行分别测试，提高寻找问题效率，同时测试时还可以跳过一些简单、绝对正确的模块，进一步提高开发效率。

除此之外，模块化还有一个好处，如果想添加新的功能，只需要在相应模块里修改即可，而无需大幅变动其他代码。

6.2 注释的重要性

在编写一个较大的工程时，如果没有及时添加注释，很容易就会忘记一个变量的含义，或者一段语句的作用。添加注释既能提高可读性，还能理清自己的思路。

6.3 测试不上板，就是耍流氓

很多问题在仿真时是不会发现的，但在上板后就成为了致命问题，如上述的按钮抖动问题。此外，在板子上也能更直观的发现一些问题，比如数码管相关问题，单看波形图只会让人头疼。

6.4 规范的代码习惯

比如上述的 if 后要跟 else，case 内要有 default，从而避免在组合逻辑中却综合出锁存器的情况。再如上述的单语句 if 也不要省去 begin end。再如上述中运算的位宽问题，要保证操作数与结果的位宽一致。养成良好的代码习惯，能够将大量 bug 扼杀在摇篮之中。

6.5 个人总结

这次的大作业让我体验了一把“硬件编程”，与早已习惯的高级语言不同，使用 Verilog 编程时还需要考虑到代码是否能对应到实际的硬件上，这需要遵守一定的代码编写规则，否则写了半天代码的收获是“无法综合”就只能默默吐血了。

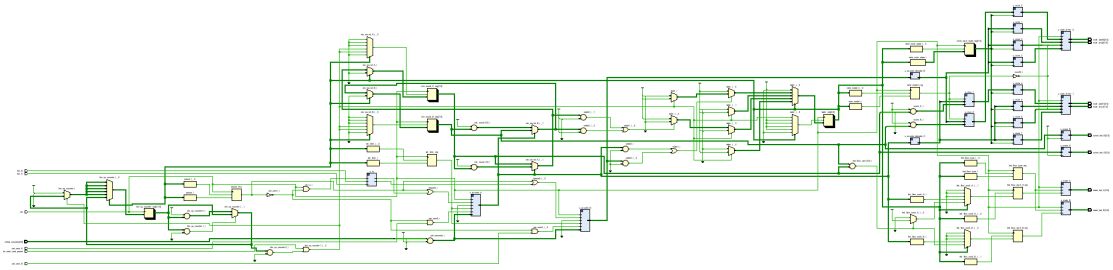
虽然于高级语言有着很多不同之处，但是其内在也是有很多相通之处的。比如良好的编程习惯，以及及时添加注释，甚至与模块化的设计。这让我在进行 Verilog 编程时也感受到了一种“亲切感”，使我感觉到先前在我面前充满“神秘感”的硬件也没有那么“神秘”了。

参考文献

EGo1 用户手册 https://e-elements.readthedocs.io/zh/ego1_v2.2/EGo1.html#i-o
其他均为独立完成，未参考相关文献

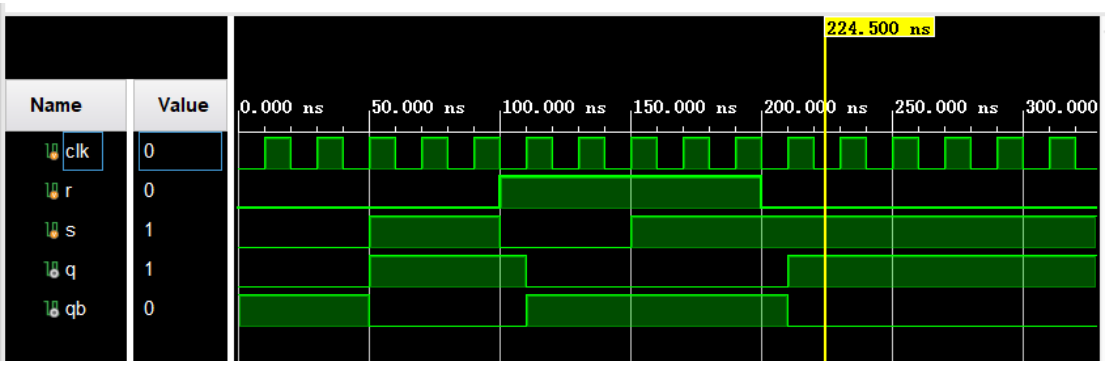
附录

附录一：总体设计图

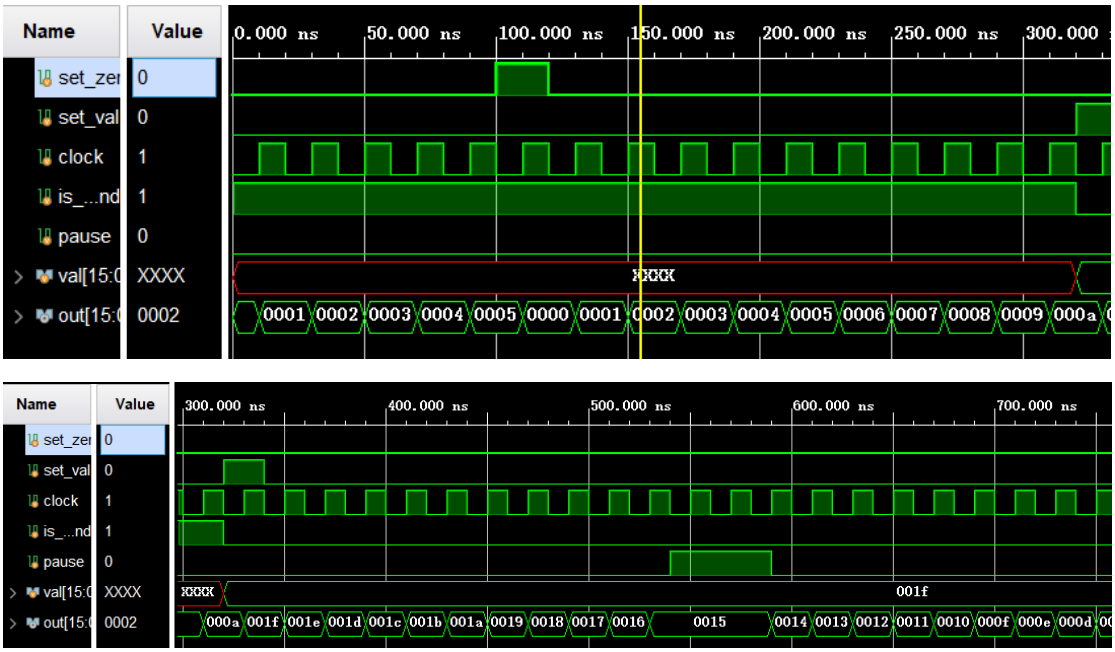


附录二：各模块仿真截图

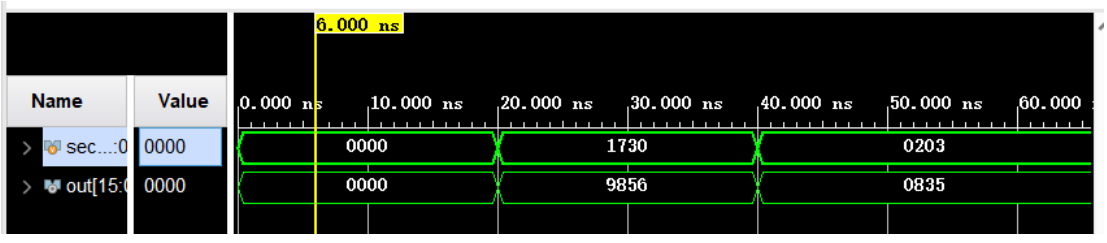
1、RS 锁存器的仿真波形



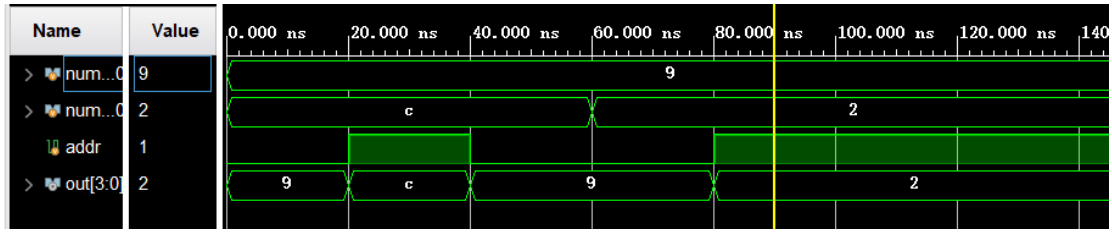
2、加减计数器的仿真波形



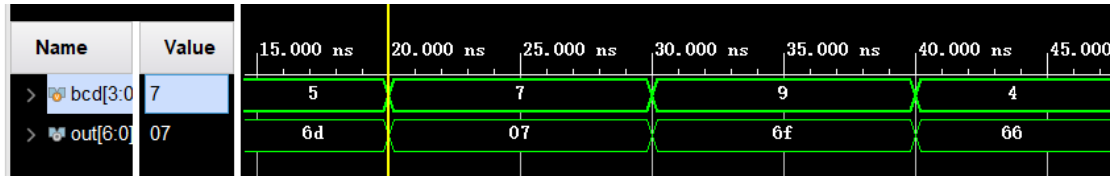
3、总秒数转 8421BCD 码译码器的仿真波形



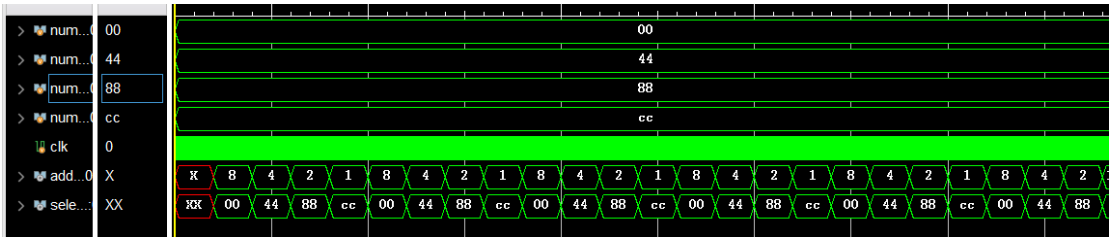
4、四位二选一数据选择器的仿真波形



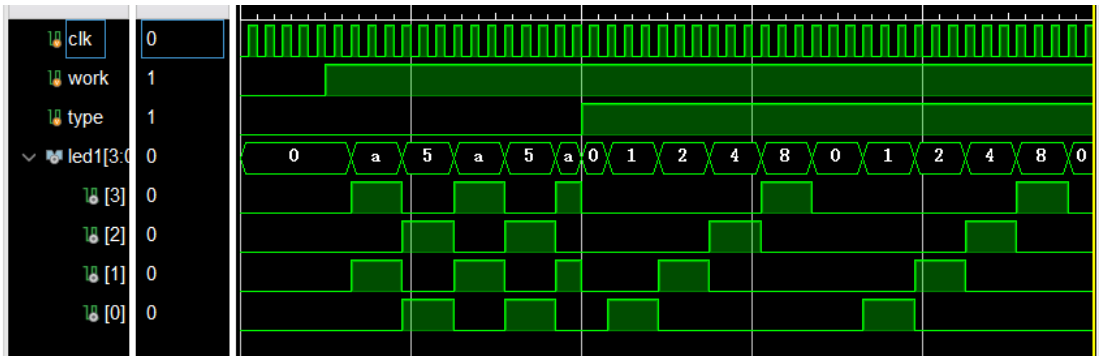
5、8421BCD 转数码管段码译码器的仿真波形



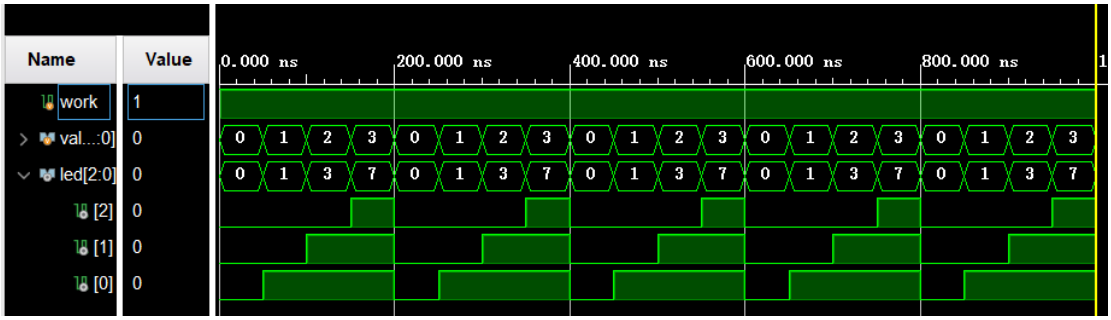
6、数码管显示驱动器的仿真波形



7、状态 LED 显示驱动的仿真波形



8、累计胜局 LED 显示驱动的仿真波形



附录三：小组各成员所做工作说明

所有工作由本人独自完成