北　京　邮　电　大　学

数字电路实验报告

实验名称：“移动靶”游戏的设计

学　　院：　　信息与通信工程

姓　　名：　　王思恢

班　　级：　　2016211110

学　　号：　　2016210271

班内序号：　　03

一、设计课题的任务要求：

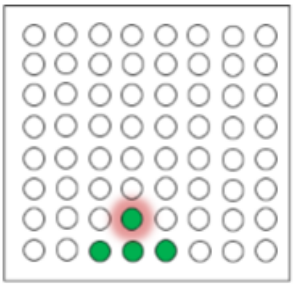
设计并实现一个“移动靶”游戏。

基本要求：

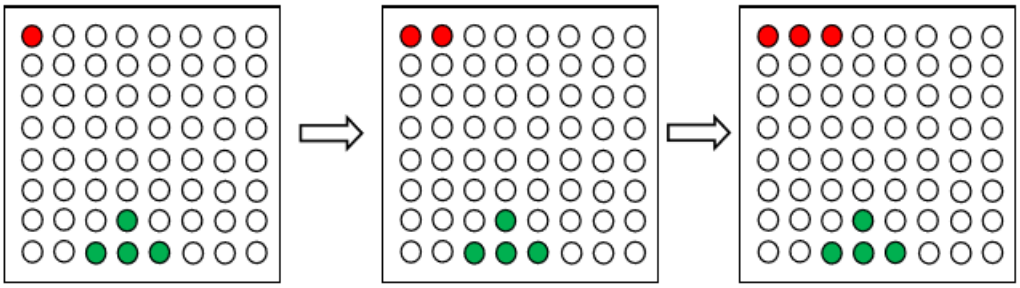
1.用 SW7 作为系统开关，LED0 作为电源指示灯，打开 SW7，电源指示灯亮，系统上电。同时显示器件自检：8×8 点阵以 5Hz 频率进行自上而下的单行扫描显示，8 位数码管以 5Hz 频率进行从左至右的位扫描显示，重复三个循环后全灭，进入待机状态；

2.使用按键 BTN0 进入游戏状态：

a) 在 8×8 点阵的底部用 3s 时间渐亮显示四个点组成的绿色三角形状，代表射击枪，其中顶端的点代表枪口位置；



b) 枪口显示稳定后，在最高一行以滚入滚出方式滚动显示三个点组成的红色“移动靶”（滚入方式如下图，滚出同理），同时数码管最左边两位显示 40 秒倒计时，最右边显示游戏得分“00”，移动靶滚动速度可用按钮调节，BTN7 加速，BTN6 减速；



c) 按键 BTN1 为射击按钮，按动射击按钮，枪口可以发射出红色子弹，子弹沿直线向上飞行，飞行速度为 0.1s/行，子弹飞行过程中，“移动靶”应正常移动；

d) 子弹击中“移动靶”任意位置，游戏得分，得分显示+3，计满 15 分时，得分闪烁显示且点阵显示“√”，本轮游戏结束；

e) 若游戏时间倒计时为“00”时得分未计满 15 分，则“00”闪烁显示且点阵显示“x”，本轮游戏结束；

3.游戏过程中或游戏结束后按 BTN0 键，进入新一轮游戏。

提高要求：

1.自检时、游戏过程中、一轮游戏结束时分别伴有不同的背景音乐和音效；

2.分等级计分：子弹击中“移动靶”中间的点计 3 分，击中两边的点计 2 分；

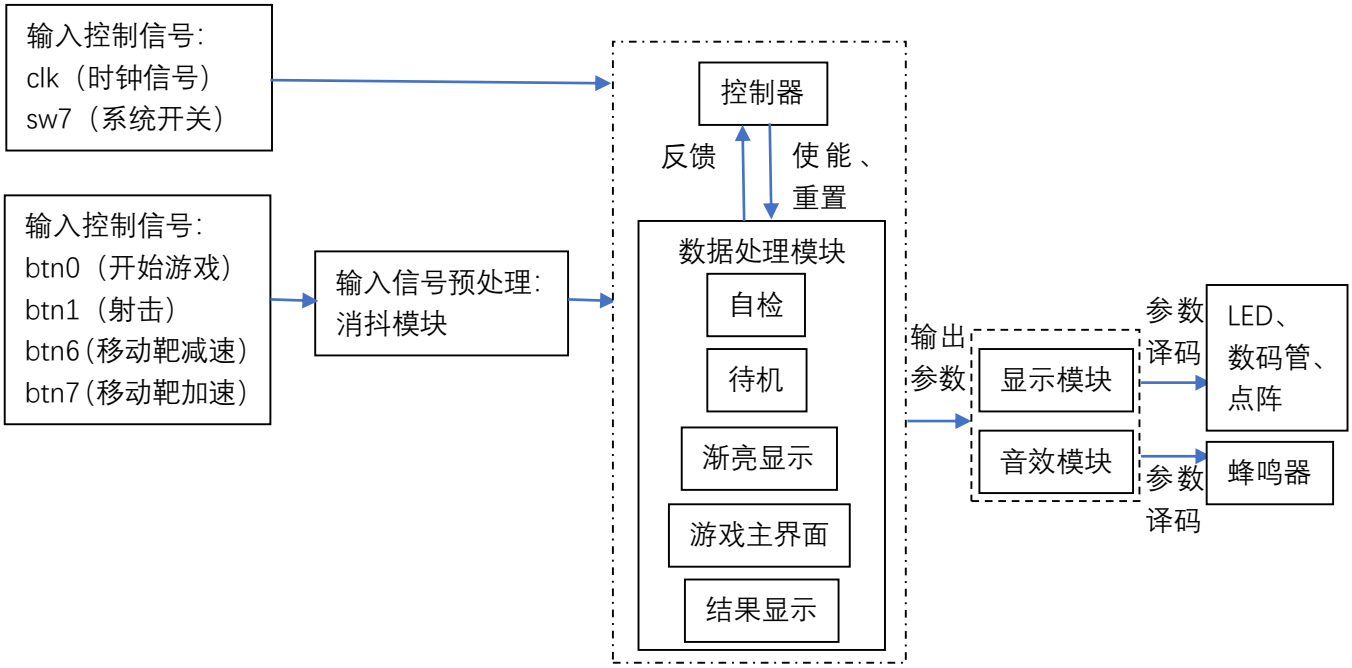
3.“移动靶”位置随机变化；

4.自拟其他功能。

二、系统设计：

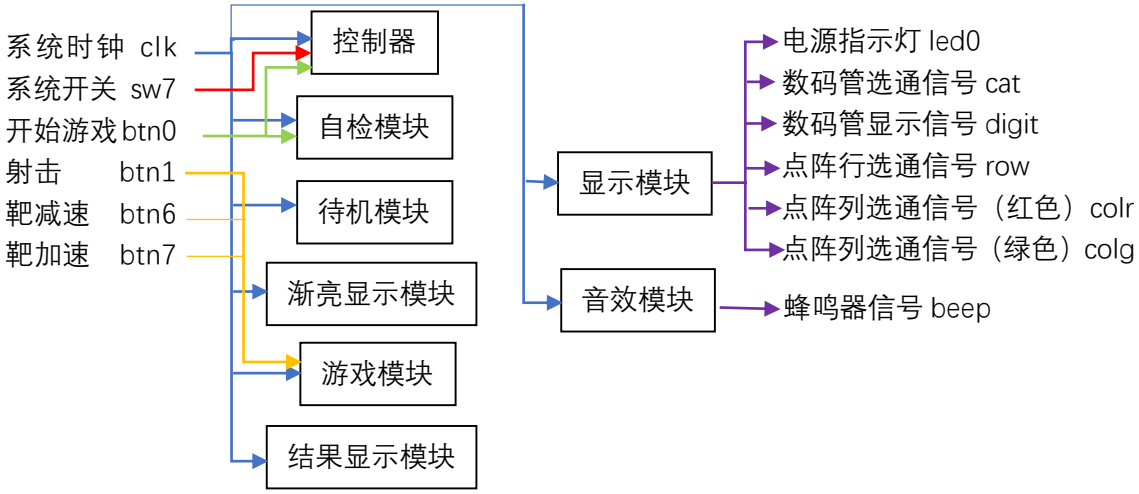
1.总体设计：

根据移动靶游戏的设计要求，我把移动靶游戏电路分解为如下模块：



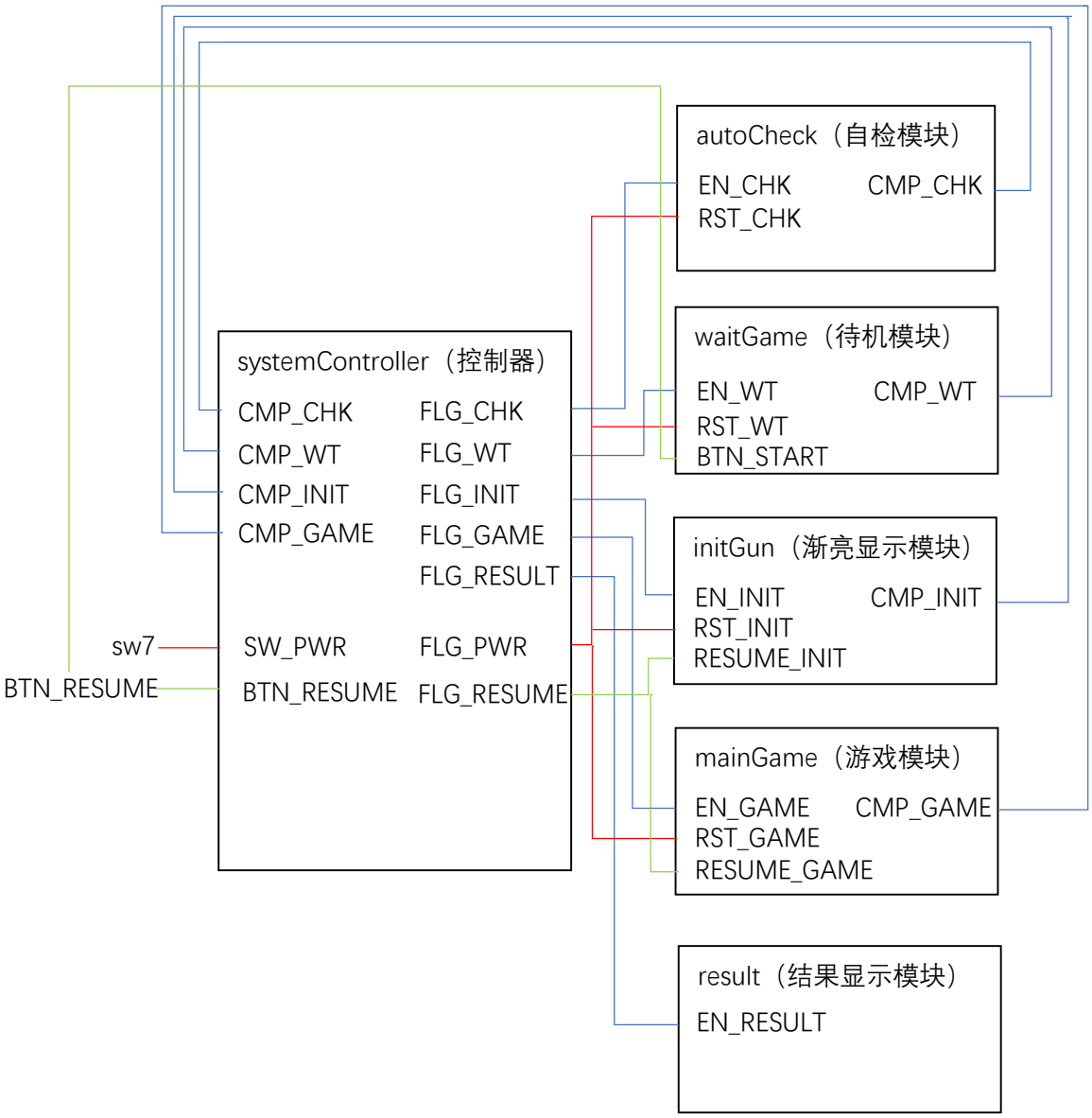
移动靶游戏电路设计方框图

该电路的对外接口为：



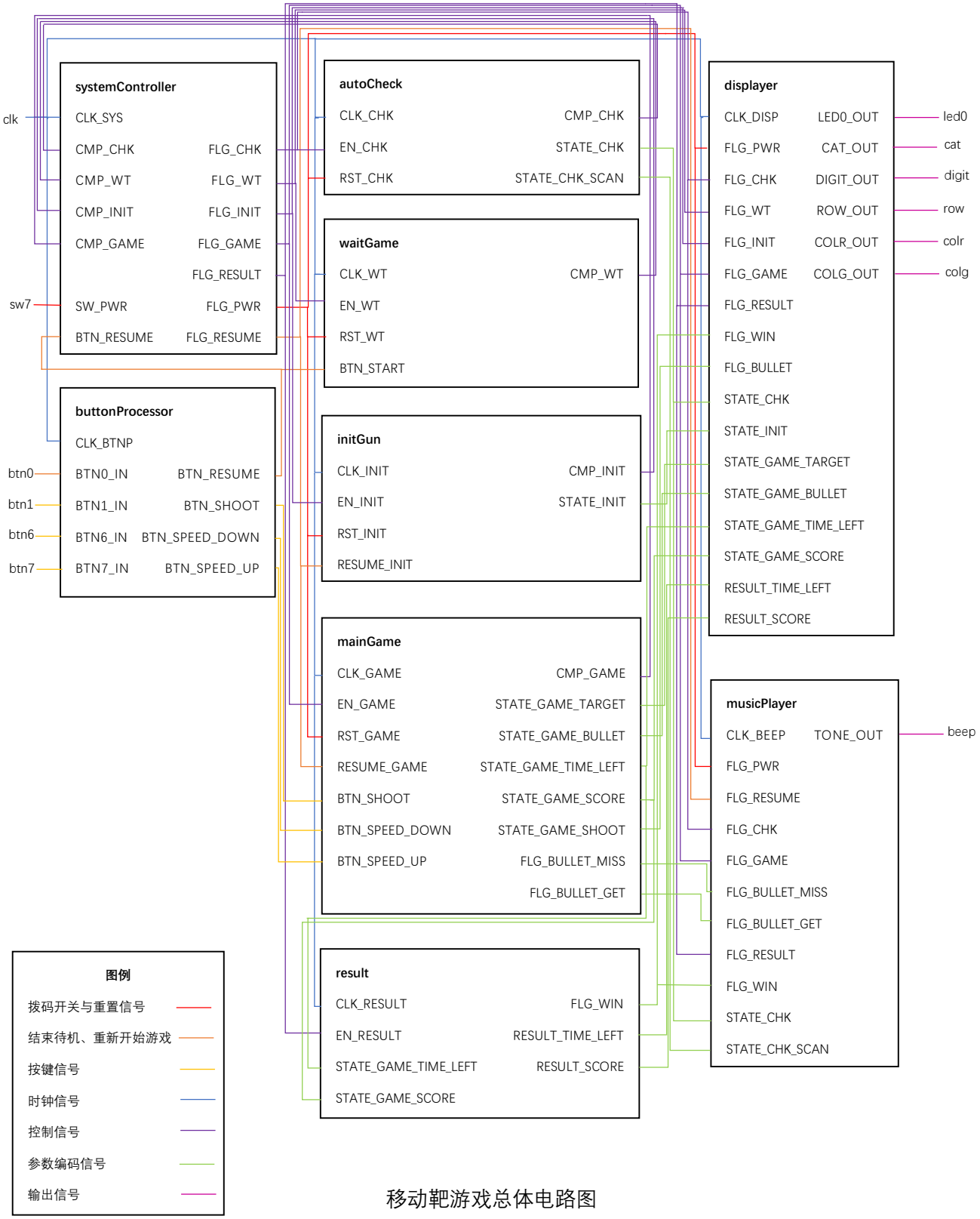
移动靶游戏电路对外接口图

其中控制器与各游戏进程模块之间的连线如下图所示：



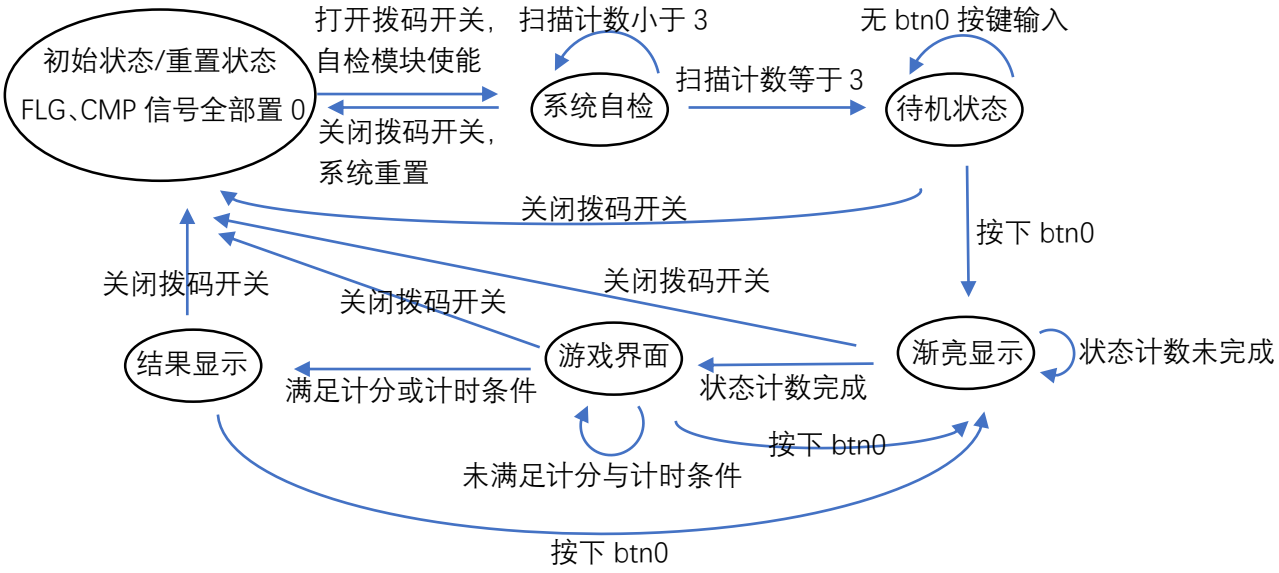
游戏控制电路原理图

数字电路的整体电路图如下图所示：



移动靶游戏总体电路图

根据移动靶游戏的功能描述，我们可以画出移动靶游戏的状态图：



移动靶游戏状态转移图

由此我们勾勒出了移动靶游戏的整体设计图纸，下面开始进行控制电路与功能电路的具体设计。

2.控制电路的设计：

在移动靶游戏设计中，我们通过控制器与游戏进程各模块的交互来实现对于游戏进程的控制。

当拨码开关保持打开，而且在游戏进程中“重新开始游戏”的按键 BTN0 没有按下时，游戏进程将顺次执行。每个游戏进程模块执行结束后，该模块将把完成标记置 1，控制器读取到该模块的完成标记信号为 1 后，就把此模块使能端置 0，把下一个模块的使能端置 1，这就实现了各模块的依次使能，也就实现了游戏进程的顺次执行。

控制器通过各个模块的完成标记来判断当前应当使能哪个模块。自检模块完成标记 CMP_CHK、待机模块完成标记 CMP_WT、渐亮显示模块完成标记 CMP_INIT、游戏主界面模块完成标记 CMP_GAME 构成一个向量，控制器将检测该向量的哪个分量为 1，第一个完成标记值不为 1 的模块就是当前应当运行的进程，控制器将选择将该模块的标志位置 1，使得该模块使能。

当“重新开始游戏”的按键 BTN0 按下时，游戏进程将发生跳转。控制器检测到 BTN0 的按键信息后，将把将把 FLG_RESUME 标志位置 1，该信号使得渐亮显示模块将 CMP_INIT 置 0，游戏主界面模块将 CMP_GAME 置 0，这样，渐亮显示模块就成为第一个完成标记值不为 1 的模块。根据控制器的逻辑，此时将把 FLG_INIT 置 1，使能渐亮显示模块，这样，就实现了游戏进程的跳转。

当拨码开关关闭后，控制器将把 FLG_PWR 置 0，该信号使得全部模块的完成标记清零。这样，再次打开拨码开关时，控制器将检测到全部完成标记为 0，此时，控制器将使能第一个模块，也就是自检模块。这样，游戏就将从开机自检开始，重新运行。

通过上述逻辑设计，我们就实现了控制器对于各游戏进程模块的控制。

3.显示模块的设计：

在本次数字电路设计中，我们设立显示模块，使其独立于任何一个游戏进程模块。在游戏过程中，控制器将各个模块的运行标志信号 FLG_CHK、FLG_WT、FLG_INIT、FLG_GAME、

FLG_RESULT 发送给显示模块，使得显示模块了解当前的游戏进程。此外，各个游戏进程模块将与显示有关的参数发送给显示模块，使得显示模块得以绘制相应的图案。

例如，显示模块了解到 FLG_CHK 为 1 时，将读取自检模块发送的显示参数，根据相应的参数值，选择选通哪一行点阵以及哪一位数码管。

当显示模块得知 FLG_INIT 为 1 时，将读取渐亮显示模块发送的显示参数，该参数表示渐亮显示模块所处的阶段数。显示模块将根据该阶段数，决定显示射击枪的波形的占空比，从而决定射击枪显示的亮度。

当显示模块得知 FLG_GAME 为 1 时，将从游戏主界面模块获取移动靶位置、子弹位置、子弹是否处于飞行状态、倒计时信息、分数信息等参数，根据上述参数，判断当前应当绘制何种图案，并将图案用逐行扫描的方式进行呈现。

当显示模块得知 FLG_RESULT 为 1 时，将从结果显示模块获取游戏胜负、剩余时间、游戏总分 3 个参数，通过上述参数，决定绘制何种图案，并进行逐行、逐位的呈现。

4.音频播放模块的设计：

与显示模块类似地，本设计将音效播放功能设置为独立于各游戏进程的单独模块。音频模块同样接收控制器发送的各模块运行标志信号 FLG_CHK、FLG_WT、FLG_INIT、FLG_GAME、FLG_RESULT，判断当前运行的游戏进程，并接收相应游戏进程发送的与音频播放有关的参数。

当 FLG_CHK 为 1 时，音频播放模块接收自检模块发送的行扫描信息与扫描次数信息，由行扫描信息 0~7 确定所播放音符为 Do、Re、Mi 直到高八度的 Do，由扫描次数信息确定所播放的音符在哪个八度上。

当 FLG_GAME 为 1 时，音频播放模块将通过一个循环计数器控制音乐旋律的循环播放。当子弹飞行至底线时，程序将判断子弹是否击中移动靶，并将标志位 FLG_BULLET_GET 与 FLG_BULLET_MISS 分别置 0 或置 1。检测到 FLG_BULLET_GET 为 1 或者 FLG_BULLET_MISS 为 1 时，音效播放模块将暂停音乐播放的循环计数器，启动命中音效的计数器或者未命中音效的计数器，播放一个由 4 个十六分音符组成的音效，然后控制此音效播放的计数器清零，以等待下一次子弹命中或未命中的事件触发，而控制音乐播放的计数器从停止的地方继续开始计数，使得游戏的背景音乐从刚才中断的地方开始继续播放。

当 FLG_RESULT 为 1 时，音频播放模块将读取 FLG_WIN 的信息，根据胜负情况，分别播放胜利音效（Do-Sol-高音 Do）或失败音效（高音 Do-Sol-Do）。

5.开机自检模块的设计：

在开机自检模块内部，维护了两个循环计数器。一个计数器负责生成行扫描信息，其在 5Hz 时钟分频信号的驱动下，从 0~7 循环计数。另一个计数器负责生成扫描次数信息，行扫描计数器每从 0~7 循环一次，扫描次数计数器就加 1，当扫描次数计数器的值达到 3 时，开机自检模块向控制器发出 CMP_CHK=1 的信号，随后控制器令开机自检模块失能，自检结束。

行扫描信息和扫描次数信息将转化为输出信号，控制显示模块与音效播放模块的功能。

6.待机模块的设计：

待机模块接收 BTN0 发出的 BTN_START 即开始游戏信号。接收到 BTN_START 信号后，待机模块向控制器发出 CMP_WT=1 的信号，随后控制器令待机模块失能，待机过程结束。

7.渐亮显示模块的设计：

渐亮显示模块内部维护了一个 0~7 的计数器，该模块从 0 开始计数，每隔 0.375 秒，该计数器的计数值加 1，这样，计数器的值从 0 增加到 7，共维持 3 秒的时间。该计数信号对外输出给显示模块，显示模块利用该信号的值控制扫描显示射击枪的波形占空比，从而调控射击枪的显示亮度。3 秒钟结束时，渐亮显示模块向控制器发出 FLG_INIT=1 的信号，随后

控制器令渐亮显示模块失能，渐亮显示过程结束。

8.游戏主界面模块的设计：

游戏主界面模块内部维护了两个分频计数器。一个分频计数器 bulletPosition 由 0~6 计数，固定地每隔 0.1 秒更新一次，该计数器用于生成子弹飞行的位置信息。另一个分频计数器 targetPosition 由 0~9 循环计数，该计数器的更新频率可以调控，该计数器用于生成移动靶的位置信息。按下 BTN6 或 BTN7 时，该计数器的更新频率分别降低或升高，从而控制了移动靶的移动速度减慢或加快。

在未按下 BTN1 时，控制子弹飞行位置信息的计数器 bulletPosition 的计数值一般为 0，此时内部信号 flagBullet=0，表示显示模块无需绘制子弹。按下 BTN1 后，flagBullet 置 1，表示显示模块需要绘制子弹，bulletPosition 开始从 0~6 计数。当 bulletPosition 达到 6 之后，再经过 0.1 秒，bulletPosition 将自动清零，并将 flagBullet 置 0。这样，就使得在没有子弹射出的情况下，bulletPosition 的值“锁死”为 0，且子弹不会被显示模块绘制，只有按下 BTN1 后，bulletPosition 将一次性地从 0~6 计数，并且令显示模块绘制子弹信息。随后，bulletPosition 和 flagBullet 均清零，等待下一次 BTN1 按键的触发。

每次子弹到达底线时，本模块会判断移动靶的位置，以确定子弹是否击中移动靶以及击中了移动靶的边缘或中心，从而更新计分信息。当计分或计时信息达到胜利条件或者失败条件时，CMP_GAME 置 1，随后控制器令游戏主界面模块失能，游戏主界面进程结束。

9.结果显示模块的设计：

游戏结束后，结果显示模块从游戏主界面模块读取剩余时间信息与分数信息，判断游戏胜负，并将胜负、时间、分数 3 个信息输出给显示模块和音效模块，用于相应效果的展示。

10.按键消抖模块的设计：

按键消抖模块用于接收 BTN0、BTN1、BTN6、BTN7 的按键信息，经过处理生成消抖信号。

由于按键造成的抖动一般在 20 毫秒之内，按键消抖模块每隔 20 毫秒检测一次按键按下的信息，如果有按键按下，则生成一个相应按键信号的宽度为 20 毫秒的高电平方波信息。在游戏主界面模块中，每隔 20 毫秒才检测一次该方波信息，这就使得每次按键恰好能被检测到，也只能被检测到一次，这就实现了按键消抖的功能。通过实际测试，该方法有效。

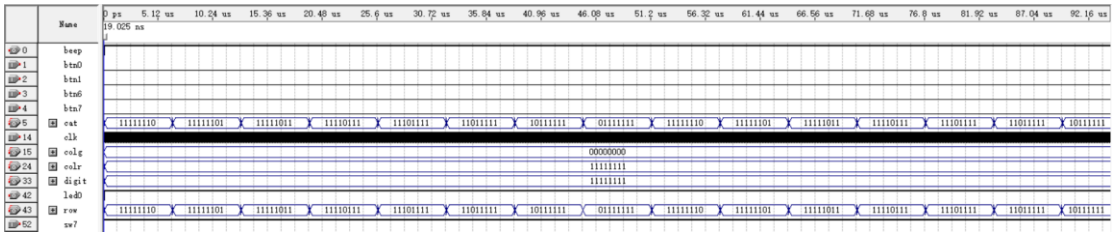
三、仿真波形与波形分析：

1.仿真参数设定：

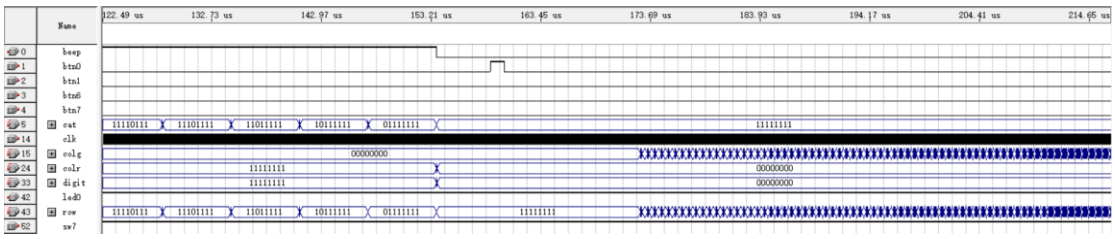
为了减少计算机的运算负荷，在有效时间内得到能够反映电路实际运行情况的仿真波形，我们在仿真文件中，将参数设定如下：

令一个系统时钟周期为 40ns，用 800 个系统时钟周期仿真表示真实世界中的 1 秒，其余参数按照上述设定进行相应调整。

2.自检模块的仿真测试：



如图所示，在开机自检阶段 led0 输出高电平，表示 LED0 点亮；colg 为"00000000"，表示绿色点阵不点亮，colr 为"11111111"，表示红色点阵各列均选通，digit 为"11111111"，表示数码管 AA~AG 以及 AP 均选通，即选通位的数码管应当显示"8."。row 与 cat 信号同步地由"11111110"变化至"01111111"，在这里，我们定义的 row 与 cat 数组均为下标渐增型，即"11111110"中的"0"表示 row7 与 cat7,分别为点阵的最上面的一行与数码管的最左边一位。这样，从仿真波形中我们可以看出，点阵从上到下各行依次选通，数码管各位从左到右依次选通。



根据仿真中的 32 微秒等于现实世界的 1 秒可以计算出，自检应当持续 32*0.2*24=153.6 微秒。由仿真波形图可以验证这一点，说明自检模块在点阵与数码管循环扫描 3 次后关闭。

3.待机模块的仿真测试：

从上图我们还可以看出，自检结束后，我们给 btn0 加了一个高电平脉冲，这模仿了显示世界中按下 BTN0 的操作。我们看到，按下 BTN0 前，cat 与 row 均为"11111111"，表示数码管与点阵均全灭，按下 BTN0 后，row 和 colg 信号开始出现变化，说明点阵开始显示绿色的射击枪，渐亮显示模块开始工作。由此，我们验证了待机模块的仿真性能。

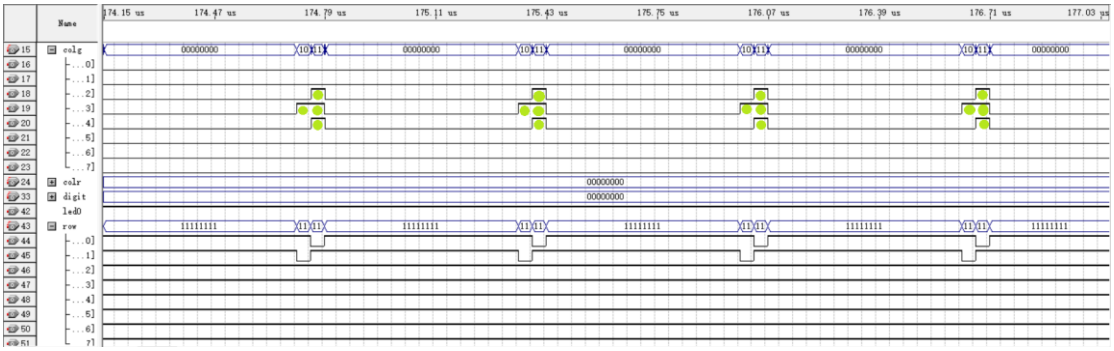
4.渐亮显示模块的仿真测试：

根据我们的代码设计，渐亮显示的 3 秒钟被分为 8 个阶段，在第 1 个阶段，射击枪不显示；在第 2 个阶段，点阵在 7/8 的时间内全灭，在 1/8 的时间内显示射击枪；在第 3 个阶段，点阵在 6/8 的时间内全灭，在 2/8 的时间内显示射击枪，以此类推，直到第 8 个阶段，点阵在 1/8 的时间内全灭，在 7/8 的时间内显示射击枪。

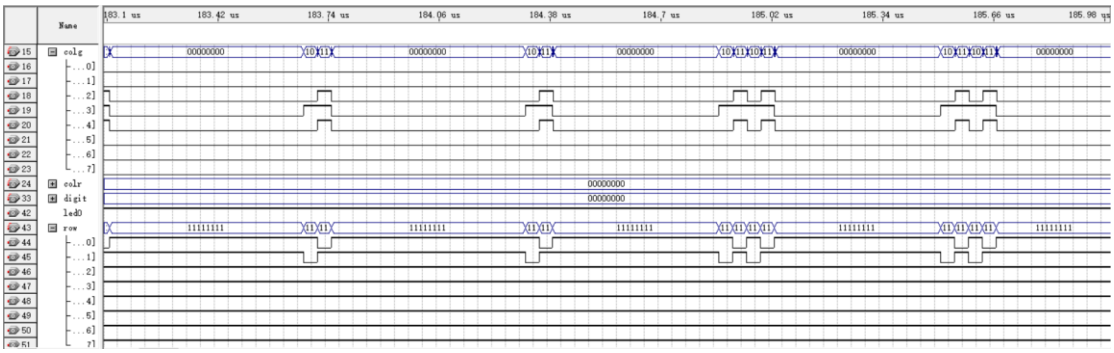
从上面的仿真波形图中我们已经可以看出，按下 BTN0 后，row 仍在一段时间内保持"11111111"即全灭的状态，这就是渐亮显示的第 1 阶段的波形图。

如下图所示，在 7/8 的时间内，点阵行选通信号 row 为"11111111"，这表示点阵在 7/8 的时间内全灭，在剩余 1/8 的时间内，我们可以看到 row(0)与 row(1)交替选通，通过对应时

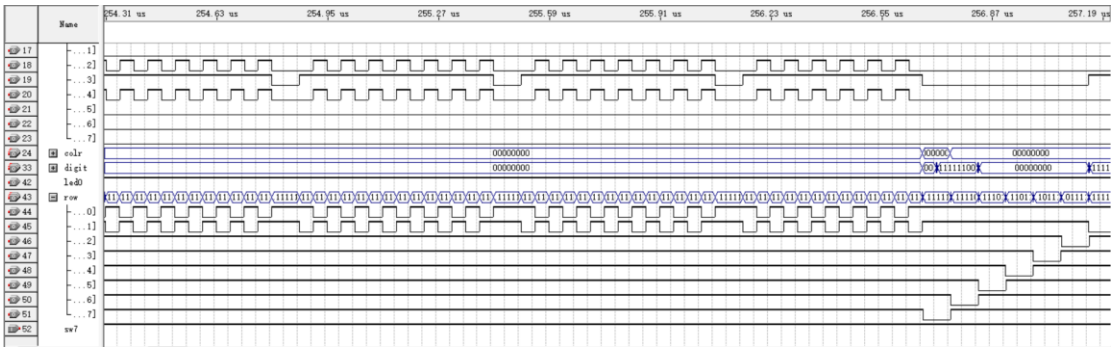
刻绿色点阵列选通信号波形 colg，我们可以确认，点阵在这 1/8 的时间内，通过交替扫描的方式显示了绿色的射击枪，这就是渐亮显示模块第 2 阶段的工作波形仿真图。



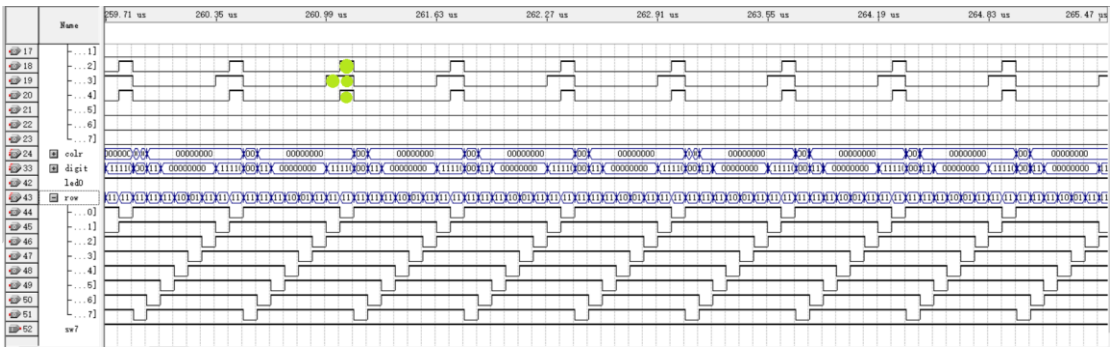
从下图中我们可以看到，点阵行选通信号的占空比发生了变化，点阵在更多的时间内显示射击枪，在更少的时间内处于全灭状态，这就验证了渐亮显示的效果。



从下图我们可以看到，在渐亮显示模块工作的第 8 个阶段，点阵在 7/8 的时间内处于扫描显示状态，在 1/8 的时间内处于全灭状态。随后，点阵行扫描信号波形发生变化，row(7)至 row(0)开始依次导通，这说明渐亮显示进程结束，游戏进入主界面进程。

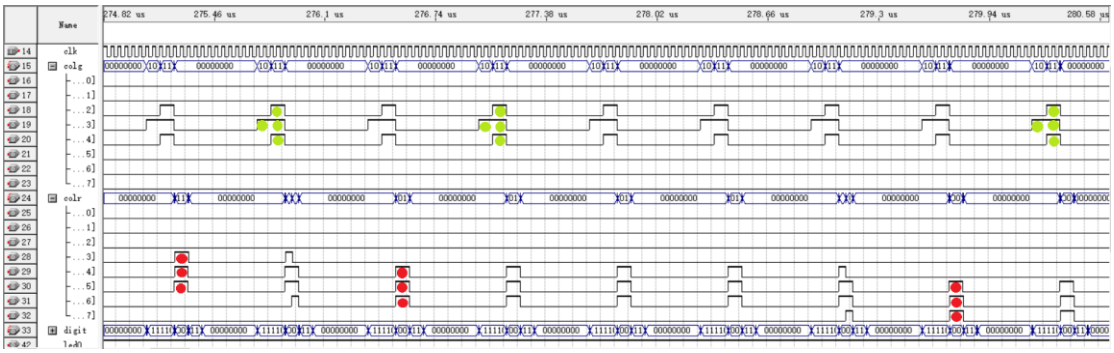


5.游戏主界面模块的仿真测试：

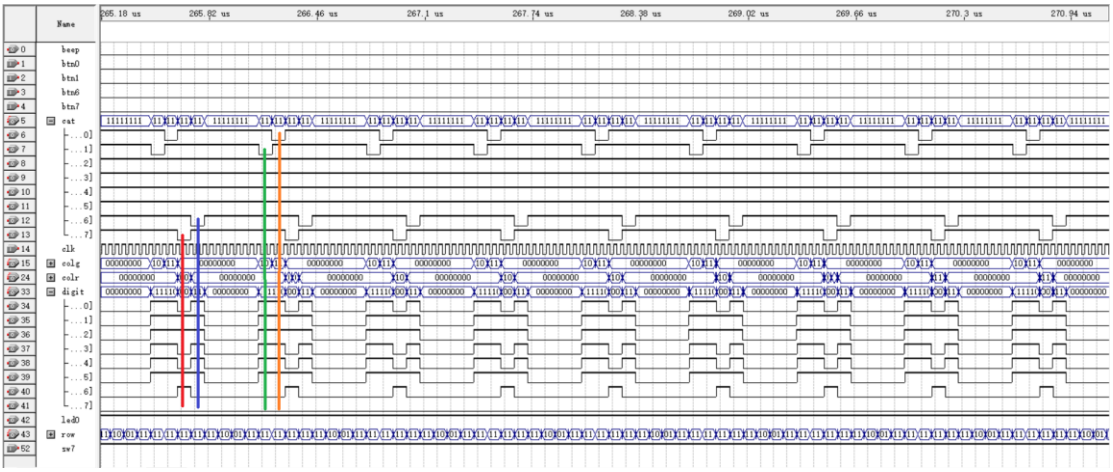


从上图中我们可以看到游戏主界面模块对于射击枪的绘制。从仿真波形图我们可以看到，点阵行扫描信号 row 从 row(7)到 row(0)依次导通，这说明了点阵处于逐行扫描状态。同时我们可以看到，row(1)导通时，绿色点阵列选通信号 colg(3)选通，row(0)导通时，绿色点阵列选通信号 colg(2)、colg(3)、colg(4)选通，这说明点阵在正确的位置绘制了绿色射击枪。

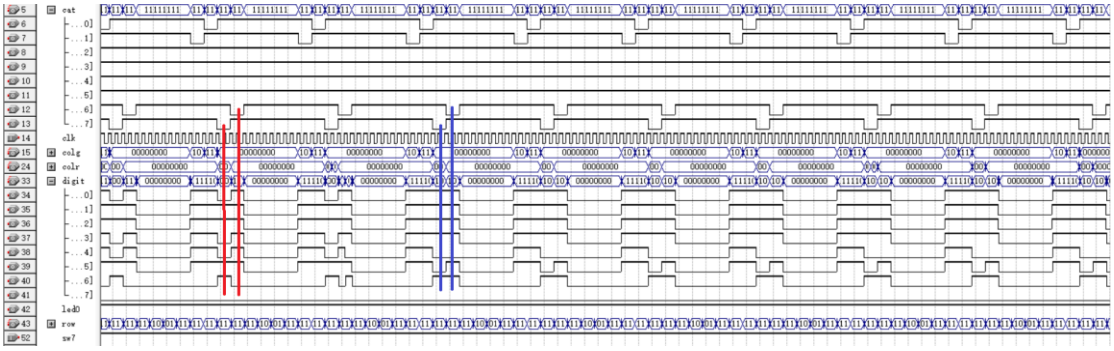
如下图所示，我们可以验证红色移动靶与绿色射击枪的相对位置显示正确。同时我们可以看到移动靶的移动过程。



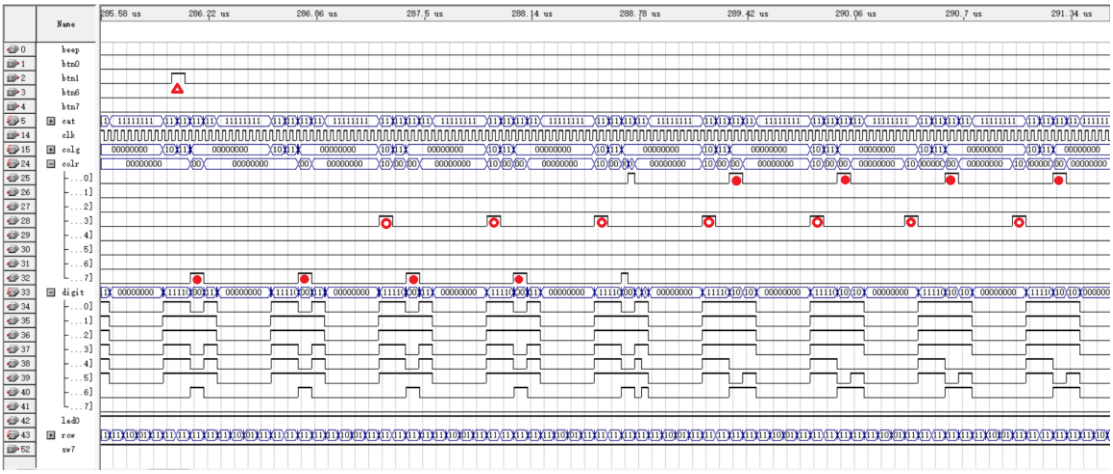
从下图我们可以看出，红色直线所指的时段，cat(7)选通，最左边的数码管点亮，此时 digit 信号为“01100110”，显示为“4”；蓝色直线所指的时段，cat(6)选通，第 2 位数码管点亮，此时 digit 信号为“11111100”，显示为“0”；绿色直线所指的时段，cat(1)选通，倒数第 2 位数码管点亮，此时 digit 信号为“11111100”，显示为“0”；橙色直线所指的时段，cat(0)选通，最右边的数码管点亮，此时 digit 信号为“11111100”，显示为“0”。由此可以看出，此时倒计时信息为“40”，计分信息为“00”。



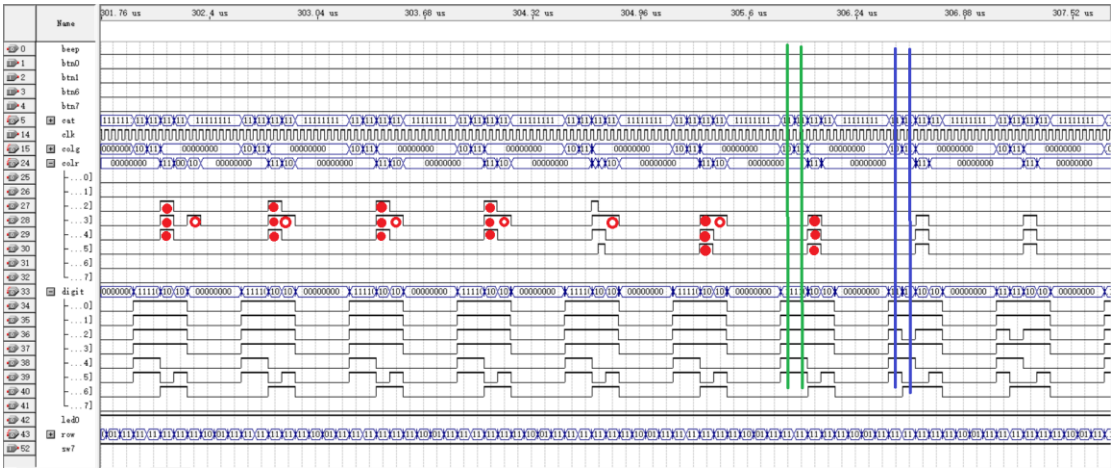
由下图可以看到，最左端的两个数码管的显示信息由“40”转变为“39”，这体现了游戏的倒计时功能。



如下图所示，在红色三角位置，BTN1 按键按下，表示“射击”，随后，红色空心圆圈处表示子弹的出现。从下图中，我们还可以看到红色实心圆圈表示的移动靶在最右端消失，又在最左边出现，这就是移动靶的滚动显示效果。

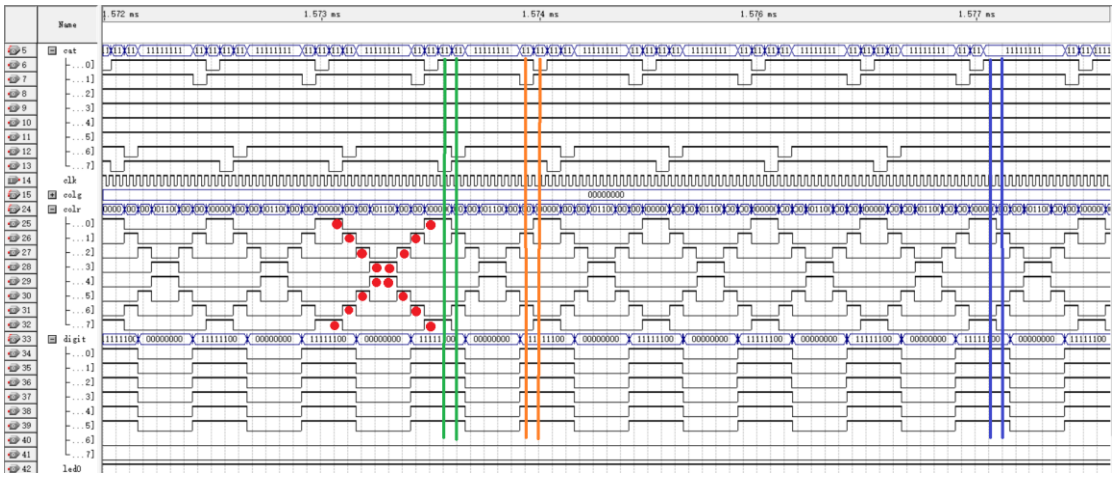


如下图所示，可以看到红色空心圆圈表示的子弹在飞行并逐渐命中移动靶的过程。绿色直线所示的位置表示数码管的计分显示“00”，而蓝色直线所示的位置表示数码管的计分显示“02”，这说明子弹命中了移动靶的边缘，游戏主界面模块成功地完成了计分功能。

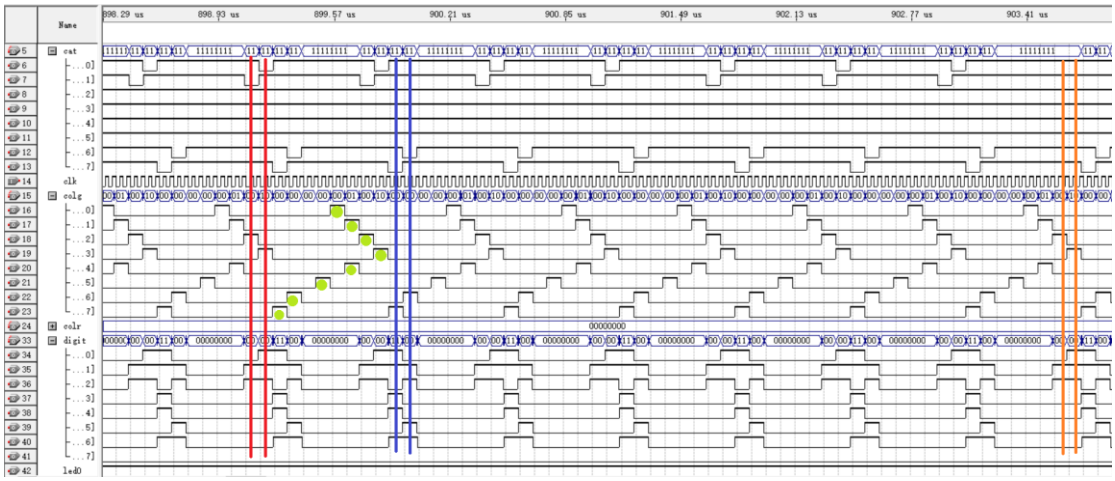


6.结果显示模块的仿真测试：

如下图所示，游戏失败时，点阵显示红色“叉”图案。由橙色直线所示的位置可以看出，当 cat(1)、cat(0)选通时，也就是最右边的两个数码管依次选通的时候，数码管显示的 digit 信号依次为“0”、“0”，即此次游戏的最终游戏得分显示“00”。由绿色直线所示的位置可以看出，当 cat(7)、cat(6)依次选通时，也就是最左边的两个数码管依次选通的时候，数码管显示的 digit 信号依次为“0”、“0”，即当前剩余时间显示为“00”。由蓝色直线所示的位置可以看出，此时 digit 信号依次为“0”、“0”，这是游戏的剩余时间“00”，然而此时 cat(7)、cat(6)并不选通，这说明 cat(7)、cat(6)在有的周期内选通，在有的周期内不选通，也就表明在游戏失败时，数码管最左边两位数字为闪烁显示。



如图所示，游戏胜利时，点阵显示绿色“对勾”图案。由红色直线所示的位置可以看出，当 cat(1)、cat(0)选通时，也就是最右边的两个数码管依次选通的时候，数码管显示的 digit 信号依次为“1”、“7”，即此次游戏的最终游戏得分显示“17”。由蓝色直线所示的位置可以看出，当 cat(7)、cat(6)依次选通时，也就是最左边的两个数码管依次选通的时候，数码管显示的 digit 信号依次为“2”、“4”，即当前剩余时间显示为“24”。由橙色直线所示的位置可以看出，此时 digit 信号依次为“1”、“7”，这是游戏的得分信息“17”，然而此时 cat(1)、cat(0)并不选通，这说明 cat(1)、cat(0)在有的周期内选通，在有的周期内不选通，也就表明在游戏胜利时，数码管最右边两位数字为闪烁显示。



四、代码原理：

```
--顶层设计文件
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--总体电路的对外接口
ENTITY shootingGame IS
    PORT(
        clk:IN STD_LOGIC;--系统 50MHz 时钟
        sw7:IN STD_LOGIC;--拨码开关 SW7
        btn0:IN STD_LOGIC;--按键 BTN0，控制游戏开始
        btn1:IN STD_LOGIC;--按键 BTN1，控制射击
        btn6:IN STD_LOGIC;--按键 BTN6，控制靶的移动速度降低
        btn7:IN STD_LOGIC;--按键 BTN7，控制靶的移动速度提高
        led0:OUT STD_LOGIC;--LED0，开关指示灯
        cat:OUT STD_LOGIC_VECTOR(0 TO 7);--数码管的选通信号
        digit:OUT STD_LOGIC_VECTOR(0 TO 7);--七段数码管 AA~AG 以及 AP 的电平信号
        row:OUT STD_LOGIC_VECTOR(0 TO 7);--点阵行选通信号
        colr:OUT STD_LOGIC_VECTOR(0 TO 7);--红色点阵列选通信号
        colg:OUT STD_LOGIC_VECTOR(0 TO 7);--绿色点阵列选通信号
        beep:OUT STD_LOGIC--蜂鸣器信号
    );
END ENTITY;

--各模块的接口以及接口之间的连线关系
ARCHITECTURE gameDesign OF shootingGame IS

    SIGNAL SIG_BTN_RESUME:STD_LOGIC;
    SIGNAL SIG_BTN_SHOOT:STD_LOGIC;
    SIGNAL SIG_BTN_SPEED_DOWN:STD_LOGIC;
    SIGNAL SIG_BTN_SPEED_UP:STD_LOGIC;
    SIGNAL SIG_CMP_CHK:STD_LOGIC;
    SIGNAL SIG_CMP_WT:STD_LOGIC;
    SIGNAL SIG_CMP_INIT:STD_LOGIC;
    SIGNAL SIG_CMP_GAME:STD_LOGIC;
    SIGNAL SIG_FLG_CHK:STD_LOGIC;
    SIGNAL SIG_FLG_WT:STD_LOGIC;
    SIGNAL SIG_FLG_INIT:STD_LOGIC;
    SIGNAL SIG_FLG_GAME:STD_LOGIC;
```

```
    SIGNAL SIG_FLG_RESULT:STD_LOGIC;
    SIGNAL SIG_FLG_PWR:STD_LOGIC;
    SIGNAL SIG_FLG_RESUME:STD_LOGIC;
    SIGNAL SIG_FLG_WIN:STD_LOGIC;
    SIGNAL SIG_FLG_BULLET:STD_LOGIC;
    SIGNAL SIG_STATE_CHK:INTEGER RANGE 0 TO 7;
    SIGNAL SIG_STATE_CHK_SCAN:INTEGER RANGE 0 TO 2;
    SIGNAL SIG_STATE_INIT:INTEGER RANGE 0 TO 7;
    SIGNAL SIG_STATE_GAME_TARGET:INTEGER RANGE 0 TO 9;
    SIGNAL SIG_STATE_GAME_BULLET:INTEGER RANGE 0 TO 6;
    SIGNAL SIG_STATE_GAME_TIME_LEFT:INTEGER RANGE 0 TO 40;
    SIGNAL SIG_STATE_GAME_SCORE:INTEGER RANGE 0 TO 19;
    SIGNAL SIG_RESULT_TIME_LEFT:INTEGER RANGE 0 TO 40;
    SIGNAL SIG_RESULT_SCORE:INTEGER RANGE 0 TO 19;
    SIGNAL SIG_FLG_BULLET_MISS:STD_LOGIC;
    SIGNAL SIG_FLG_BULLET_GET:STD_LOGIC;

    --控制器的接口设计
    COMPONENT systemController
        PORT(
            CLK_SYS:IN STD_LOGIC;--系统 50MHz 时钟信号
            SW_PWR:IN STD_LOGIC;--拨码开关的状态
            BTN_RESUME:IN STD_LOGIC;--按键 BTN0 用于重新开始游戏时，该信号置 1，
否则置 0
            CMP_CHK:IN STD_LOGIC;--自检过程完成标记信号，自检过程完成后，由自检
模块反馈给控制器
            CMP_WT:IN STD_LOGIC;--待机过程完成标记信号，待机结束后，由待机模块
反馈给控制器
            CMP_INIT:IN STD_LOGIC;--渐亮显示过程完成标记信号，渐亮显示结束后，由
渐亮显示模块反馈给控制器
            CMP_GAME:IN STD_LOGIC;--游戏过程完成标记信号，游戏结束后，由游戏主
界面模块反馈给控制器
            FLG_CHK:OUT STD_LOGIC;--自检模块正在执行的标记信号，此信号使能自检
模块
            FLG_WT:OUT STD_LOGIC;--待机模块正在执行的标记信号，此信号使能待机模
块
            FLG_INIT:OUT STD_LOGIC;--渐亮显示模块正在执行的标记信号，此信号使能
渐亮显示模块
            FLG_GAME:OUT STD_LOGIC;--游戏主界面正在执行的标记信号，此信号使能
游戏主界面模块
            FLG_RESULT:OUT STD_LOGIC;--结果显示过程正在执行的标记信号，此信号使
能结果显示模块
            FLG_PWR:OUT STD_LOGIC;--拨码开关状态标记，由控制器通过此信号控制各
子模块的重置
```

```
            FLG_RESUME:OUT STD_LOGIC--按键 BTN0 用于重新开始游戏时，控制器运用此标志信号控制渐亮显示、游戏主界面与结果显示模块的重置
        );
    END COMPONENT;

    --自检模块的接口设计
    COMPONENT autoCheck
        PORT(
            CLK_CHK:IN STD_LOGIC;--系统时钟信号
            EN_CHK:IN STD_LOGIC;--自检使能信号，由 FLG_CHK 驱动
            RST_CHK:IN STD_LOGIC;--自检模块重置信号，由 FLG_PWR 驱动
            CMP_CHK:OUT STD_LOGIC;--自检过程完成标记信号，用于反馈给控制器
            STATE_CHK:OUT INTEGER RANGE 0 TO 7;--当前扫描到第几行以及第几个数码管，用于控制显示模块以及音频播放模块
            STATE_CHK_SCAN:OUT INTEGER RANGE 0 TO 2--当前完成的扫描次数计数，用于判断自检是否完成
        );
    END COMPONENT;

    --待机模块的接口设计
    COMPONENT waitGame
        PORT(
            CLK_WT:IN STD_LOGIC;--系统时钟信号
            EN_WT:IN STD_LOGIC;--待机使能信号，由 FLG_WT 驱动
            RST_WT:IN STD_LOGIC;--待机模块重置信号，由 FLG_PWR 驱动
            BTN_START:IN STD_LOGIC;--接收 BTN0 按下的信号，判断待机过程是否完成
            CMP_WT:OUT STD_LOGIC--待机过程完成时，将此信号置 1，反馈至控制器
        );
    END COMPONENT;

    --渐亮显示模块的接口设计
    COMPONENT initGun
        PORT(
            CLK_INIT:IN STD_LOGIC;--系统时钟信号
            EN_INIT:IN STD_LOGIC;--渐亮显示模块使能信号，由 FLG_INIT 驱动
            RST_INIT:IN STD_LOGIC;--渐亮显示模块重置信号，由 FLG_PWR 驱动
            RESUME_INIT:IN STD_LOGIC;--BTN0 用于重新开始游戏时，重置渐亮显示模块
            CMP_INIT:OUT STD_LOGIC;--渐亮显示过程完成后，将此信号置 1 反馈至控制器
            STATE_INIT:OUT INTEGER RANGE 0 TO 7--在渐亮显示的不同阶段，此参数由 0 至 7 变化，由此控制点阵选通信号的占空比
        );
    END COMPONENT;
```

```
    --游戏主界面模块的接口设计
    COMPONENT mainGame
        PORT(
            CLK_GAME:IN STD_LOGIC;--系统时钟信号
            EN_GAME:IN STD_LOGIC;--游戏主界面模块使能信号，由 FLG_GAME 驱动
            RST_GAME:IN STD_LOGIC;--游戏主界面模块重置信号，由 FLG_PWR 驱动
            RESUME_GAME:IN STD_LOGIC;--BTN0 用于重新开始游戏时，重置游戏主界面模块
            BTN_SHOOT:IN STD_LOGIC;--接收 BTN1 按下的射击信号
            BTN_SPEED_DOWN:IN STD_LOGIC;--接收 BTN6 按下的靶速减慢信号
            BTN_SPEED_UP:IN STD_LOGIC;--接收 BTN7 按下的靶速增加信号
            CMP_GAME:OUT STD_LOGIC;--游戏结束时，将此信号置 1，反馈至控制器
            STATE_GAME_TARGET:OUT INTEGER RANGE 0 TO 9;--此参数用于记录移动靶的位置信息，显示模块根据此参数绘制点阵
            STATE_GAME_BULLET:OUT INTEGER RANGE 0 TO 6;--此参数用于记录子弹的位置信息，显示模块根据此参数绘制点阵
            STATE_GAME_TIME_LEFT:OUT INTEGER RANGE 0 TO 40;--此参数用于倒计时，显示模块根据此参数设置数码管
            STATE_GAME_SCORE:OUT INTEGER RANGE 0 TO 19;--此参数用于计分，显示模块根据此参数设置数码管
            STATE_GAME_SHOOT:OUT STD_LOGIC;--此参数用于表示子弹是否处于飞行状态，置 1 时，表示子弹正在飞行，此时显示模块绘制子弹轨迹，游戏模块不再响应 BTN1 的射击命令；置 0 时，表示没有子弹射出，此时显示模块不绘制子弹，游戏模块能够响应 BTN1 的射击命令
            FLG_BULLET_MISS:OUT STD_LOGIC;--子弹未命中的标记信号，用于控制音效播放模块
            FLG_BULLET_GET:OUT STD_LOGIC--子弹命中的标记信号，用于控制音效播放模块
        );
    END COMPONENT;

    --结果显示模块的接口设计
    COMPONENT result
        PORT(
            CLK_RESULT:IN STD_LOGIC;--系统时钟
            EN_RESULT:IN STD_LOGIC;--结果显示模块的使能信号，由 FLG_RESULT 驱动
            STATE_GAME_TIME_LEFT:IN INTEGER RANGE 0 TO 40;--接收游戏主界面模块的倒计时信息，表示游戏结束时的剩余时间
            STATE_GAME_SCORE:IN INTEGER RANGE 0 TO 19;--接收游戏主界面模块的计分信息，表示游戏结束时的总分
            FLG_WIN:OUT STD_LOGIC;--表示游戏胜负，用于控制点阵的显示
            RESULT_TIME_LEFT:OUT INTEGER RANGE 0 TO 40;--输出游戏剩余时间，控制数码管显示
            RESULT_SCORE:OUT INTEGER RANGE 0 TO 19--输出游戏获得总分，控制数码
```

```
管显示
        );
    END COMPONENT;

    --显示模块的接口设计
    COMPONENT displayer
        PORT(
            CLK_DISP:IN STD_LOGIC;--系统时钟
            FLG_PWR:IN STD_LOGIC;--拨码开关是否打开
            FLG_CHK:IN STD_LOGIC;--自检模块是否正在运行
            FLG_WT:IN STD_LOGIC;--待机模块是否正在运行
            FLG_INIT:IN STD_LOGIC;--渐亮显示模块是否正在运行
            FLG_GAME:IN STD_LOGIC;--游戏主界面模块是否正在运行
            FLG_RESULT:IN STD_LOGIC;--结果显示模块是否正在运行
            FLG_WIN:IN STD_LOGIC;--游戏结果的输赢情况
            --以上信息用于判断游戏处于哪个进程，从而控制显示模块进行相应的显示
            FLG_BULLET:IN STD_LOGIC;--表示子弹是否正在飞行，控制显示模块是否绘制
子弹轨迹
            STATE_CHK:IN INTEGER RANGE 0 TO 7;--表示自检过程中正在扫描的点阵与
数码管的定位信息，用于控制选通相应的点阵行与数码管
            STATE_INIT:IN INTEGER RANGE 0 TO 7;--表示渐亮显示过程中处于何种状态，
用于控制点阵行选通信号的占空比
            STATE_GAME_TARGET:IN INTEGER RANGE 0 TO 9;--通过移动靶位置信息控制
点阵绘制
            STATE_GAME_BULLET:IN INTEGER RANGE 0 TO 6;--通过子弹飞行位置信息控
制点阵绘制
            STATE_GAME_TIME_LEFT:IN INTEGER RANGE 0 TO 40;--接收游戏剩余时间，
控制数码管显示
            STATE_GAME_SCORE:IN INTEGER RANGE 0 TO 19;--接收得分信息，控制数码
管显示
            RESULT_TIME_LEFT:IN INTEGER RANGE 0 TO 40;--接收游戏结束后的游戏剩余
时间，控制数码管是否闪烁显示
            RESULT_SCORE:IN INTEGER RANGE 0 TO 19;--接收游戏结束后的总得分信息，
控制数码管是否闪烁显示
            LED0_OUT:OUT STD_LOGIC;--根据 FLG_PWR 设置此信号，决定开关指示灯是
否点亮
            CAT_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--控制数码管选通
            DIGIT_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--控制数码管显示
            ROW_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--控制点阵行选通
            COLR_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--控制红色点阵列选通
            COLG_OUT:OUT STD_LOGIC_VECTOR(0 TO 7)--控制绿色点阵列选通
        );
    END COMPONENT;
```

```
    --按键消抖模块的接口设计
    COMPONENT buttonProcessor
        PORT(
            CLK_BTNP:IN STD_LOGIC;--系统时钟
            BTN0_IN:IN STD_LOGIC;--接收 BTN0 的按键信号
            BTN1_IN:IN STD_LOGIC;--接收 BTN1 的按键信号
            BTN6_IN:IN STD_LOGIC;--接收 BTN6 的按键信号
            BTN7_IN:IN STD_LOGIC;--接收 BTN7 的按键信号
            BTN_RESUME:OUT STD_LOGIC;--发出消抖后的 BTN0 的开始游戏或者重新开
始游戏的信号
            BTN_SHOOT:OUT STD_LOGIC;--发出消抖后的 BTN1 的射击信号
            BTN_SPEED_DOWN:OUT STD_LOGIC;--发出消抖后的 BTN6 的靶减速信号
            BTN_SPEED_UP:OUT STD_LOGIC--发出消抖后的 BTN7 的靶加速信号
        );
    END COMPONENT;

    --音效播放模块的接口设计
    COMPONENT musicPlayer
        PORT(
            CLK_BEEP:IN STD_LOGIC;--系统时钟
            FLG_PWR:IN STD_LOGIC;--拨码开关是否打开
            FLG_RESUME:IN STD_LOGIC;--是否重新开始游戏
            FLG_CHK:IN STD_LOGIC;--游戏是否处于自检状态
            STATE_CHK:IN INTEGER RANGE 0 TO 7;--自检阶段，根据此信号控制蜂鸣器的
音阶播放
            STATE_CHK_SCAN:IN INTEGER RANGE 0 TO 2;--自检阶段，根据此信号控制蜂
鸣器的音高信息
            FLG_GAME:IN STD_LOGIC;--是否处于游戏主界面的运行状态，控制播放游戏
背景音乐
            FLG_BULLET_MISS:IN STD_LOGIC;--控制播放“未命中”音效
            FLG_BULLET_GET:IN STD_LOGIC;--控制播放“命中”音效
            FLG_RESULT:IN STD_LOGIC;--是否处于结果显示界面
            FLG_WIN:IN STD_LOGIC;--判断游戏结果的输赢，控制播放相应音效
            TONE_OUT:OUT STD_LOGIC--实际输出声音信号
        );
    END COMPONENT;

BEGIN

    --各模块接口间的连线方式
    u1:systemController PORT MAP(
        CLK_SYS=>clk,
        SW_PWR=>sw7,
        BTN_RESUME=>SIG_BTN_RESUME,
```

```
        CMP_CHK=>SIG_CMP_CHK,
        CMP_WT=>SIG_CMP_WT,
        CMP_INIT=>SIG_CMP_INIT,
        CMP_GAME=>SIG_CMP_GAME,
        FLG_CHK=>SIG_FLG_CHK,
        FLG_WT=>SIG_FLG_WT,
        FLG_INIT=>SIG_FLG_INIT,
        FLG_GAME=>SIG_FLG_GAME,
        FLG_RESULT=>SIG_FLG_RESULT,
        FLG_PWR=>SIG_FLG_PWR,
        FLG_RESUME=>SIG_FLG_RESUME
    );

    u2:autoCheck PORT MAP(
        CLK_CHK=>clk,
        EN_CHK=>SIG_FLG_CHK,
        RST_CHK=>SIG_FLG_PWR,
        CMP_CHK=>SIG_CMP_CHK,
        STATE_CHK=>SIG_STATE_CHK,
        STATE_CHK_SCAN=>SIG_STATE_CHK_SCAN
    );

    u3:waitGame PORT MAP(
        CLK_WT=>clk,
        EN_WT=>SIG_FLG_WT,
        RST_WT=>SIG_FLG_PWR,
        BTN_START=>SIG_BTN_RESUME,
        CMP_WT=>SIG_CMP_WT
    );

    u4:initGun PORT MAP(
        CLK_INIT=>clk,
        EN_INIT=>SIG_FLG_INIT,
        RST_INIT=>SIG_FLG_PWR,
        RESUME_INIT=>SIG_FLG_RESUME,
        CMP_INIT=>SIG_CMP_INIT,
        STATE_INIT=>SIG_STATE_INIT
    );

    u5:mainGame PORT MAP(
        CLK_GAME=>clk,
        EN_GAME=>SIG_FLG_GAME,
        RST_GAME=>SIG_FLG_PWR,
        RESUME_GAME=>SIG_FLG_RESUME,
```

```
        BTN_SHOOT=>SIG_BTN_SHOOT,
        BTN_SPEED_DOWN=>SIG_BTN_SPEED_DOWN,
        BTN_SPEED_UP=>SIG_BTN_SPEED_UP,
        CMP_GAME=>SIG_CMP_GAME,
        STATE_GAME_TARGET=>SIG_STATE_GAME_TARGET,
        STATE_GAME_BULLET=>SIG_STATE_GAME_BULLET,
        STATE_GAME_TIME_LEFT=>SIG_STATE_GAME_TIME_LEFT,
        STATE_GAME_SCORE=>SIG_STATE_GAME_SCORE,
        STATE_GAME_SHOOT=>SIG_FLG_BULLET,
        FLG_BULLET_MISS=>SIG_FLG_BULLET_MISS,
        FLG_BULLET_GET=>SIG_FLG_BULLET_GET
    );

    u6:result PORT MAP(
        CLK_RESULT=>clk,
        EN_RESULT=>SIG_FLG_RESULT,
        STATE_GAME_TIME_LEFT=>SIG_STATE_GAME_TIME_LEFT,
        STATE_GAME_SCORE=>SIG_STATE_GAME_SCORE,
        FLG_WIN=>SIG_FLG_WIN,
        RESULT_TIME_LEFT=>SIG_RESULT_TIME_LEFT,
        RESULT_SCORE=>SIG_RESULT_SCORE
    );

    u7:displayer PORT MAP(
        CLK_DISP=>clk,
        FLG_PWR=>SIG_FLG_PWR,
        FLG_CHK=>SIG_FLG_CHK,
        FLG_WT=>SIG_FLG_WT,
        FLG_INIT=>SIG_FLG_INIT,
        FLG_GAME=>SIG_FLG_GAME,
        FLG_RESULT=>SIG_FLG_RESULT,
        FLG_WIN=>SIG_FLG_WIN,
        FLG_BULLET=>SIG_FLG_BULLET,
        STATE_CHK=>SIG_STATE_CHK,
        STATE_INIT=>SIG_STATE_INIT,
        STATE_GAME_TARGET=>SIG_STATE_GAME_TARGET,
        STATE_GAME_BULLET=>SIG_STATE_GAME_BULLET,
        STATE_GAME_TIME_LEFT=>SIG_STATE_GAME_TIME_LEFT,
        STATE_GAME_SCORE=>SIG_STATE_GAME_SCORE,
        RESULT_TIME_LEFT=>SIG_RESULT_TIME_LEFT,
        RESULT_SCORE=>SIG_RESULT_SCORE,
        LED0_OUT=>led0,
        CAT_OUT=>cat,
        DIGIT_OUT=>digit,
```

```
        ROW_OUT=>row,
        COLR_OUT=>colr,
        COLG_OUT=>colg
    );

    u8:buttonProcessor PORT MAP(
        CLK_BTNP=>clk,
        BTN0_IN=>btn0,
        BTN1_IN=>btn1,
        BTN6_IN=>btn6,
        BTN7_IN=>btn7,
        BTN_RESUME=>SIG_BTN_RESUME,
        BTN_SHOOT=>SIG_BTN_SHOOT,
        BTN_SPEED_DOWN=>SIG_BTN_SPEED_DOWN,
        BTN_SPEED_UP=>SIG_BTN_SPEED_UP
    );

    u9:musicPlayer PORT MAP(
        CLK_BEEP=>clk,
        FLG_PWR=>SIG_FLG_PWR,
        FLG_RESUME=>SIG_FLG_RESUME,
        FLG_CHK=>SIG_FLG_CHK,
        STATE_CHK=>SIG_STATE_CHK,
        STATE_CHK_SCAN=>SIG_STATE_CHK_SCAN,
        FLG_GAME=>SIG_FLG_GAME,
        FLG_BULLET_MISS=>SIG_FLG_BULLET_MISS,
        FLG_BULLET_GET=>SIG_FLG_BULLET_GET,
        FLG_RESULT=>SIG_FLG_RESULT,
        FLG_WIN=>SIG_FLG_WIN,
        TONE_OUT=>beep
    );

END gameDesign;



--控制器的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--控制器的接口设计
ENTITY systemController IS
```

```
    PORT(
        CLK_SYS:IN STD_LOGIC;--系统时钟
        SW_PWR:IN STD_LOGIC;--拨码开关
        BTN_RESUME:IN STD_LOGIC;--BTN0 是否按下

        --接收各模块过程是否完成的反馈信息，判断当前应当使能哪个模块
        CMP_CHK:IN STD_LOGIC;
        CMP_WT:IN STD_LOGIC;
        CMP_INIT:IN STD_LOGIC;
        CMP_GAME:IN STD_LOGIC;

        --各模块正在运行的标志信号，用于使能各模块
        FLG_CHK:OUT STD_LOGIC:='0';
        FLG_WT:OUT STD_LOGIC:='0';
        FLG_INIT:OUT STD_LOGIC:='0';
        FLG_GAME:OUT STD_LOGIC:='0';
        FLG_RESULT:OUT STD_LOGIC:='0';

        --将拨码开关的状态与 BTN0 的按键状态输出，用于控制各模块是否重置
        FLG_PWR:OUT STD_LOGIC:='0';
        FLG_RESUME:OUT STD_LOGIC:='0'
    );
END ENTITY;

ARCHITECTURE systemControl OF systemController IS
BEGIN

    PROCESS(CLK_SYS)--响应拨码开关的状态与 BTN0 的状态，用于重置游戏进程模块；通过各游戏进程模块反馈的 CMP 信号，判断当前游戏进程，控制各游戏进程模块按照顺序使能与失能
    BEGIN
        IF SW_PWR='0' THEN--拨码开关关闭时
            FLG_CHK<='0';
            FLG_WT<='0';
            FLG_INIT<='0';
            FLG_GAME<='0';
            FLG_RESULT<='0';--令所有模块失能
            FLG_PWR<='0';--令所有模块重置到初始状态
            FLG_RESUME<='0';
        ELSIF SW_PWR='1' AND CMP_WT='1' AND BTN_RESUME='1' THEN
        --如果拨码开关打开，且游戏处于渐亮显示、游戏主界面、结果显示三个进程之一时，由控制器响应 BTN0 的按键信息
            FLG_GAME<='0';
            FLG_RESULT<='0';--按下 BTN0 时，令游戏主界面、结果显示两个模块失能
```

```
            FLG_PWR<='1';
            FLG_RESUME<='1';--同时令渐亮显示、游戏主界面、结果显示三个模块重置到
初始状态
        ELSE--如果拨码开关打开，且没有 BTN0 用于发出“重新开始游戏”的命令
            --根据各模块反馈的信息判断游戏进程
            IF CMP_CHK='0' THEN--自检未完成时，自检模块使能，其它模块失能
                FLG_CHK<='1';
                FLG_WT<='0';
                FLG_INIT<='0';
                FLG_GAME<='0';
                FLG_RESULT<='0';
                FLG_PWR<='1';
                FLG_RESUME<='0';
            ELSIF CMP_WT='0' THEN--自检完成而待机未完成时，待机模块使能，其余模
块失能
                FLG_CHK<='0';
                FLG_WT<='1';
                FLG_INIT<='0';
                FLG_GAME<='0';
                FLG_RESULT<='0';
                FLG_PWR<='1';
                FLG_RESUME<='0';
            ELSIF CMP_INIT='0' THEN--待机完成而渐亮显示未完成时，渐亮显示模块使
能，其余模块失能
                FLG_CHK<='0';
                FLG_WT<='0';
                FLG_INIT<='1';
                FLG_GAME<='0';
                FLG_RESULT<='0';
                FLG_PWR<='1';
                FLG_RESUME<='0';
            ELSIF CMP_GAME='0' THEN--渐亮显示完成而游戏主界面未完成时，游戏主界
面模块使能，其余模块失能
                FLG_CHK<='0';
                FLG_WT<='0';
                FLG_INIT<='0';
                FLG_GAME<='1';
                FLG_RESULT<='0';
                FLG_PWR<='1';
                FLG_RESUME<='0';
            ELSE--否则，结果显示模块使能，其余模块失能
                FLG_CHK<='0';
                FLG_WT<='0';
                FLG_INIT<='0';
```

```
                    FLG_GAME<='0';
                    FLG_RESULT<='1';
                    FLG_PWR<='1';
                    FLG_RESUME<='0';
                END IF;
            END IF;
        END PROCESS;

END systemControl;



--自检模块的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--自检模块的接口设计
ENTITY autoCheck IS
    PORT(
        CLK_CHK:IN STD_LOGIC;--系统时钟
        EN_CHK:IN STD_LOGIC;--使能信号
        RST_CHK:IN STD_LOGIC;--重置信号
        CMP_CHK:OUT STD_LOGIC:='0';--将自检是否完成的信息反馈给控制器
        STATE_CHK:OUT INTEGER RANGE 0 TO 7:=0;--控制点阵与数码管的扫描
        STATE_CHK_SCAN:OUT INTEGER RANGE 0 TO 2:=0--扫描计数器，用于判断自检是
否完成
    );
END ENTITY;

ARCHITECTURE check OF autoCheck IS

    SIGNAL clkCnt:INTEGER RANGE 0 TO 9999999;--5Hz 分频计数器
    SIGNAL rowCnt:INTEGER RANGE 0 TO 7;--控制点阵行扫描与数码管扫描
    SIGNAL scanCnt:INTEGER RANGE 0 TO 3;--扫描次数的计数器

BEGIN

    PROCESS(CLK_CHK)
    BEGIN
        IF CLK_CHK'EVENT AND CLK_CHK='1' THEN
            IF RST_CHK='0' THEN--如果拨码开关关闭，将自检模块重置
                CMP_CHK<='0';--将自检完成标志信息清零，允许程序从头开始执行
```

```
            ELSIF RST_CHK='1' AND scanCnt=3 THEN--如果拨码开关打开，且扫描次数达到 3 次，说明自检完成
                CMP_CHK<='1';
            END IF;
            IF EN_CHK='0' THEN--自检模块失能后，将内部信号重置为 0
                clkCnt<=0;
                rowCnt<=0;
                scanCnt<=0;
            --综上，自检正常完成时，自检模块失能但不重置，自检模块内部信号重置为 0，而完成标记 CMP_CHK 保持为 1，
            --此时控制器判断令待机、渐亮显示等后面的模块依次使能执行
            --拨码开关关闭，游戏进程被打断时，自检模块失能且重置。自检模块内部信号重置为 0，完成标记 CMP_CHK 也清零
            --再次打开开关时，由于 CMP_CHK 清零，控制器判断令自检模块使能。
            --由于自检模块内部信号已经重置为 0，因而自检模块可以重新开始扫描过程
            ELSE
                IF clkCnt=9999999 THEN--5Hz 分频计数器循环计数
                    clkCnt<=0;
                    IF rowCnt=7 THEN--每隔 0.2 秒，更新 rowCnt，rowCnt 由 0 至 7 循环计数，表示当前扫描点阵哪一行
                        rowCnt<=0;
                        IF scanCnt<3 THEN--rowCnt 每完成一次循环计数，令扫描计数 scanCnt 加一
                            scanCnt<=scanCnt+1;
                        END IF;
                    ELSE
                        rowCnt<=rowCnt+1;
                    END IF;
                ELSE
                    clkCnt<=clkCnt+1;
                END IF;
                STATE_CHK<=rowCnt;--将 rowCnt 与 scanCnt 输出，控制显示模块与音效模块
                IF scanCnt<3 THEN
                    STATE_CHK_SCAN<=scanCnt;
                END IF;
            END IF;
        END IF;
    END PROCESS;

END check;
```

```
--待机模块的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--待机模块的接口设计
ENTITY waitGame IS
    PORT(
        CLK_WT:IN STD_LOGIC;--系统时钟
        EN_WT:IN STD_LOGIC;--使能信号
        RST_WT:IN STD_LOGIC;--重置信号
        BTN_START:IN STD_LOGIC;--响应 BTN0 按下的状态，结束待机状态
        CMP_WT:OUT STD_LOGIC:='0'--待机结束的完成标记
    );
END ENTITY;

ARCHITECTURE waiting OF waitGame IS
BEGIN

    PROCESS(CLK_WT)
    BEGIN
        IF CLK_WT'EVENT AND CLK_WT='1' THEN
            IF RST_WT='0' THEN--拨码开关关闭时，将待机模块的完成标记清零
                CMP_WT<='0';
            ELSIF EN_WT='1' AND BTN_START='1' THEN--拨码开关打开时，响应 BTN0 的
按键信息
                CMP_WT<='1';--BTN0 按下时，将待机结束的完成标记置 1，反馈给控制
器，从而结束待机进程
            END IF;
        END IF;
    END PROCESS;

END waiting;



--渐亮显示模块的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--渐亮显示模块的接口设计
```

```
ENTITY initGun IS
    PORT(
        CLK_INIT:IN STD_LOGIC;--系统时钟
        EN_INIT:IN STD_LOGIC;--使能信号
        RST_INIT:IN STD_LOGIC;--重置信号
        RESUME_INIT:IN STD_LOGIC;--游戏重新开始信号
        CMP_INIT:OUT STD_LOGIC:='0';--渐亮显示模块完成标记
        STATE_INIT:OUT INTEGER RANGE 0 TO 7:=0--渐亮显示模块的当前运行状态，控制点阵选通信号的占空比
    );
END ENTITY;

ARCHITECTURE initializer OF initGun IS

    SIGNAL clkCnt:INTEGER RANGE 0 TO 18749999;
    --将 3 秒时间分成 8 个阶段，每个时段亮度不同，为此设计分频计数器 clkCnt
    SIGNAL stateCnt:INTEGER RANGE 0 TO 7;
    --记录渐亮显示模块的当前状态

BEGIN

    PROCESS(CLK_INIT)
    BEGIN
        IF CLK_INIT'EVENT AND CLK_INIT='1' THEN
            IF RST_INIT='0' THEN
                CMP_INIT<='0';
                clkCnt<=0;
                stateCnt<=0;
                STATE_INIT<=stateCnt;
            ELSIF RESUME_INIT='1' THEN
                CMP_INIT<='0';
                clkCnt<=0;
                stateCnt<=0;
                STATE_INIT<=stateCnt;
            --拨码开关关闭，或者 BTN0 按下时，重置渐亮显示模块，将内部信号置零，将完成标记 CMP_INIT 清零
            ELSIF EN_INIT='0' THEN
                clkCnt<=0;
                stateCnt<=0;
                STATE_INIT<=stateCnt;
            --在正常游戏进程中，如果渐亮显示模块失能，说明渐亮显示过程未开始或者已经结束，此时将渐亮显示模块的内部信号重置为 0，同时保持完成标记 CMP_INIT 的值，不将其重置
            ELSIF EN_INIT='1' THEN--渐亮显示进程正在执行时
```

```
                IF clkCnt=18749999 THEN--每隔 0.375 秒
                    clkCnt<=0;
                    IF stateCnt<7 THEN--更新渐亮显示模块的内部状态
                        stateCnt<=stateCnt+1;
                    ELSE
                        CMP_INIT<='1';
                    END IF;
                ELSE
                    clkCnt<=clkCnt+1;
                END IF;
                STATE_INIT<=stateCnt;--将内部状态输出，控制点阵扫描占空比
            END IF;
        END IF;
    END PROCESS;

END initializer;



--游戏主界面的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--游戏主界面的接口设计
ENTITY mainGame IS
    PORT(
        CLK_GAME:IN STD_LOGIC;--系统时钟
        EN_GAME:IN STD_LOGIC;--使能信号
        RST_GAME:IN STD_LOGIC;--重置信号
        RESUME_GAME:IN STD_LOGIC;--游戏重新开始
        BTN_SHOOT:IN STD_LOGIC;--射击按键
        BTN_SPEED_DOWN:IN STD_LOGIC;--靶减速按键
        BTN_SPEED_UP:IN STD_LOGIC;--靶加速按键
        CMP_GAME:OUT STD_LOGIC:='0';--游戏主界面进程完成标记
        STATE_GAME_TARGET:OUT INTEGER RANGE 0 TO 9:=0;--靶的位置信息
        STATE_GAME_BULLET:OUT INTEGER RANGE 0 TO 6:=0;--子弹的位置信息
        STATE_GAME_TIME_LEFT:OUT INTEGER RANGE 0 TO 40:=40;--倒计时信息
        STATE_GAME_SCORE:OUT INTEGER RANGE 0 TO 19:=0;--分数信息
        STATE_GAME_SHOOT:OUT STD_LOGIC:='0';--子弹是否正在飞行
        FLG_BULLET_MISS:OUT STD_LOGIC:='0';--未击中
        FLG_BULLET_GET:OUT STD_LOGIC:='0'--击中
    );
```

```
END ENTITY;

ARCHITECTURE game OF mainGame IS

    SIGNAL clkSecondsCnt:INTEGER RANGE 0 TO 49999999:=0;--倒计时分频计数器
    SIGNAL clkTargetCnt:INTEGER RANGE 0 TO 49999999:=0;--控制靶移动的分频计数器
    SIGNAL clkBulletCnt:INTEGER RANGE 0 TO 4999999:=0;--控制子弹飞行的 0.1 秒分频计数器
    SIGNAL clkButtonCheckCnt:INTEGER RANGE 0 TO 999999:=0;--控制每隔 20 毫秒进行一次按键检测
    SIGNAL targetSpeed:INTEGER RANGE 0 TO 49999999:=4999999;--控制靶移动速度的变量，决定靶移动分频器的分频数值
    SIGNAL targetPosition:INTEGER RANGE 0 TO 9:=0;--靶的位置信息
    SIGNAL bulletPosition:INTEGER RANGE 0 TO 6:=0;--子弹的位置信息
    SIGNAL timeLeft:INTEGER RANGE 0 TO 40:=40;--倒计时信息
    SIGNAL score:INTEGER RANGE 0 TO 19:=0;--分数信息
    SIGNAL flagBullet:STD_LOGIC:='0';--子弹是否处于飞行状态

BEGIN

    PROCESS(CLK_GAME)
    BEGIN
        IF CLK_GAME'EVENT AND CLK_GAME='1' THEN
            IF RST_GAME='0' THEN
                CMP_GAME<='0';
                clkSecondsCnt<=0;
                clkTargetCnt<=0;
                clkBulletCnt<=0;
                clkButtonCheckCnt<=0;
                targetSpeed<=4999999;
                targetPosition<=0;
                bulletPosition<=0;
                timeLeft<=40;
                score<=0;
                flagBullet<='0';
            ELSIF RESUME_GAME='1' THEN
                CMP_GAME<='0';
                clkSecondsCnt<=0;
                clkTargetCnt<=0;
                clkBulletCnt<=0;
                clkButtonCheckCnt<=0;
                targetSpeed<=4999999;
                targetPosition<=0;
                bulletPosition<=0;
```

```
            timeLeft<=40;
            score<=0;
            flagBullet<='0';
        --拨码开关关闭或者 BTN0 按下时，将内部信号重置，完成标记 CMP_GAME
清零
        ELSIF EN_GAME='0' THEN
            clkSecondsCnt<=0;
            clkTargetCnt<=0;
            clkBulletCnt<=0;
            clkButtonCheckCnt<=0;
            STATE_GAME_TARGET<=targetPosition;
            STATE_GAME_BULLET<=bulletPosition;
            STATE_GAME_TIME_LEFT<=timeLeft;
            STATE_GAME_SCORE<=score;
        --在正常游戏进程中，如果游戏主界面模块失能，说明该进程尚未开始或者已
经结束，此时重置内部信号，但不改变完成标记 CMP_GAME
        ELSE
            IF clkSecondsCnt=49999999 THEN--倒计时
                clkSecondsCnt<=0;
                IF timeLeft>0 THEN
                    timeLeft<=timeLeft-1;
                END IF;
            ELSE
                clkSecondsCnt<=clkSecondsCnt+1;
            END IF;
            IF clkTargetCnt=targetSpeed THEN--控制靶的位置移动，由 targetSpeed
决定靶的移动频率，也就是靶的移动速度
                clkTargetCnt<=0;
                IF targetPosition=9 THEN--靶位置的循环滚动
                    targetPosition<=0;
                ELSE
                    targetPosition<=targetPosition+1;
                END IF;
            ELSE
                clkTargetCnt<=clkTargetCnt+1;
            END IF;
            IF flagBullet='1' THEN--当子弹处于飞行状态时
                IF clkBulletCnt=4999999 THEN--每隔 0.1 秒更新一次子弹位置
                    clkBulletCnt<=0;
                    IF bulletPosition<5 THEN
                        bulletPosition<=bulletPosition+1;
                    ELSIF bulletPosition=5 THEN
                        bulletPosition<=6;--子弹到达底线时
                        IF targetPosition=3 OR targetPosition=5 THEN--如果子弹
```

击中靶边缘

```
                    score<=score+2;--加 2 分
                    FLG_BULLET_GET<='1';--击中标记信号置 1
                    FLG_BULLET_MISS<='0';
                ELSIF targetPosition=4 THEN--如果子弹击中靶中央
                    score<=score+3;--加 3 分
                    FLG_BULLET_GET<='1';--击中标记信号置 1
                    FLG_BULLET_MISS<='0';
                ELSE--如果子弹未击中
                    FLG_BULLET_GET<='0';
                    FLG_BULLET_MISS<='1';--未击中标记信号置 1
                END IF;
            ELSE--子弹到达底线后，再经过 0.1 秒，子弹位置清零，设置 flagBullet 为 0，结束对子弹的绘制，重新允许接受按键 BTN1 的信号
                flagBullet<='0';
                bulletPosition<=0;
            END IF;
        ELSE
            clkBulletCnt<=clkBulletCnt+1;
        END IF;
    ELSE--子弹未射出或者子弹飞行结束后，将控制子弹飞行的分频计数器清零，控制“命中”与“未命中”音效的信号清零
        clkBulletCnt<=0;
        FLG_BULLET_GET<='0';
        FLG_BULLET_MISS<='0';
    END IF;
    IF clkButtonCheckCnt=999999 THEN--每隔 20 毫秒检测一次 BTN1、BTN6、BTN7 的按键信息
--一般而言，按键抖动的时长在 20 毫秒以内。通过 buttonProcessor 模块，每次按键的输出信号都被转化为时长 20 毫秒的高电平方波信号。在这里，每隔 20 毫秒检测一次按键信息，恰好保证了每次按键信息都被检测到，且每个按键信息只被检测到一次，这就达到了按键消抖的目的
        clkButtonCheckCnt<=0;
        IF BTN_SPEED_UP='1' THEN--按下 BTN7 时，修改 targetSpeed 使得靶移动频率增高，即靶加速
            IF targetSpeed>100000 THEN
                targetSpeed<=targetSpeed-100000;
            END IF;
        ELSIF BTN_SPEED_DOWN='1' THEN--按下 BTN6 时，靶移动频率降低，即靶减速
            IF targetSpeed<24899999 THEN
                targetSpeed<=targetSpeed+100000;
            END IF;
        ELSIF flagBullet='0' AND BTN_SHOOT='1' THEN--点阵中没有子弹时，
```

```
响应 BTN1 按键，将子弹飞行状态标记为 1，表示子弹正在飞行
                        flagBullet<='1';
                    END IF;
                ELSE
                    clkButtonCheckCnt<=clkButtonCheckCnt+1;
                END IF;
                IF timeLeft=0 OR score>=15 THEN--达到 15 分或者超过时限时，将完成
标志置 1，结束游戏主界面进程
                    CMP_GAME<='1';
                END IF;
                --将内部信号赋值给接口
                STATE_GAME_TARGET<=targetPosition;
                STATE_GAME_BULLET<=bulletPosition;
                STATE_GAME_TIME_LEFT<=timeLeft;
                STATE_GAME_SCORE<=score;
                STATE_GAME_SHOOT<=flagBullet;
            END IF;
        END IF;
    END PROCESS;

END game;



--结果显示模块的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--结果显示模块的接口设计
ENTITY result IS
    PORT(
        CLK_RESULT:IN STD_LOGIC;--系统时钟
        EN_RESULT:IN STD_LOGIC;--使能信号
        STATE_GAME_TIME_LEFT:IN INTEGER RANGE 0 TO 40;--从游戏主界面模块接收倒
计时信息
        STATE_GAME_SCORE:IN INTEGER RANGE 0 TO 19;--从游戏主界面模块接收分数信
息
        FLG_WIN:OUT STD_LOGIC;--判断胜负，控制点阵显示
        RESULT_TIME_LEFT:OUT INTEGER RANGE 0 TO 40;--输出游戏结束时的剩余时间，
控制数码管显示
        RESULT_SCORE:OUT INTEGER RANGE 0 TO 19--输出游戏结束时的得分信息，控制
数码管显示
```

```
    );
END ENTITY;

ARCHITECTURE resultScreen OF result IS

BEGIN

    PROCESS(CLK_RESULT)
    BEGIN
        IF CLK_RESULT'EVENT AND CLK_RESULT='1' THEN
            IF EN_RESULT='1' THEN
                IF STATE_GAME_SCORE>=15 THEN--根据得分信息判断输赢
                    FLG_WIN<='1';
                ELSE
                    FLG_WIN<='0';
                END IF;
                RESULT_TIME_LEFT<=STATE_GAME_TIME_LEFT;--将从游戏主界面模块获
得的分数、时间信息输出
                RESULT_SCORE<=STATE_GAME_SCORE;
            END IF;
        END IF;
    END PROCESS;

END resultScreen;



--按键信息处理模块的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--按键信息处理模块的接口设计
ENTITY buttonProcessor IS
    PORT(
        CLK_BTNP:IN STD_LOGIC;--系统时钟
        BTN0_IN:IN STD_LOGIC;--BTN0 的按键信息
        BTN1_IN:IN STD_LOGIC;--BTN1 的按键信息
        BTN6_IN:IN STD_LOGIC;--BTN6 的按键信息
        BTN7_IN:IN STD_LOGIC;--BTN7 的按键信息
        BTN_RESUME:OUT STD_LOGIC;--处理后的 BTN0 的按键信息
        BTN_SHOOT:OUT STD_LOGIC;--处理后的 BTN1 的按键信息
        BTN_SPEED_DOWN:OUT STD_LOGIC;--处理后的 BTN6 的按键信息
```

```
        BTN_SPEED_UP:OUT STD_LOGIC--处理后的 BTN7 的按键信息
    );
END ENTITY;

ARCHITECTURE buttonProcess OF buttonProcessor IS
    SIGNAL clkCnt:INTEGER RANGE 0 TO 999999:=0;
BEGIN

    PROCESS(CLK_BTNP)
    BEGIN
        IF CLK_BTNP'EVENT AND CLK_BTNP='1' THEN
            IF clkCnt=999999 THEN--每隔 20 毫秒，检测一次按键信息，这样，每次按键
信息的输出都是时长为 20 毫秒的高电平方波信号
                clkCnt<=0;
                BTN_RESUME<=BTN0_IN;
                BTN_SHOOT<=BTN1_IN;
                BTN_SPEED_DOWN<=BTN6_IN;
                BTN_SPEED_UP<=BTN7_IN;
            ELSE
                clkCnt<=clkCnt+1;
            END IF;
        END IF;
    END PROCESS;

END buttonProcess;



--显示模块的程序设计
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--显示模块的接口设计
ENTITY displayer IS
    PORT(
        CLK_DISP:IN STD_LOGIC;--系统时钟
        FLG_PWR:IN STD_LOGIC;--拨码开关状态
        FLG_CHK:IN STD_LOGIC;
        FLG_WT:IN STD_LOGIC;
        FLG_INIT:IN STD_LOGIC;
        FLG_GAME:IN STD_LOGIC;
        FLG_RESULT:IN STD_LOGIC;
```

```
        --哪个模块处于运行状态
        FLG_WIN:IN STD_LOGIC;--结果显示的输赢情况
        FLG_BULLET:IN STD_LOGIC;--子弹是否处于飞行状态
        STATE_CHK:IN INTEGER RANGE 0 TO 7;--自检行扫描信息
        STATE_INIT:IN INTEGER RANGE 0 TO 7;--渐亮显示状态信息
        STATE_GAME_TARGET:IN INTEGER RANGE 0 TO 9; --移动靶位置信息
        STATE_GAME_BULLET:IN INTEGER RANGE 0 TO 6;--子弹位置信息
        STATE_GAME_TIME_LEFT:IN INTEGER RANGE 0 TO 40;--游戏过程中的倒计时信息
        STATE_GAME_SCORE:IN INTEGER RANGE 0 TO 19;--游戏过程中的计分信息
        RESULT_TIME_LEFT:IN INTEGER RANGE 0 TO 40;--结果显示的剩余时间信息
        RESULT_SCORE:IN INTEGER RANGE 0 TO 19;--结果显示的总分信息
        LED0_OUT:OUT STD_LOGIC;--电源指示灯
        CAT_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--数码管选通信号
        DIGIT_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--数码管显示信号
        ROW_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--点阵行选通信号
        COLR_OUT:OUT STD_LOGIC_VECTOR(0 TO 7);--红色点阵列选通信号
        COLG_OUT:OUT STD_LOGIC_VECTOR(0 TO 7)--绿色点阵列选通信号
    );
END ENTITY;

ARCHITECTURE display OF displayer IS

    SIGNAL clkInitCnt:INTEGER RANGE 0 TO 799;--渐亮显示阶段控制扫描周期的分频计数器
    SIGNAL rowInitCnt:INTEGER RANGE 0 TO 99;--渐亮显示阶段控制行选通的分频计数器
    SIGNAL clkGameCnt:INTEGER RANGE 0 TO 199;--游戏过程中以及结果显示时，控制扫描周期的分频计数器
    SIGNAL rowGameCnt:INTEGER RANGE 0 TO 7;--游戏过程中以及结果显示时，控制选通信号的循环计数器
    SIGNAL digitTime1:INTEGER RANGE 0 TO 4;--倒计时十位数的十进制数值
    SIGNAL digitTime2:INTEGER RANGE 0 TO 9;--倒计时个位数的十进制数值
    SIGNAL digitScore1:INTEGER RANGE 0 TO 1;--分数十位数的十进制数值
    SIGNAL digitScore2:INTEGER RANGE 0 TO 9;--分数个位数的十进制数值
    SIGNAL binaryTime1:STD_LOGIC_VECTOR(0 TO 7);
    SIGNAL binaryTime2:STD_LOGIC_VECTOR(0 TO 7);
    SIGNAL binaryScore1:STD_LOGIC_VECTOR(0 TO 7);
    SIGNAL binaryScore2:STD_LOGIC_VECTOR(0 TO 7);--相应的译码信号
    SIGNAL clkSparkCnt:INTEGER RANGE 0 TO 24999999;--结果显示时，控制数码管每 0.5 秒闪烁一次的分频计数器

BEGIN

    PROCESS(CLK_DISP)
    BEGIN
```

```
        IF CLK_DISP'EVENT AND CLK_DISP='1' THEN
            IF FLG_PWR='0' THEN--拨码开关关闭时，LED0 灭，数码管、点阵全灭
                LED0_OUT<='0';
                DIGIT_OUT<="00000000";
                CAT_OUT<="11111111";
                COLR_OUT<="00000000";
                COLG_OUT<="00000000";
                ROW_OUT<="11111111";
            ELSE
                LED0_OUT<='1';--拨码开关打开时，LED0 点亮
                IF FLG_CHK='1' THEN--如果处于自检状态
                    DIGIT_OUT<="11111111";--数码管显示“8.”
                    COLR_OUT<="11111111";--红色点阵各列全部选通
                    COLG_OUT<="00000000";--绿色点阵各列均不选通
                    CASE STATE_CHK IS--根据自检模块输出的参数由 0 到 7，决定红色点阵选通哪一行，以及数码管选通哪一位
                        WHEN 0=>CAT_OUT<="11111110";ROW_OUT<="11111110";
                        WHEN 1=>CAT_OUT<="11111101";ROW_OUT<="11111101";
                        WHEN 2=>CAT_OUT<="11111011";ROW_OUT<="11111011";
                        WHEN 3=>CAT_OUT<="11110111";ROW_OUT<="11110111";
                        WHEN 4=>CAT_OUT<="11101111";ROW_OUT<="11101111";
                        WHEN 5=>CAT_OUT<="11011111";ROW_OUT<="11011111";
                        WHEN 6=>CAT_OUT<="10111111";ROW_OUT<="10111111";
                        WHEN 7=>CAT_OUT<="01111111";ROW_OUT<="01111111";
                    END CASE;
                ELSIF FLG_INIT='1' THEN--渐亮显示阶段
                    DIGIT_OUT<="00000000";--关闭数码管
                    CAT_OUT<="11111111";
                    IF clkInitCnt=799 THEN
                        clkInitCnt<=0;
                    ELSE
                        clkInitCnt<=clkInitCnt+1;
                    END IF;
                    IF rowInitCnt=99 THEN
                        rowInitCnt<=0;
                    ELSE
                        rowInitCnt<=rowInitCnt+1;
                    END IF;
                    IF clkInitCnt>700-STATE_INIT*100 THEN--由渐亮显示模块的状态参数 STATE_INIT 决定点阵选通信号的占空比，STATE_INIT 由 0 到 7，选通时长逐渐增长，点阵显示逐渐变亮
                        IF rowInitCnt>=50 THEN--交替选通点阵的第一行和第二行，完成射击枪的显示
                            COLG_OUT<="00111000";
```

```
                COLR_OUT<="00000000";
                ROW_OUT<="01111111";
            ELSE
                COLG_OUT<="00010000";
                COLR_OUT<="00000000";
                ROW_OUT<="10111111";
            END IF;
        ELSE
            COLG_OUT<="00000000";
            COLR_OUT<="00000000";
            ROW_OUT<="11111111";
        END IF;
    ELSIF FLG_GAME='1' THEN--游戏主界面进程运行时
        IF clkGameCnt=199 THEN
            clkGameCnt<=0;
            IF rowGameCnt=7 THEN
                rowGameCnt<=0;
            ELSE
                rowGameCnt<=rowGameCnt+1;
            END IF;
        ELSE
            clkGameCnt<=clkGameCnt+1;
        END IF;
        digitTime1<=STATE_GAME_TIME_LEFT/10;--计算倒计时与分数的十
位数字与个位数字
        digitTime2<=STATE_GAME_TIME_LEFT MOD 10;
        digitScore1<=STATE_GAME_SCORE/10;
        digitScore2<=STATE_GAME_SCORE MOD 10;
        CASE digitTime1 IS
            WHEN 4=>binaryTime1<="01100110";--完成译码
            WHEN 3=>binaryTime1<="11110010";
            WHEN 2=>binaryTime1<="11011010";
            WHEN 1=>binaryTime1<="01100000";
            WHEN 0=>binaryTime1<="11111100";
        END CASE;
        CASE digitTime2 IS
            WHEN 9=>binaryTime2<="11110110";
            WHEN 8=>binaryTime2<="11111110";
            WHEN 7=>binaryTime2<="11100000";
            WHEN 6=>binaryTime2<="10111110";
            WHEN 5=>binaryTime2<="10110110";
            WHEN 4=>binaryTime2<="01100110";
            WHEN 3=>binaryTime2<="11110010";
            WHEN 2=>binaryTime2<="11011010";
```

```
            WHEN 1=>binaryTime2<="01100000";
            WHEN 0=>binaryTime2<="11111100";
        END CASE;
        CASE digitScore1 IS
            WHEN 1=>binaryScore1<="01100000";
            WHEN 0=>binaryScore1<="11111100";
        END CASE;
        CASE digitScore2 IS
            WHEN 9=>binaryScore2<="11110110";
            WHEN 8=>binaryScore2<="11111110";
            WHEN 7=>binaryScore2<="11100000";
            WHEN 6=>binaryScore2<="10111110";
            WHEN 5=>binaryScore2<="10110110";
            WHEN 4=>binaryScore2<="01100110";
            WHEN 3=>binaryScore2<="11110010";
            WHEN 2=>binaryScore2<="11011010";
            WHEN 1=>binaryScore2<="01100000";
            WHEN 0=>binaryScore2<="11111100";
        END CASE;
        --rowGameCnt 循环计数，依次扫描选通点阵第一行至第八行
        IF rowGameCnt=0 THEN--绘制点阵第一行，显示数码管第一位
            CAT_OUT<="11111110";
            DIGIT_OUT<=binaryTime1;
            ROW_OUT<="11111110";
            COLG_OUT<="00000000";
            IF STATE_GAME_BULLET=6 THEN--根据子弹位置与靶的位置分
情况讨论
                CASE STATE_GAME_TARGET IS--既需要绘制靶也需要绘制
子弹的情况
                    WHEN 0=>COLR_OUT<="10010000";
                    WHEN 1=>COLR_OUT<="11010000";
                    WHEN 2=>COLR_OUT<="11110000";
                    WHEN 3=>COLR_OUT<="01110000";
                    WHEN 4=>COLR_OUT<="00111000";
                    WHEN 5=>COLR_OUT<="00011100";
                    WHEN 6=>COLR_OUT<="00011110";
                    WHEN 7=>COLR_OUT<="00010111";
                    WHEN 8=>COLR_OUT<="00010011";
                    WHEN 9=>COLR_OUT<="00010001";
                END CASE;
            ELSE
                CASE STATE_GAME_TARGET IS--只需要绘制靶的情况
                    WHEN 0=>COLR_OUT<="10000000";
                    WHEN 1=>COLR_OUT<="11000000";
```

```
                WHEN 2=>COLR_OUT<="11100000";
                WHEN 3=>COLR_OUT<="01110000";
                WHEN 4=>COLR_OUT<="00111000";
                WHEN 5=>COLR_OUT<="00011100";
                WHEN 6=>COLR_OUT<="00001110";
                WHEN 7=>COLR_OUT<="00000111";
                WHEN 8=>COLR_OUT<="00000011";
                WHEN 9=>COLR_OUT<="00000001";
            END CASE;
        END IF;
    ELSIF rowGameCnt=1 THEN--点阵第二行，数码管第二位
        CAT_OUT<="11111101";
        DIGIT_OUT<=binaryTime2;
        ROW_OUT<="11111101";
        COLG_OUT<="00000000";
        IF STATE_GAME_BULLET=5 THEN--根据子弹位置决定是否绘制
            COLR_OUT<="00010000";
        ELSE
            COLR_OUT<="00000000";
        END IF;
    ELSIF rowGameCnt=2 THEN--以下以此类推
        CAT_OUT<="11111111";
        DIGIT_OUT<="00000000";
        ROW_OUT<="11111011";
        COLG_OUT<="00000000";
        IF STATE_GAME_BULLET=4 THEN
            COLR_OUT<="00010000";
        ELSE
            COLR_OUT<="00000000";
        END IF;
    ELSIF rowGameCnt=3 THEN
        CAT_OUT<="11111111";
        DIGIT_OUT<="00000000";
        ROW_OUT<="11110111";
        COLG_OUT<="00000000";
        IF STATE_GAME_BULLET=3 THEN
            COLR_OUT<="00010000";
        ELSE
            COLR_OUT<="00000000";
        END IF;
    ELSIF rowGameCnt=4 THEN
        CAT_OUT<="11111111";
        DIGIT_OUT<="00000000";
        ROW_OUT<="11101111";
```

```
                COLG_OUT<="00000000";
                IF STATE_GAME_BULLET=2 THEN
                    COLR_OUT<="00010000";
                ELSE
                    COLR_OUT<="00000000";
                END IF;
            ELSIF rowGameCnt=5 THEN
                CAT_OUT<="11111111";
                DIGIT_OUT<="00000000";
                ROW_OUT<="11011111";
                COLG_OUT<="00000000";
                IF STATE_GAME_BULLET=1 THEN
                    COLR_OUT<="00010000";
                ELSE
                    COLR_OUT<="00000000";
                END IF;
            ELSIF rowGameCnt=6 THEN
                CAT_OUT<="10111111";
                DIGIT_OUT<=binaryScore1;
                ROW_OUT<="10111111";
                COLG_OUT<="00010000";--点阵第七行，绘制射击枪枪口
                IF STATE_GAME_BULLET=0 AND FLG_BULLET='1' THEN--根据子弹发射状态决定是否绘制子弹
                    COLR_OUT<="00010000";
                ELSE
                    COLR_OUT<="00000000";
                END IF;
            ELSIF rowGameCnt=7 THEN
                CAT_OUT<="01111111";
                DIGIT_OUT<=binaryScore2;
                ROW_OUT<="01111111";
                COLG_OUT<="00111000";--点阵第八行，绘制射击枪枪体
                COLR_OUT<="00000000";
            END IF;
        ELSIF FLG_RESULT='1' THEN--结果显示模块
            IF clkGameCnt=199 THEN--每 200 个系统时钟周期，更新一次数码管及点阵的选通位置。更新频率不宜过高，否则由于时钟沿的不同步，可能会出现显示效果的错误
                clkGameCnt<=0;
                IF rowGameCnt=7 THEN
                    rowGameCnt<=0;
                ELSE
                    rowGameCnt<=rowGameCnt+1;
                END IF;
```

```
ELSE
    clkGameCnt<=clkGameCnt+1;
END IF;
IF clkSparkCnt=24999999 THEN
    clkSparkCnt<=0;
ELSE
    clkSparkCnt<=clkSparkCnt+1;
END IF;
digitTime1<=RESULT_TIME_LEFT/10;--生成时间与分数的个位数字、
十位数字信息
digitTime2<=RESULT_TIME_LEFT MOD 10;
digitScore1<=RESULT_SCORE/10;
digitScore2<=RESULT_SCORE MOD 10;
CASE digitTime1 IS--数字译码
    WHEN 4=>binaryTime1<="01100110";
    WHEN 3=>binaryTime1<="11110010";
    WHEN 2=>binaryTime1<="11011010";
    WHEN 1=>binaryTime1<="01100000";
    WHEN 0=>binaryTime1<="11111100";
END CASE;
CASE digitTime2 IS
    WHEN 9=>binaryTime2<="11110110";
    WHEN 8=>binaryTime2<="11111110";
    WHEN 7=>binaryTime2<="11100000";
    WHEN 6=>binaryTime2<="10111110";
    WHEN 5=>binaryTime2<="10110110";
    WHEN 4=>binaryTime2<="01100110";
    WHEN 3=>binaryTime2<="11110010";
    WHEN 2=>binaryTime2<="11011010";
    WHEN 1=>binaryTime2<="01100000";
    WHEN 0=>binaryTime2<="11111100";
END CASE;
CASE digitScore1 IS
    WHEN 1=>binaryScore1<="01100000";
    WHEN 0=>binaryScore1<="11111100";
END CASE;
CASE digitScore2 IS
    WHEN 9=>binaryScore2<="11110110";
    WHEN 8=>binaryScore2<="11111110";
    WHEN 7=>binaryScore2<="11100000";
    WHEN 6=>binaryScore2<="10111110";
    WHEN 5=>binaryScore2<="10110110";
    WHEN 4=>binaryScore2<="01100110";
    WHEN 3=>binaryScore2<="11110010";
```

```
            WHEN 2=>binaryScore2<="11011010";
            WHEN 1=>binaryScore2<="01100000";
            WHEN 0=>binaryScore2<="11111100";
        END CASE;
        IF rowGameCnt=0 THEN--数码管、点阵逐位、逐行扫描
            DIGIT_OUT<=binaryTime1;
            ROW_OUT<="11111110";
            IF FLG_WIN='1' THEN
                CAT_OUT<="11111110";
                COLR_OUT<="00000000";--胜利时，红色点阵关闭，绿色点
阵绘制对勾
                COLG_OUT<="00000001";
            ELSE
                IF clkSparkCnt<12500000 THEN--失败时，数码管时间信息
闪烁
                    CAT_OUT<="11111110";
                ELSE
                    CAT_OUT<="11111111";
                END IF;
                COLR_OUT<="10000001";
                COLG_OUT<="00000000";
            END IF;
        --以下以此类推
        ELSIF rowGameCnt=1 THEN
            DIGIT_OUT<=binaryTime2;
            ROW_OUT<="11111101";
            IF FLG_WIN='1' THEN
                CAT_OUT<="11111101";
                COLR_OUT<="00000000";
                COLG_OUT<="00000010";
            ELSE
                IF clkSparkCnt<12500000 THEN
                    CAT_OUT<="11111101";
                ELSE
                    CAT_OUT<="11111111";
                END IF;
                COLR_OUT<="01000010";
                COLG_OUT<="00000000";
            END IF;
        ELSIF rowGameCnt=2 THEN
            CAT_OUT<="11111111";
            DIGIT_OUT<="00000000";
            COLG_OUT<="00000000";
            ROW_OUT<="11111011";
```

```
        IF FLG_WIN='1' THEN
            COLR_OUT<="00000000";
        ELSE
            COLR_OUT<="00100100";
        END IF;
    ELSIF rowGameCnt=3 THEN
        CAT_OUT<="11111111";
        DIGIT_OUT<="00000000";
        ROW_OUT<="11110111";
        IF FLG_WIN='1' THEN
            COLR_OUT<="00000000";
            COLG_OUT<="00000100";
        ELSE
            COLR_OUT<="00011000";
            COLG_OUT<="00000000";
        END IF;
    ELSIF rowGameCnt=4 THEN
        CAT_OUT<="11111111";
        DIGIT_OUT<="00000000";
        ROW_OUT<="11101111";
        IF FLG_WIN='1' THEN
            COLR_OUT<="00000000";
            COLG_OUT<="10000000";
        ELSE
            COLR_OUT<="00011000";
            COLG_OUT<="00000000";
        END IF;
    ELSIF rowGameCnt=5 THEN
        CAT_OUT<="11111111";
        DIGIT_OUT<="00000000";
        ROW_OUT<="11011111";
        IF FLG_WIN='1' THEN
            COLR_OUT<="00000000";
            COLG_OUT<="01001000";
        ELSE
            COLR_OUT<="00100100";
            COLG_OUT<="00000000";
        END IF;
    ELSIF rowGameCnt=6 THEN
        DIGIT_OUT<=binaryScore1;
        ROW_OUT<="10111111";
        IF FLG_WIN='1' THEN
            IF clkSparkCnt<12500000 THEN
                CAT_OUT<="10111111";
```

```
                    ELSE
                        CAT_OUT<="11111111";
                    END IF;
                    COLR_OUT<="00000000";
                    COLG_OUT<="00100000";
                ELSE
                    CAT_OUT<="10111111";
                    COLR_OUT<="01000010";
                    COLG_OUT<="00000000";
                END IF;
            ELSIF rowGameCnt=7 THEN
                DIGIT_OUT<=binaryScore2;
                ROW_OUT<="01111111";
                IF FLG_WIN='1' THEN
                    IF clkSparkCnt<12500000 THEN--胜利时，分数闪烁显示
                        CAT_OUT<="01111111";
                    ELSE
                        CAT_OUT<="11111111";
                    END IF;
                    COLR_OUT<="00000000";
                    COLG_OUT<="00010000";
                ELSE
                    CAT_OUT<="01111111";
                    COLR_OUT<="10000001";
                    COLG_OUT<="00000000";
                END IF;
            END IF;
        ELSE
            CAT_OUT<="11111111";
            DIGIT_OUT<="00000000";
            ROW_OUT<="11111111";
            COLG_OUT<="00000000";
            COLR_OUT<="00000000";
        END IF;
    END IF;
END IF;
END PROCESS;

END display;



--音效播放模块的程序设计
LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED;
USE IEEE.STD_LOGIC_ARITH;

--音效播放模块的接口设计
ENTITY musicPlayer IS
    PORT(
        CLK_BEEP:IN STD_LOGIC;--系统时钟
        FLG_PWR:IN STD_LOGIC;--拨码开关状态
        FLG_RESUME:IN STD_LOGIC;--是否重新开始游戏
        FLG_CHK:IN STD_LOGIC;
        STATE_CHK:IN INTEGER RANGE 0 TO 7;
        STATE_CHK_SCAN:IN INTEGER RANGE 0 TO 2;
        FLG_GAME:IN STD_LOGIC;
        FLG_BULLET_MISS:IN STD_LOGIC;
        FLG_BULLET_GET:IN STD_LOGIC;
        FLG_RESULT:IN STD_LOGIC;
        FLG_WIN:IN STD_LOGIC;
        --判断游戏处于哪个阶段，状态参数如何
        TONE_OUT:OUT STD_LOGIC--输出声音信号
    );
END musicPlayer;

ARCHITECTURE player OF musicPlayer IS

    SIGNAL toneFreq:INTEGER RANGE 0 TO 100000;
    SIGNAL toneCnt:INTEGER RANGE 0 TO 100000:=0;
    SIGNAL stateChk:INTEGER RANGE 0 TO 23;--自检音阶状态
    SIGNAL stateMiss:INTEGER RANGE 0 TO 4:=0;--未命中音效状态
    SIGNAL stateGet:INTEGER RANGE 0 TO 4:=0;--命中音效状态
    SIGNAL soundCnt:INTEGER RANGE 0 TO 3124999:=0;--十六分之一音符分频计数器
    SIGNAL musicCnt:INTEGER RANGE 0 TO 12499999:=0;--四分之一音符分频计数器
    SIGNAL gameMusicCnt:INTEGER RANGE 0 TO 23:=0;--游戏音乐状态
    SIGNAL winMusicCnt:INTEGER RANGE 0 TO 3:=0;--胜利音效状态
    SIGNAL loseMusicCnt:INTEGER RANGE 0 TO 3:=0;--失败音效状态
    SIGNAL beep:STD_LOGIC;--声音输出

BEGIN

    PROCESS(CLK_BEEP)
    BEGIN
        IF CLK_BEEP'EVENT AND CLK_BEEP='1' THEN
            IF FLG_PWR='0' OR FLG_RESUME='1' THEN--关闭拨码开关或 BTN0 按下时，
状态清零，声音关闭
```

```
            toneCnt<=0;
            stateMiss<=0;
            stateGet<=0;
            soundCnt<=0;
            musicCnt<=0;
            gameMusicCnt<=0;
            winMusicCnt<=0;
            loseMusicCnt<=0;
            beep<='0';
        ELSIF FLG_CHK='1' THEN--自检时，根据自检模块的输出状态，依次输出低音、
中音、高音音阶
            stateChk<=STATE_CHK_SCAN*8+STATE_CHK;
            CASE stateChk IS
                WHEN 0=>toneFreq<=95566;
                WHEN 1=>toneFreq<=85121;
                WHEN 2=>toneFreq<=75850;
                WHEN 3=>toneFreq<=71592;
                WHEN 4=>toneFreq<=63776;
                WHEN 5=>toneFreq<=56818;
                WHEN 6=>toneFreq<=50618;
                WHEN 7=>toneFreq<=47774;
                WHEN 8=>toneFreq<=47774;
                WHEN 9=>toneFreq<=42568;
                WHEN 10=>toneFreq<=37919;
                WHEN 11=>toneFreq<=35791;
                WHEN 12=>toneFreq<=31888;
                WHEN 13=>toneFreq<=28409;
                WHEN 14=>toneFreq<=25309;
                WHEN 15=>toneFreq<=23912;
                WHEN 16=>toneFreq<=23912;
                WHEN 17=>toneFreq<=21282;
                WHEN 18=>toneFreq<=18961;
                WHEN 19=>toneFreq<=17971;
                WHEN 20=>toneFreq<=15944;
                WHEN 21=>toneFreq<=14205;
                WHEN 22=>toneFreq<=12655;
                WHEN 23=>toneFreq<=11956;
            END CASE;
            IF toneCnt>=toneFreq THEN--设定蜂鸣器电平翻转的频率，即可实现相
应的音高
                toneCnt<=0;
                beep<=NOT beep;
            ELSE
                toneCnt<=toneCnt+1;
```

```
            END IF;
        ELSIF FLG_GAME='1' THEN--在游戏过程中
            IF FLG_BULLET_MISS='0' AND FLG_BULLET_GET='0' AND stateMiss=0
AND stateGet=0 THEN--播放游戏背景音乐
                IF musicCnt=12499999 THEN--每个音符都是四分之一音符
                    musicCnt<=0;
                    IF gameMusicCnt=22 THEN
                        gameMusicCnt<=0;
                    ELSE
                        gameMusicCnt<=gameMusicCnt+1;
                    END IF;
                ELSE
                    musicCnt<=musicCnt+1;
                END IF;
                CASE gameMusicCnt IS
                    WHEN 0=>toneFreq<=47774;
                    WHEN 1=>toneFreq<=63776;
                    WHEN 2=>toneFreq<=47774;
                    WHEN 3=>toneFreq<=37919;
                    WHEN 4=>toneFreq<=47774;
                    WHEN 5=>toneFreq<=37919;
                    WHEN 6=>toneFreq<=31888;
                    WHEN 7=>toneFreq<=23912;
                    WHEN 8=>toneFreq<=25309;
                    WHEN 9=>toneFreq<=28409;
                    WHEN 10=>toneFreq<=31888;
                    WHEN 11=>toneFreq<=31888;
                    WHEN 12=>toneFreq<=35791;
                    WHEN 13=>toneFreq<=37919;
                    WHEN 14=>toneFreq<=47774;
                    WHEN 15=>toneFreq<=63776;
                    WHEN 16=>toneFreq<=56818;
                    WHEN 17=>toneFreq<=47774;
                    WHEN 18=>toneFreq<=35791;
                    WHEN 19=>toneFreq<=37919;
                    WHEN 20=>toneFreq<=47774;
                    WHEN 21=>toneFreq<=28409;
                    WHEN 22=>toneFreq<=31888;
                    WHEN 23=>toneFreq<=0;
                END CASE;
                IF gameMusicCnt<23 THEN
                    IF toneCnt>=toneFreq THEN
                        toneCnt<=0;
                        beep<=NOT beep;
```

```
                ELSE
                    toneCnt<=toneCnt+1;
                END IF;
            ELSE
                beep<='0';
            END IF;
        ELSIF FLG_BULLET_MISS='1' OR stateMiss>0 THEN
        --子弹到达底线且未命中时，游戏主界面模块将 FLG_BULLET_MISS 置 1，此时 stateMiss 的值为 0，未命中音效开始播放；
        --在游戏主界面模块中，0.1 秒后 FLG_BULLET_MISS 被清零，此时未命中音效并未播放完毕，由于 stateMiss 值大于 0，因而未命中音效继续播放；
        --未命中音效播放完毕后，stateMiss 被清零。未命中音效的播放时间为 0.25 秒，小于下一颗子弹从发射到飞行至底线的时间 0.6 秒，因而 stateMiss 被清零时，
        --FLG_BULLET_MISS 与 FLG_BULLET_GET 均为 0，此时未命中音效播放结束，游戏背景音效从断点处开始继续播放
            IF soundCnt=3124999 THEN
                soundCnt<=0;
                IF stateMiss=4 THEN
                    stateMiss<=0;
                ELSE
                    stateMiss<=stateMiss+1;
                END IF;
            ELSE
                soundCnt<=soundCnt+1;
            END IF;
            CASE stateMiss IS
                WHEN 0=>toneFreq<=47774;
                WHEN 1=>toneFreq<=63776;
                WHEN 2=>toneFreq<=75850;
                WHEN 3=>toneFreq<=95566;
                WHEN 4=>toneFreq<=0;
            END CASE;
            IF stateMiss<4 THEN
                IF toneCnt>=toneFreq THEN
                    toneCnt<=0;
                    beep<=NOT beep;
                ELSE
                    toneCnt<=toneCnt+1;
                END IF;
            ELSE
                beep<='0';
            END IF;
        ELSIF FLG_BULLET_GET='1' OR stateGet>0 THEN--播放命中音效，原理与上面相同
```

```
            IF soundCnt=3124999 THEN
                soundCnt<=0;
                IF stateGet=4 THEN
                    stateGet<=0;
                ELSE
                    stateGet<=stateGet+1;
                END IF;
            ELSE
                soundCnt<=soundCnt+1;
            END IF;
            CASE stateGet IS
                WHEN 0=>toneFreq<=95566;
                WHEN 1=>toneFreq<=75850;
                WHEN 2=>toneFreq<=63776;
                WHEN 3=>toneFreq<=47774;
                WHEN 4=>toneFreq<=0;
            END CASE;
            IF stateGet<4 THEN
                IF toneCnt>=toneFreq THEN
                    toneCnt<=0;
                    beep<=NOT beep;
                ELSE
                    toneCnt<=toneCnt+1;
                END IF;
            ELSE
                beep<='0';
            END IF;
        END IF;
    ELSIF FLG_RESULT='1' THEN--结果显示模块运行时
        IF FLG_WIN='1' THEN--如果游戏胜利
            IF musicCnt=12499999 THEN
                musicCnt<=0;
                IF winMusicCnt<3 THEN--播放胜利音效（3 个音符），然后停止
                    winMusicCnt<=winMusicCnt+1;
                END IF;
            ELSE
                musicCnt<=musicCnt+1;
            END IF;
            CASE winMusicCnt IS
                WHEN 0=>toneFreq<=47774;
                WHEN 1=>toneFreq<=31888;
                WHEN 2=>toneFreq<=23912;
                WHEN 3=>toneFreq<=0;
            END CASE;
```

```
            IF winMusicCnt<3 THEN
                IF toneCnt>=toneFreq THEN
                    toneCnt<=0;
                    beep<=NOT beep;
                ELSE
                    toneCnt<=toneCnt+1;
                END IF;
            ELSE
                beep<='0';
            END IF;
        ELSIF FLG_WIN='0' THEN--如果游戏失败
            IF musicCnt=12499999 THEN
                musicCnt<=0;
                IF loseMusicCnt<3 THEN--播放失败音效（3 个音符），然后停止
                    loseMusicCnt<=loseMusicCnt+1;
                END IF;
            ELSE
                musicCnt<=musicCnt+1;
            END IF;
            CASE loseMusicCnt IS
                WHEN 0=>toneFreq<=23912;
                WHEN 1=>toneFreq<=31888;
                WHEN 2=>toneFreq<=47774;
                WHEN 3=>toneFreq<=0;
            END CASE;
            IF loseMusicCnt<3 THEN
                IF toneCnt>=toneFreq THEN
                    toneCnt<=0;
                    beep<=NOT beep;
                ELSE
                    toneCnt<=toneCnt+1;
                END IF;
            ELSE
                beep<='0';
            END IF;
        END IF;
    ELSE--游戏的其余各阶段，状态清零，蜂鸣器不发声
        toneCnt<=0;
        stateMiss<=0;
        stateGet<=0;
        soundCnt<=0;
        musicCnt<=0;
        gameMusicCnt<=0;
        winMusicCnt<=0;
```

```
                loseMusicCnt<=0;
                beep<='0';
            END IF;
            TONE_OUT<=beep;
        END IF;
    END PROCESS;

END player;
```

五、功能说明及资源利用情况：

1.功能说明：

从功能的角度看，本次数字电路实验实现了“移动靶”游戏的开机自检功能、射击枪的渐亮显示效果、移动靶的滚动显示功能、按键防抖功能、移动靶的速度调节功能、射击计分与倒计时功能、游戏结果判断与相应效果显示功能、音乐音效播放功能以及分等级计分功能。

从硬件的角度看，本次数字电路实验实现了数码管、点阵和蜂鸣器的相关功能，使得我们对于相关硬件的原理有了较为深入的认识。

从电路与程序设计的角度看，本次数字电路实验实现了较大规模综合程序的控制模块的设计，完成了控制器与各模块之间控制逻辑与信息反馈的线路与程序设计。

2.资源利用情况：

（1）管脚分配情况：

如图所示，在总共 116 个管脚中，本数字电路设计共利用了 48 个管脚，占管脚总数的 41%。

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	beep	Output	PIN_60	4		3.3-V LVTTL (default)	
2	btn0	Input	PIN_61	4		3.3-V LVTTL (default)	
3	btn1	Input	PIN_20	1		3.3-V LVTTL (default)	
4	btn6	Input	PIN_123	2		3.3-V LVTTL (default)	
5	btn7	Input	PIN_124	2		3.3-V LVTTL (default)	
6	cat[0]	Output	PIN_63	4		3.3-V LVTTL (default)	
7	cat[1]	Output	PIN_66	4		3.3-V LVTTL (default)	
8	cat[2]	Output	PIN_67	4		3.3-V LVTTL (default)	
9	cat[3]	Output	PIN_68	4		3.3-V LVTTL (default)	
10	cat[4]	Output	PIN_69	4		3.3-V LVTTL (default)	
11	cat[5]	Output	PIN_70	4		3.3-V LVTTL (default)	
12	cat[6]	Output	PIN_30	1		3.3-V LVTTL (default)	
13	cat[7]	Output	PIN_31	1		3.3-V LVTTL (default)	
14	clk	Input	PIN_18	1		3.3-V LVTTL (default)	
15	colg[0]	Output	PIN_45	4		3.3-V LVTTL (default)	
16	colg[1]	Output	PIN_44	4		3.3-V LVTTL (default)	
17	colg[2]	Output	PIN_43	4		3.3-V LVTTL (default)	
18	colg[3]	Output	PIN_42	4		3.3-V LVTTL (default)	
19	colg[4]	Output	PIN_41	4		3.3-V LVTTL (default)	
20	colg[5]	Output	PIN_40	4		3.3-V LVTTL (default)	
21	colg[6]	Output	PIN_39	4		3.3-V LVTTL (default)	
22	colg[7]	Output	PIN_38	4		3.3-V LVTTL (default)	
23	colr[0]	Output	PIN_22	1		3.3-V LVTTL (default)	
24	colr[1]	Output	PIN_21	1		3.3-V LVTTL (default)	
25	colr[2]	Output	PIN_16	1		3.3-V LVTTL (default)	
26	colr[3]	Output	PIN_15	1		3.3-V LVTTL (default)	
27	colr[4]	Output	PIN_14	1		3.3-V LVTTL (default)	
28	colr[5]	Output	PIN_13	1		3.3-V LVTTL (default)	
29	colr[6]	Output	PIN_12	1		3.3-V LVTTL (default)	
30	colr[7]	Output	PIN_11	1		3.3-V LVTTL (default)	
31	digit[0]	Output	PIN_62	4		3.3-V LVTTL (default)	
32	digit[1]	Output	PIN_59	4		3.3-V LVTTL (default)	
33	digit[2]	Output	PIN_58	4		3.3-V LVTTL (default)	
34	digit[3]	Output	PIN_57	4		3.3-V LVTTL (default)	
35	digit[4]	Output	PIN_55	4		3.3-V LVTTL (default)	
36	digit[5]	Output	PIN_53	4		3.3-V LVTTL (default)	
37	digit[6]	Output	PIN_52	4		3.3-V LVTTL (default)	
38	digit[7]	Output	PIN_51	4		3.3-V LVTTL (default)	
39	led0	Output	PIN_80	3		3.3-V LVTTL (default)	
40	row[0]	Output	PIN_8	1		3.3-V LVTTL (default)	
41	row[1]	Output	PIN_7	1		3.3-V LVTTL (default)	
42	row[2]	Output	PIN_6	1		3.3-V LVTTL (default)	
43	row[3]	Output	PIN_5	1		3.3-V LVTTL (default)	
44	row[4]	Output	PIN_4	1		3.3-V LVTTL (default)	
45	row[5]	Output	PIN_3	1		3.3-V LVTTL (default)	
46	row[6]	Output	PIN_2	1		3.3-V LVTTL (default)	
47	row[7]	Output	PIN_1	1		3.3-V LVTTL (default)	
48	sw7	Input	PIN_125	2		3.3-V LVTTL (default)	

（2）逻辑单元使用情况：

由 Quartus 在程序编译后出具的报告我们可以看到，本次数字电路设计综合生成的硬件电路一共使用了 1219 个逻辑元件，占总可用逻辑元件数的 96%。

由此可见，本次数字电路设计对于硬件资源的消耗较大。我认为其主要原因有二：一是

控制器的设计有些复杂，尤其是游戏主界面模块和结果显示模块之间的连线有冗余之嫌，应当尝试简化；二是显示模块和音效模块中的分支结构 IF 与 CASE 使用了大量的逻辑元件，应当尝试改进程序设计。

六、故障及问题分析：

在本次数字电路实验中，我遇到了如下故障：

1.数码管显示错误:

我遇到的具体故障是，数码管对于倒计时的十位数字的显示基本正常，但对于倒计时的个位数字的显示不正常。通过仔细观察可以发现，在显示个位数字时，七段数码管各段亮度不均，实际上，本应显示个位数字时，数码管显示的却是个位数字与十位数字相叠加而成的图像，其中重叠部分的亮度更亮，而非重叠部分的亮度较暗。

通过分析我发现，上述故障是由于数码管选通信号 cat 与数码管显示信号 digit 不能严格同步造成的。

如果选通信号 cat 由"11111110"变为"11111101"时，显示信号 digit 恰好从十位数字的译码值转变为个位数字的译码值，则数码管显示不会出现错误。然而事实上，cat 信号与 digit 信号发生变化的时机并不能做到绝对同步。如果 cat 信号先由"11111110"变为"11111101"，随后 digit 才从十位数字的译码值转变为个位数字的译码值，则本该显示个位数字的数码管就有可能出现显示错误。

在现实条件下，cat 信号与 digit 信号的变化虽然不能做到严格同步，但两者发生变化的时刻的时间差是极小的，假设该时间差为 d。我们再假设 cat 信号发生变化的时间间隔为 T，即每个数码管每次选通的时长为 T。

可以看出，如果 T 值较大，则 d/T 很小，从宏观上看，将看不出 cat 信号与 digit 信号不同步造成的影响；只有 T 值很小时，d/T 较大，这时，在每个选通周期中，我们都有较大比例的时长会看到错误的 digit 显示信号，这样，从宏观上就看到了正确数码与错误数码的混叠。

实际上，当时我直接使用 50MHz 的系统时钟信号触发 cat 与 digit 的改变，这就使得 T=20ns，而 d 的典型值也在数个纳秒左右，这就使得 d/T 值较大，误码显示效果明显，从而造成了显示故障。

将系统时钟信号进行 200 分频，改用分频时钟信号触发 cat 与 digit 的改变，此时 T 增加至 4 微秒，d/T 极小，可忽略不计，因而能够得到正确的显示效果。

2.点阵显示错误:

在本次实验中，我遇到的点阵显示错误有两个。

一是在渐亮显示模块中，射击枪的显示不正确，点阵第二行出现了第一行图像残留的影像。经分析，造成此故障的原因与造成上述数码管显示错误的原因相同，只需改变时钟分频便可以修复故障。

二是在调试点阵电路之初，我忽略了需要将 8 幅行扫描图像才能合成为一帧完整图像的点阵显示原理，试图一次性绘制完整的点阵图案，造成了显示错误。注意到需要将完整图像分解为 8 幅行扫描图像逐次绘制后，我解决了这一显示故障。

3.按键调节移动靶速度的故障:

在调测之初，我发现 BTN6 与 BTN7 无法逐级调整移动靶的速度。按下 BTN7"加速"按钮后，移动靶立刻加速至极大，而按下 BTN6"减慢"按钮后，移动靶的速度则降到极小，究其原因，是没有编写按键消抖模块，一次按键造成按键波形多次波动，波动信号多次触发移动靶加速或减速造成的。通过编写按键消抖模块，我顺利解决了这一故障。

七、总结和结论：

1.实验总结：

通过本次数字电路实验，我深刻体会到面向硬件的 VHDL 语言的并行语句的特性。在程序设计之初，我运用设计软件的编程思维来设计移动靶游戏，在不同的 process 中修改同一信号的值，而且分支语句 IF 运用较为随意，没有注意到其对于各个信号的影响，这让我在最初走了不少弯路。随后，我逐步扭转了设计思路，这使得我更加深入地体会到了数字电路设计中的代码编写与软件编程的区别。

通过本次数字电路实验，我比较深入地理解了数码管、点阵、蜂鸣器等硬件的工作原理，并能够基于它们的工作原理，对其进行比较灵活的应用。其中利用蜂鸣器实现音效控制属于提高要求，需要我们查阅相关资料，自主学习与尝试，这也提升了我的自学能力和探索钻研的兴趣。

通过本次数字电路实验，我不但深入了解了一些硬件的性能与工作原理，更认识到系统顶层设计的重要性。通过将系统功能模块化，我们减少了重复设计，这既减少了我们的工作量，也减少了逻辑元件的使用，还使得我们的程序设计更加简洁明了。通过精心设计各模块之间的信号与逻辑关系，我们可以极大地减少调测与试错所花费的时间。实际上，如果设计思路完整，整体控制模块的逻辑经过了反复的推敲，则我们写出的代码几乎不需要经过太多修改就可以在电路上正确运行，这与模拟电路实验需要反复调测硬件与线路连接的特点形成了鲜明的对比。

通过对于故障的排查，我进一步理解了 VHDL 语言与其它程序设计语言的不同。在 VHDL 的编程中，信号的同步性、竞争与冒险的可能性始终是一个关注的焦点，我们必须对这一点有充分的认识，才能够理解 VHDL 语言在硬件上的实际运行结果。这对于我们的故障排查工作很有帮助。语句的并行性、信号的同步性是 VHDL 语言与面向软件的编程语言相比的最大特色，相应地，我们初学者在使用 VHDL 语言时必须着力树立新的思维方式，注意把硬件描述语言和软件开发工具混为一谈的思维误区。

2.实验结论：

通过本次实验，我们验证了蜂鸣器、数码管、点阵的工作原理与工作性能。

其中蜂鸣器发出的声音的音高由加在蜂鸣器上的电位翻转的频率决定，翻转频率即声音频率，翻转频率越高，发出的声音也越高。

逐行扫描点阵或者逐位扫描数码管时，为防止显示错误的发生，扫描频率不宜设定得过高。

点阵显示需要将所显示的图像分解为行扫描图像，每次显示一行的图像，逐行扫描，每扫描一个周期才能完成一帧完整图像。

通过本次实验，我们验证了 VHDL 语言所具有的并行语句的特点，我们观察了由于信号不能完全同步所造成的显示错误现象。

本次实验通过分模块测试与顶层设计相结合，实现了一个综合性游戏程序。该程序能够通过控制器与游戏进程模块间的信息传递来控制游戏进程；该程序对输入的按键信息进行预处理，实现了按键检测；该程序的游戏进程模块通过输出的参数，可以间接控制点阵与数码管的显示以及蜂鸣器的输出。本次实验实现了“移动靶”游戏设计的全部基本功能和部分提高功能。

从资源利用情况上看，本设计对于逻辑元件的使用较多，这在一定程度上是控制器与游戏进程模块间的连线设计复杂造成的，这也为本次实验电路设计留下了进一步化简、提高的空间。