****



**电子电路课程设计报告**

**项 目 名 称： 四人智力竞赛抢答器**

**学 院： 电子与信息工程学院**

**专 业： 电气工程及其自动化**

**学 号 1： 2252223**

**姓 名 1： 温天佑**

**学 号 2： 2252630**

**姓 名 2： 颜辅佐**

**指 导 老 师： 赵亚辉、王晶晶、周伟**

2024年7月

# 一、设计目标

市场需求：智力竞赛在教育、娱乐和培训等多个领域非常流行。学校、公司和各种组织常常举办此类活动，抢答器作为竞赛的重要工具，其需求量稳定增长。

技术发展：FPGA技术的普及和发展，使得开发定制化、灵活的电子设备变得更加可行。FPGA具有高度的可编程性和实时性，非常适合实现复杂的逻辑控制功能。

项目学习价值：开发抢答器不仅涵盖了硬件设计、逻辑控制、信号处理等多个技术领域，还可以帮助开发者掌握FPGA设计的基本原理和实践经验。这对于我们电气工程专业的学生而言，具有很高的学习价值。

应用价值：抢答器可以用于学校的课堂互动、知识竞赛和教育培训活动，增强学生的参与感和互动性，提高教学效果。在各种智力竞赛、综艺节目和公司团建活动中，抢答器都是不可或缺的设备，能够增加活动的趣味性和竞争性。

本设计旨在开发一台供4名选手参加的智力竞赛抢答器。其基本功能特性如下：

选手编号与抢答按钮对应：

4名选手分别编号为1、2、3、4，每名选手有一个对应编号的抢答按钮。

主持人控制按钮：

设置一个控制按钮供主持人使用，用于控制系统清零（数码管灭灯）和开始抢答。

数据锁存与显示：在抢答开始后，如果有选手按下抢答按钮，系统应立即锁存该选手。的编号，并在抢答显示器上显示该编号。同时，蜂鸣器响0.5秒，禁止其他选手抢答。抢答选手的编号保持显示，直到主持人将系统清零

倒计时功能：

系统应具备9秒倒计时功能。主持人按下开始按钮后，定时器开始倒计时，定时显示器显示剩余时间。若在设定时间内无人抢答，倒计时结束时，蜂鸣器响0.5秒，输入编码电路封锁，禁止超时抢答，时间显示器显示0。

抢答成功处理：

选手在设定时间内抢答成功时，蜂鸣器响0.5秒，定时器停止倒计时，抢答显示器上显示选手编号，定时显示器上显示剩余时间，并保持显示状态，直到主持人将系统清零。

超时处理：

若倒计时结束时无人抢答，蜂鸣器响0.5秒，系统封锁输入编码电路，禁止选手超时后抢答，时间显示器显示0。

拓展功能如下：

抢答游戏进行的过程中，用4个LED实现流水灯效果。

主持人可以通过调节开关改变倒计时的时间，以适应不同的题目类型。

# 二、设计方案

为实现主持人全局控制的功能，在各底层模块中设置对于RST下降沿或低电平的扫描，如果RST有效，则全部LED熄灭，全部数码管熄灭，全部键盘和开关失效。

为实现倒计时功能，需要设计一个倒计时模块，在该模块中，设置时间变量使得该变量以一秒的时间自减，由于使用两位显示，因此需要将两位数拆分成十位和个位进行倒计时。此外，还需要将当前时间变量通过数码管进行显示。

为实现抢答功能，需要设计一个读键并消抖模块。另外，还需要设计一个抢答模块，该模块专门根据读键的结果更新倒计时模块以及更新显示模块以及蜂鸣器模块，实现抢答功能。最重要的是，在抢答模块中，需要设置一个锁存器，在没有按键按下之前，锁存器开放，可以存入按键的值，但是如果有按键按下，那么锁存器关闭，不再接收新的按键的值。

为实现超时处理的功能，需要在倒计时模块中加入一个检测倒计时自然结束的功能，如果倒计时自然结束，那么倒计时模块发送信号给蜂鸣器模块以及抢答模块，封锁对于抢答器的检测，实现游戏结束之后不能抢答的功能。

为实现拓展功能，需要增加一个调整倒计时时长模块，根据拨动开关的情况判断倒计时初始值应该调整为多少，然后将这个值一方面传入倒计时模块，更新倒计时时长，另一方面传入显示模块，使得初始值得以更新。

总体结构框图如下：

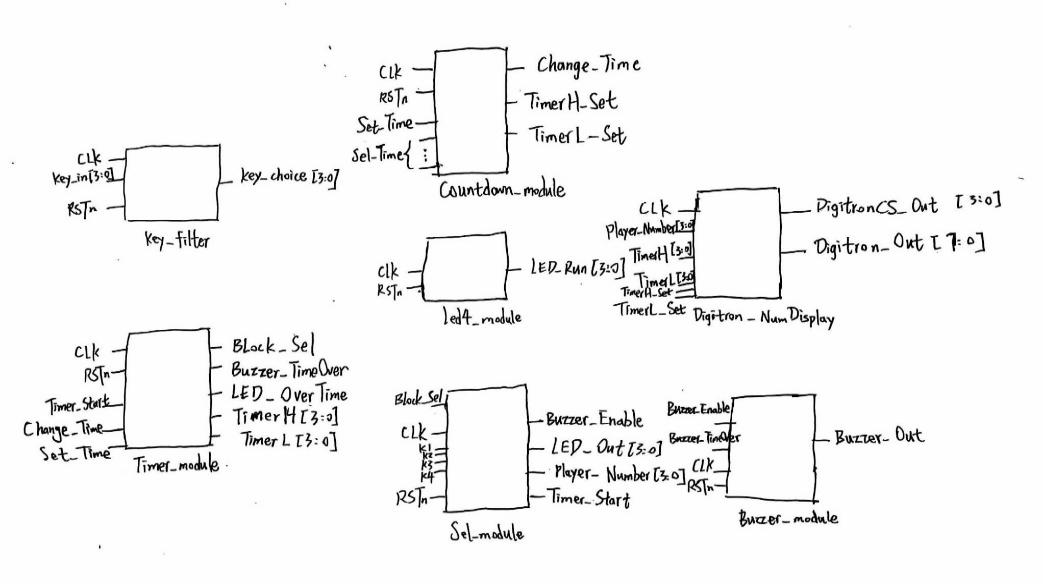


图1 总体结构框图

# 三、理论设计仿真电路

**（一）各功能模块具体实现的电路图及分析：**

我们通过Multisim绘制仿真电路原理图并完成仿真。我们根据设计方案，将原理图电路分为3大模块进行绘制：抢答锁存模块、倒计时模块、蜂鸣器模块，最后根据要求绘制各个模块的关联部分。

(1)抢答锁存模块：

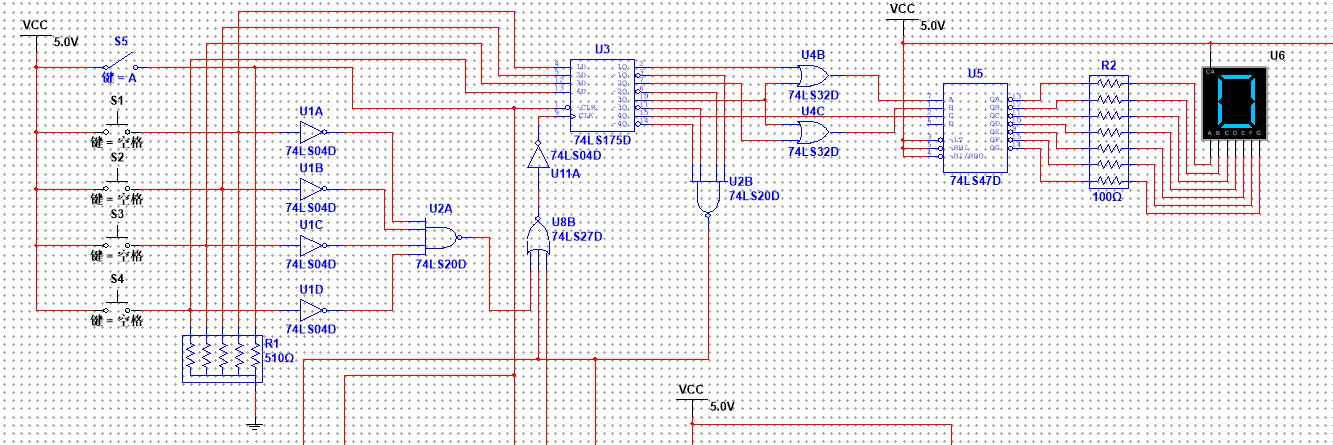


图2 抢答锁存模块原理图

在抢答锁存模块中，我们设计4个复位开关模拟选手抢答器。74LS47芯片是用于驱动共阳极7位数码管的4线-7线译码器，输入管脚从A到D，代表的二进制数位越高。为了保证驱动正常，74LS47芯片与数码管之间接入电阻。

这一模块最重要的是锁存电路的设计。我们希望有一个按键被按下时，锁存器将4个抢答器输出端同时封锁。74LS175芯片很适合这一方案，其内部为共享同一时钟信号的D触发器。我们可以设计这样一个电路，任一按键按下产生信号1作用于74LS175的时钟信号端。考虑到门电路的传输延迟时间为ns级别，而人类的反应时间为ms级别，我们可以认为不存在两个抢答器“同时”按下的情况，按键按下只会先有1个D触发器状态改变。同时触发器状态改变又产生一个信号2反馈于时钟端，将时钟信号端封锁。如此，不论按键产生的信号1如何作用，信号2封锁时钟信号端后D触发器都不会改变状态，从而实现功能。D触发器的复位则利用异步置零端。

接着就是门电路的选择，从设计的角度，我希望公平对待每一个信号输入端，这就需要1个4输入的逻辑门。然而我在Multisim的74LS系列中找不到4输入或门，便采用非门和4输入与非门的组合代替4输入“或”逻辑。

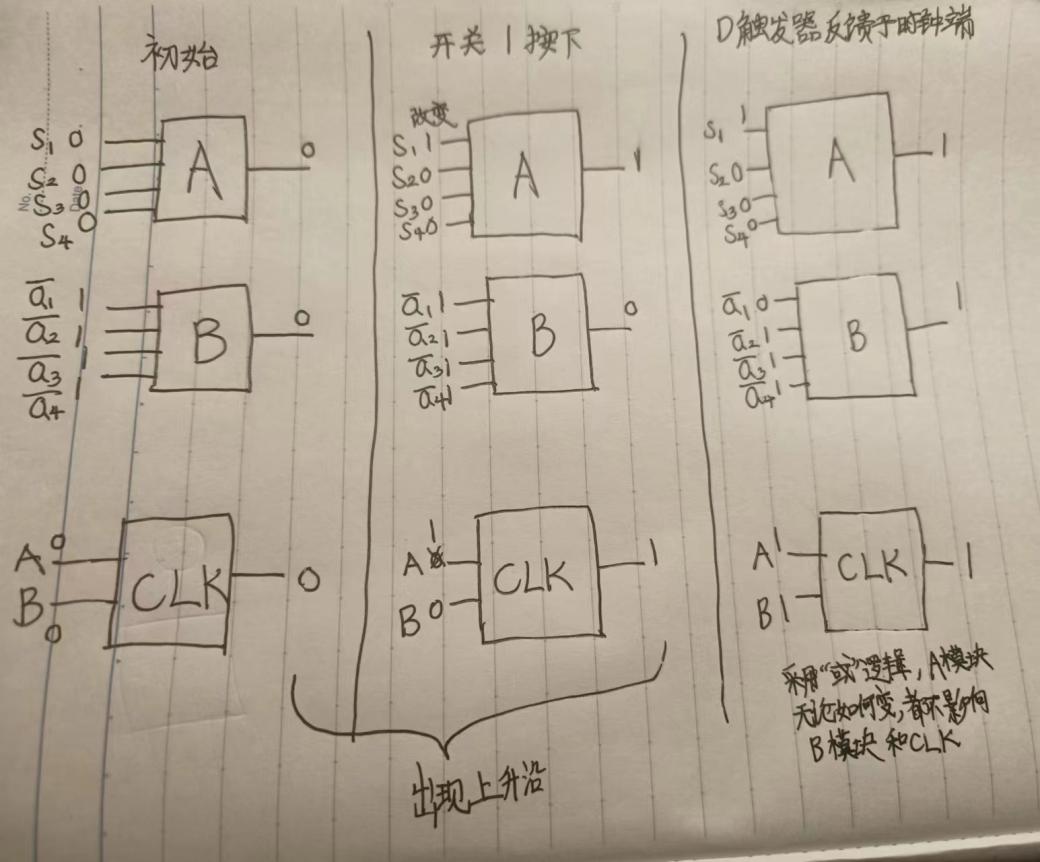


图3 门电路分析思路

采用上图的分析方式，先从电路的初始态出发，以时钟信号变化为导向，我推理出模块A应采用“或”逻辑，模块B采用“与”逻辑，时钟信号模块采用“或”逻辑，设计出了电路。其中S1、S2、S3、S4分别代表开关的输入状态，B模块的输入是4个D触发器的反相输出。

(2)倒计时模块：

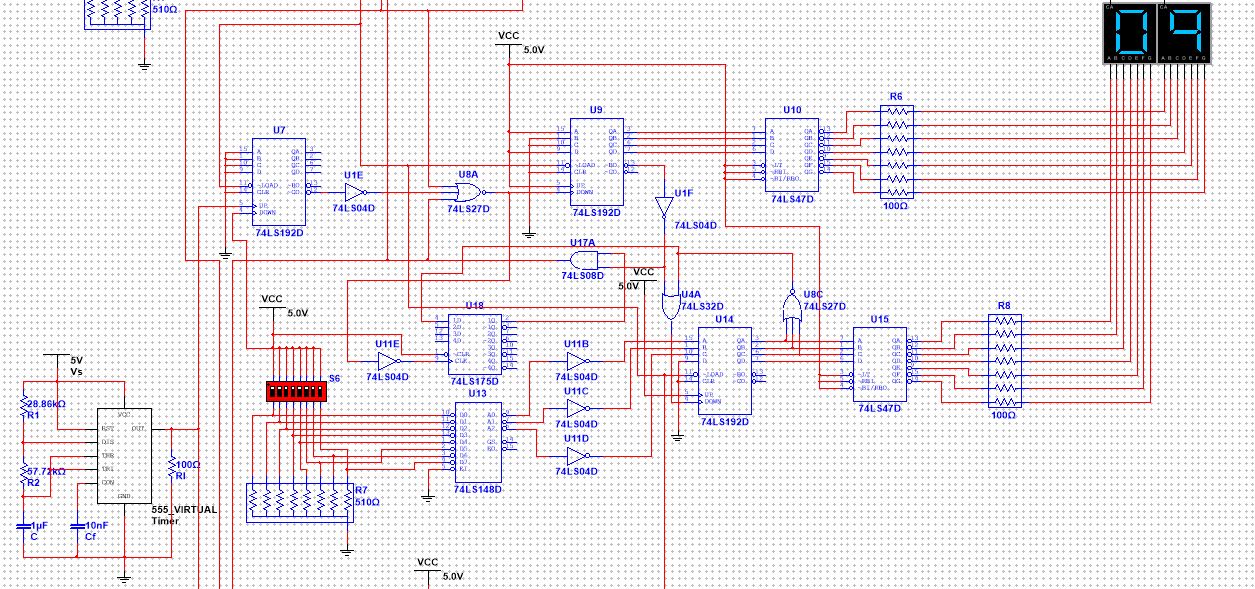


图4 倒计时模块原理图

倒计时模块需要搭建时钟电路，Multisim中提供555计时器的快速搭建模块，我们利用其搭建了1个频率为10Hz的时钟电路。10Hz的电路可以直接提供给响0.5s的蜂鸣器模块使用，利用74LS192芯片进行十分频后得到1Hz的时钟。

接下来介绍倒计时模块最重要的芯片74LS192，当初我因不熟悉该芯片的功能而在仿真时出现错误。74LS192是一个十进制计数器，支持加法计数和减法计数。74LS192的异步置零、异步置数，以及输入输出端口我就不赘述了，下面讲讲我仿真时忽视的两个要点：

1、加法计数时钟端和减法计数时钟端，在使用1个时，另一个必须置为高电平。我当时以为只要不产生上升沿即可，便把闲置的时钟端接地，结果计数器不能正常计数。现在细细思考，当上升沿信号未到达时，2个时钟端都是低电平，芯片该如何知道要以哪种方式计数？当初设计芯片的工程师还是很严谨的，也让我学会了规范使用芯片。

2、借位信号产生的条件是：工作在减法计数状态、输出为十进制的“0”、时钟信号出现下降沿。进位信号的产生也要求出现下降沿。当时我不知道芯片的这一特性，便被倒计时的退位问题困扰了许久。

可以看到我们在倒计时模块中运用了3块74LS192芯片，1块用于时钟信号十分频，剩下2块分别控制十位计数和个位计数。在调时模块中，我们利用了74LS148，8线-3线的优先编码器，可以调整十位为0~7。

本模块最重要的是十位、个位计数器之间的联动。在数电理论课的学习中，我们可以将低位的借位端接到高位的时钟端。我们还考虑了2个停止计数的条件：1、十位为“0”时十位计数器不再接收时钟信号；2、十位、个位同时为“0”时个位计数器不再接收时钟信号。

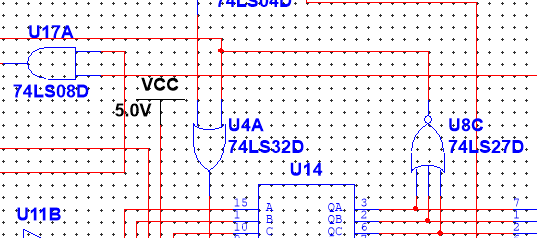


图5 十位、个位联动控制

当十位到达十进制“0”时，U8C被置为“1”，同时封锁十位计数器的时钟信号端U4A。按照我们原本的构想，U8C输出信号与个位计数器借位信号一起输入U17A，此时U8C为“1”，个位为“0”时借位信号经反相器为“1”，U17A为“1”，就可以通过“或”逻辑封锁个位时钟信号端。

但在调试过程中，我们发现数码管显示从“10”直接跳到了“00”，便采用示波器进行检查。

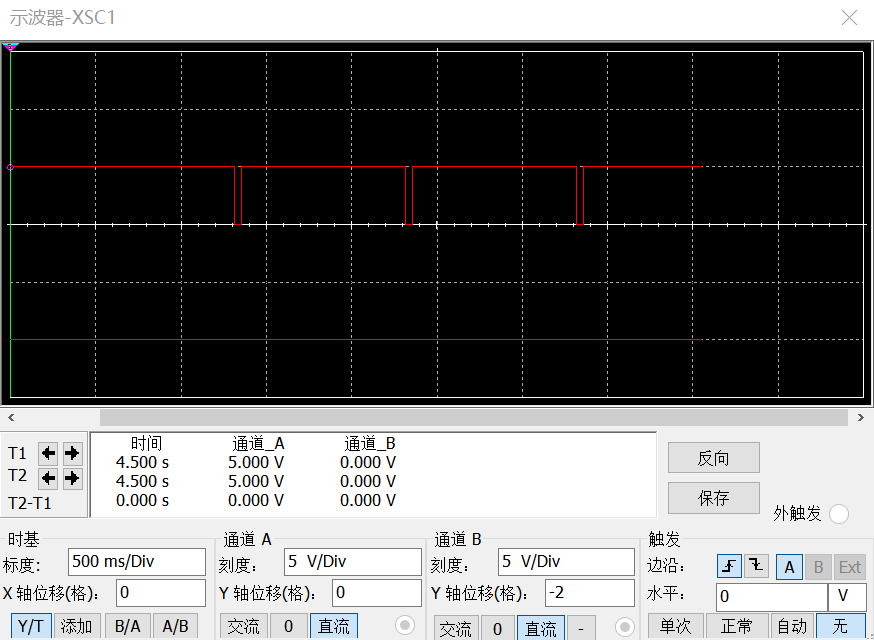
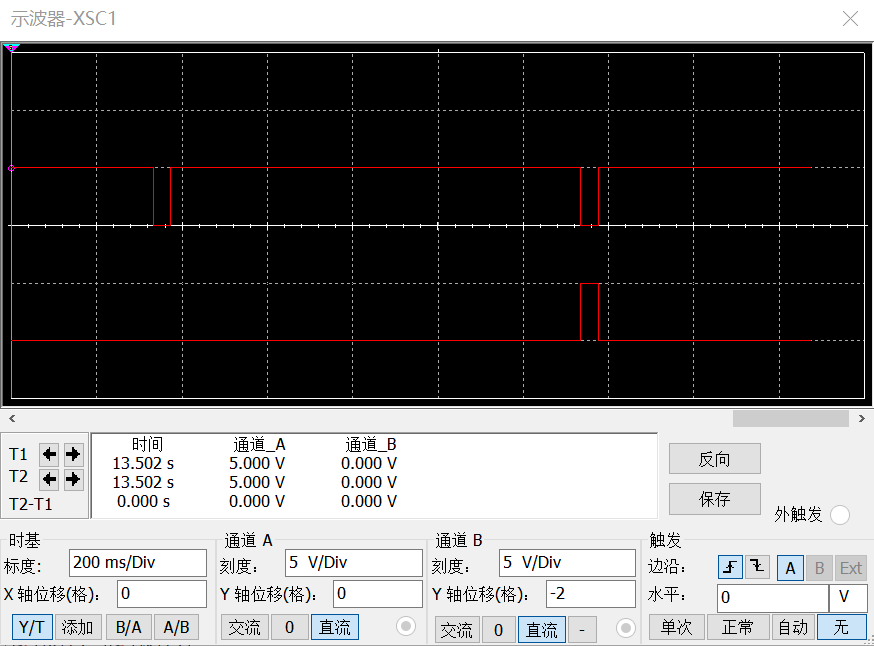


图6 示波器图1

我接入示波器的是个位计数器的时钟信号和借位信号，其中我让借位信号向下偏移。



数码管应显示“09”

数码管应显示“10”

图7 示波器图2

至此，我们发现了问题。借位信号的产生是要等到时钟信号的下降沿，而在个位时钟信号上升沿到来之前，个位已经是十进制“0”,十位由十进制“1”变为“0”，此时个位时钟信号被封锁，个位不会变为“9”。为了解决这一问题，我利用了74LS175，将个位时钟信号反相后作为D触发器的时钟信号。个位时钟信号迎来下降沿时，D触发器时钟信号正好是上升沿，而十位在变化前的一瞬是十进制“1”，输入到D触发器的部分是逻辑“0”，D触发器的输出仍是逻辑“0”。但当下一个时钟信号到来时，十位已是十进制“0”，输入到D触发器的部分是逻辑“1”，D触发器输出是逻辑“1”。通过D触发器，我们延迟1个时钟信号得到逻辑“1”，接入U17A仍可以实现对个位时钟信号的封锁。

(3)蜂鸣器模块：

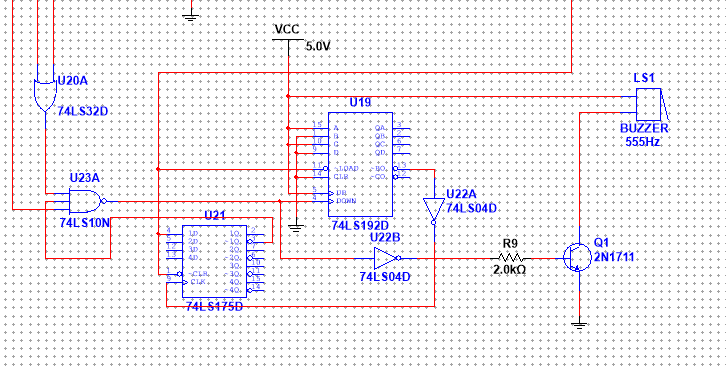
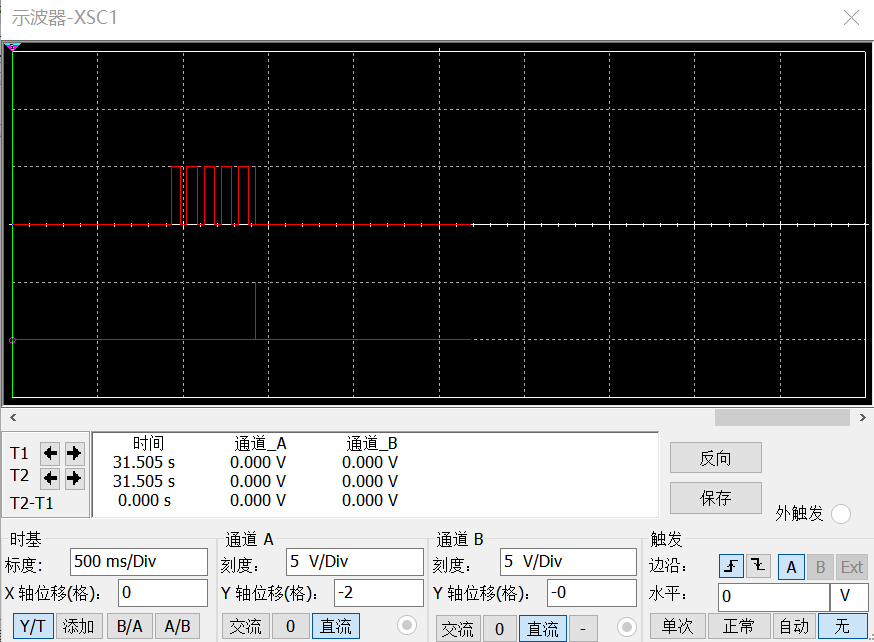


图8 蜂鸣器模块原理图

首先在蜂鸣器的选择上，我们在Multisim的元件库里只找到了有源蜂鸣器。有源蜂鸣器接通电源就能鸣叫，但发声频率是固定的。引入D触发器是因为调试时出现了与上述相同的问题，要延后反馈信号。到了蜂鸣器模块还要考虑与抢答锁存模块、倒计时模块的联动。当蜂鸣器没有接收到发声信号时(倒计时未结束、无人抢答)，计时器不工作。当接收到发生信号后，计时器计时0.5s，此时蜂鸣器发声。0.5s计时结束后，产生借位信号让D触发器锁存，进而封锁时钟信号，此时蜂鸣器输入电平改变，不再发声。



蜂鸣器运行0.5s

图9 示波器图3

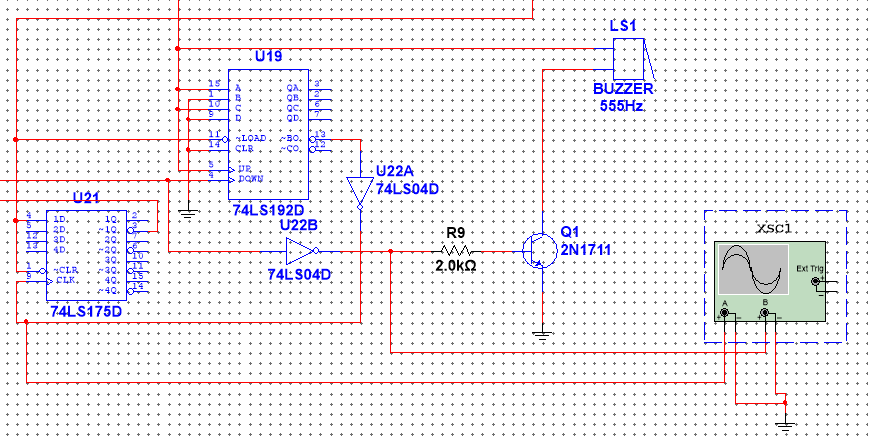


图10 示波器接法

示波器A端接的是蜂鸣器的输入，也就是时钟信号的反相；B端接的是借位信号反相。可以看到0.5s结束，反相借位信号产生正脉冲，蜂鸣器的输入则保持在低电平。

**（二）运行结果**

因为word文档里不能不好粘贴视频，我们在这里口头描述我们的运行成果，具体情况老师可以通过Multisim仿真文件查验。

(1)倒计时内按下抢答按键，数码管显示对应组号，同时蜂鸣器响0.5s，倒计时停止并维持当前数字，之后再怎么按抢答按键都没有变化

(2)倒计时结束后，蜂鸣器响0.5s，倒计时维持在“00”，之后再怎么按抢答按键都没有变化

(3)改变初始时间，倒计时能正常运行。倒计时内抢答以及倒计时结束表现同(1)(2)

**（三）电路设计亮点、存在问题及改进方法**

(1)电路设计亮点

我们认为电路设计的亮点之处就在于利用D触发器实现控制信号的延时。类比“竞争-冒险”现象，如果有多个信号同时改变可能会使结果偏离预期。组合逻辑电路对时序逻辑没有太好的办法，而我们发现利用D触发器Q\*=D的特性可以让马上要变化的信号推迟一个时钟信号，为我们处理时序逻辑提供了一个好的方法。

(2)存在问题

在我们之前仿真的过程中，我们发现数码管显示到了“00”,蜂鸣器并不会马上作响，而是要大概等待1s后再响。原因在于：倒计时结束后会传递给蜂鸣器发声信号，发声信号的传出是受借位信号控制的，而我们的计数器的时钟信号是负脉冲信号，但计数是通过上升沿控制的，所以当计数器接收到上升沿使数码管显示变为“00”后，又要经过较长的时间(1s)才能接收到下降沿而产生借位信号。这样，蜂鸣器就不能马上响应。

(3)改进方法

问题的原因在于以借位信号的产生与倒计时结束存在延时，我们决定改用与倒计时结束同时出现的信号作为控制。同时经过仿真检验，该改动不会对原有模块产生干扰。

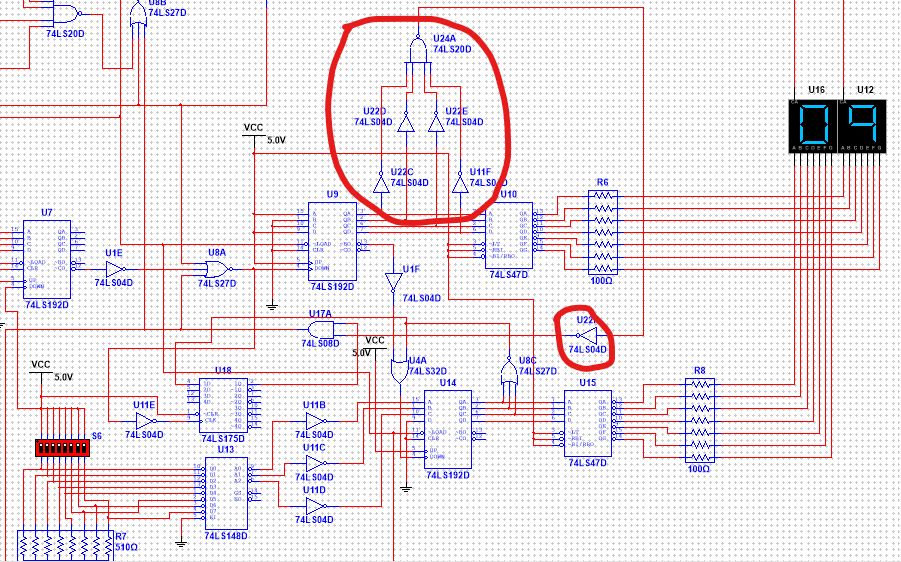


图11 改进后电路

# 四、硬件电路设计

1. FPGA实验板硬件资源分配及使用情况

A computer drawing of a circuit

Description automatically generated

图12 FPGA模块原理图

A green lines on a white background

Description automatically generated

图13 Sel\_module模块电路图

**A green lines on a white background

Description automatically generated**

图14 Timer\_module模块电路图

A diagram of a circuit

Description automatically generated

图15 Buzzer\_module模块电路图

A green and black line

Description automatically generated

图16 数码管显示模块电路

A diagram of a computer

Description automatically generated

图17 LED流水灯模块电路图

A green and white stairs

Description automatically generated

图18 countdown\_module调节倒计时时间模块电路图

A diagram of a computer

Description automatically generated

图19 key\_filter键盘消抖模块电路图

FPGA管脚配置：

set\_pin\_assignment { Buzzer\_Out } { LOCATION = H11; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { CLK } { LOCATION = R7; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { DigitronCS\_Out[0] } { LOCATION = C9; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { DigitronCS\_Out[1] } { LOCATION = B6; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { DigitronCS\_Out[2] } { LOCATION = A5; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { DigitronCS\_Out[3] } { LOCATION = A3; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[0] } { LOCATION = A4; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[1] } { LOCATION = A6; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[2] } { LOCATION = B8; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[3] } { LOCATION = E8; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[4] } { LOCATION = A7; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[5] } { LOCATION = B5; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[6] } { LOCATION = A8; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Digitron\_Out[7] } { LOCATION = C8; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_In[0] } { LOCATION = E11; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_In[1] } { LOCATION = D11; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_In[2] } { LOCATION = C11; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_In[3] } { LOCATION = F10; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { LED\_Out[0] } { LOCATION = D3; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { LED\_Out[1] } { LOCATION = E4; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { LED\_Out[2] } { LOCATION = C1; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { LED\_Out[3] } { LOCATION = C2; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { LED\_OverTime\_Out } { LOCATION = F16; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { RSTn } { LOCATION = A9; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Set\_Time } { LOCATION = A10; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_Row[0] } { LOCATION = D9; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_Row[1] } { LOCATION = F9; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_Row[2] } { LOCATION = C10; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Key\_Row[3] } { LOCATION = E10; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { LED\_Run[0] } { LOCATION = B14; IOSTANDARD = LVCMOS33; DRIVESTRENGTH = 8; PULLTYPE = NONE; }

set\_pin\_assignment { LED\_Run[1] } { LOCATION = B15; IOSTANDARD = LVCMOS33; DRIVESTRENGTH = 8; PULLTYPE = NONE; }

set\_pin\_assignment { LED\_Run[2] } { LOCATION = B16; IOSTANDARD = LVCMOS33; DRIVESTRENGTH = 8; PULLTYPE = NONE; }

set\_pin\_assignment { LED\_Run[3] } { LOCATION = C15; IOSTANDARD = LVCMOS33; DRIVESTRENGTH = 8; PULLTYPE = NONE; }

set\_pin\_assignment { Sel\_Time1 } { LOCATION = B10; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Sel\_Time2 } { LOCATION = A11 ; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Sel\_Time3 } { LOCATION = A12; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Sel\_Time4 } { LOCATION = B12 ; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Sel\_Time5 } { LOCATION = A13; IOSTANDARD = LVCMOS33; }

set\_pin\_assignment { Sel\_Time6 } { LOCATION = A14 ; IOSTANDARD = LVCMOS33; }

1. Verilog程序说明

图20是截取自底层模块Sel\_module的部分代码：

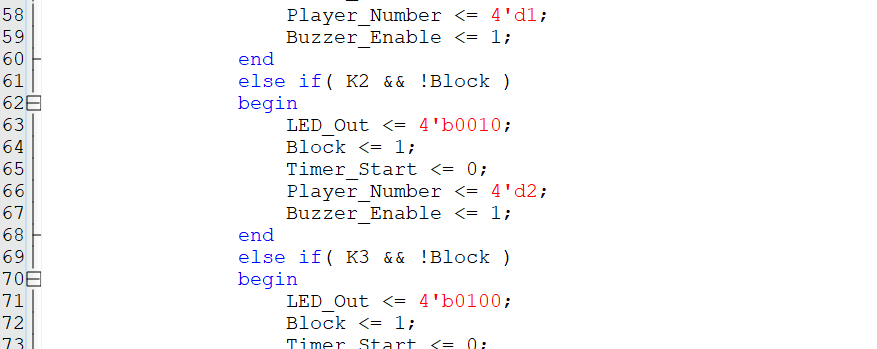
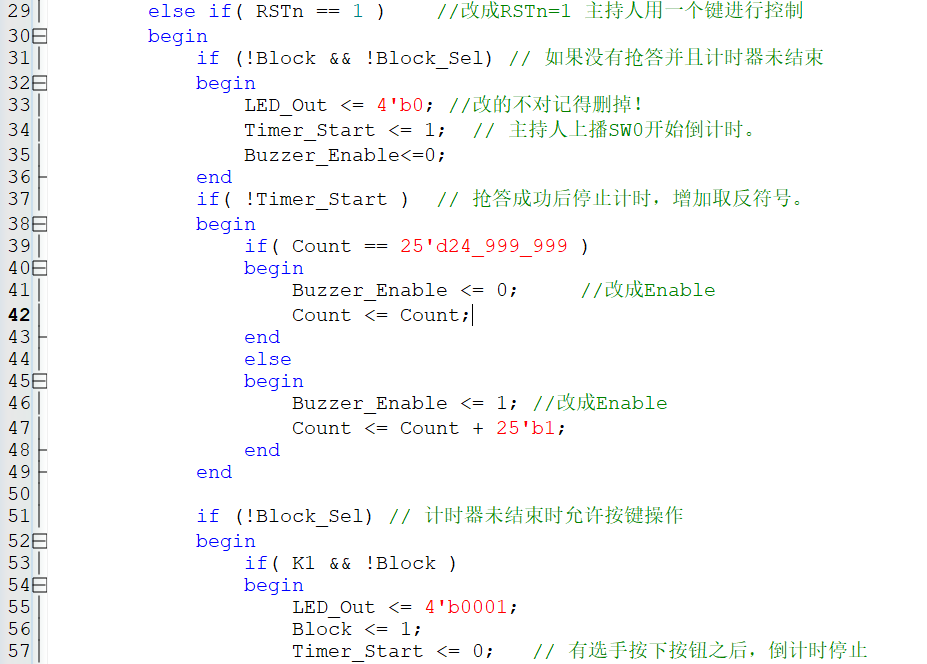
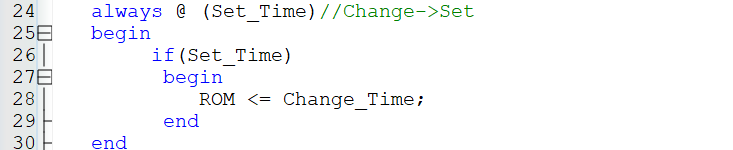


图20 Sel\_module核心代码

37-49: Timer\_Start是“计时器启动标志位”，当第一信号鉴别、锁存模块锁存了第一抢答信号后，令Timer\_Start变为0，关闭答题计时器电路。当Timer\_Start变为0后，37-49行代码将Buzzer\_Enable（“抢答成功鸣笛使能位”）置为1，保持半秒后，再置为0。这里用到了一个很简单的计数器Count，用于控制半秒的时间。

53-60：判断是否已经锁存以及第一组别是否有抢答操作，Block是锁存信号，高电平表示电路已锁存。若第一组抢答，且在此之前未有其他组抢答成功（Block为0），则认为第一组抢答成功。55-59行依次为点亮第一组选手对应的LED灯，锁存电路，关闭倒计时计时器，将第一组别的序号送给数码管，让蜂鸣器响。其他组别的代码同理。

图21是截取自底层模块Timer\_module的部分代码：



A computer screen shot of a program

Description automatically generated

A white background with black text

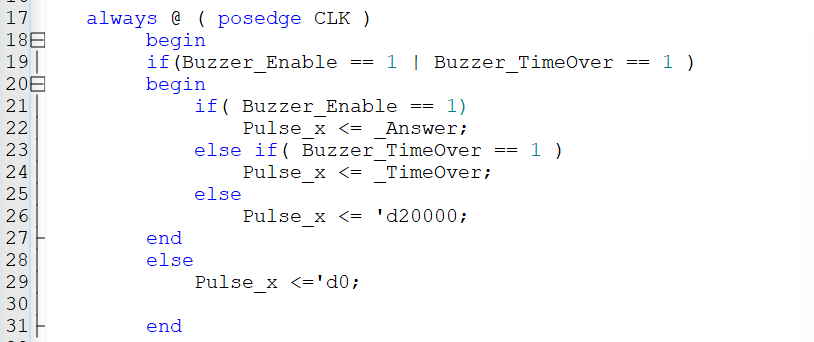
Description automatically generated

图21 Timer\_module的部分代码

24-30：如果拨动开关SW1上拨开启，则将倒计时重新设置为新的倒计时时长。

45-79：RST有效时（低电平），数码管不显示，LED不亮，表示比赛没有开始。RST为高电平且倒计时开始之后，先将最新的倒计时时长更新给TimerH和TimerL，然后62-77行的代码负责进行倒计时。注意always触发条件中的CLK1是1秒1个上升沿，实现倒计时功能。

图22是截取自底层模块Buzzer\_module的部分代码：



A screenshot of a computer code

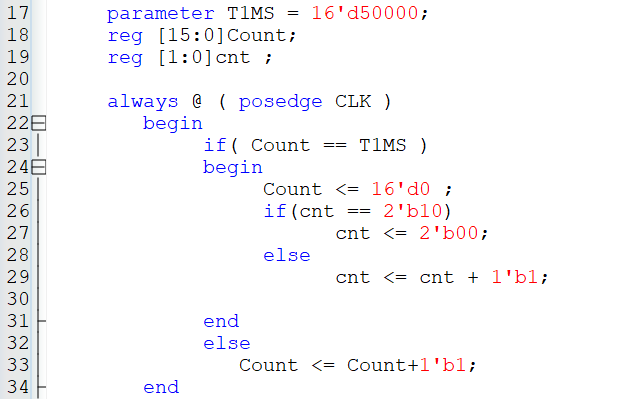
Description automatically generated

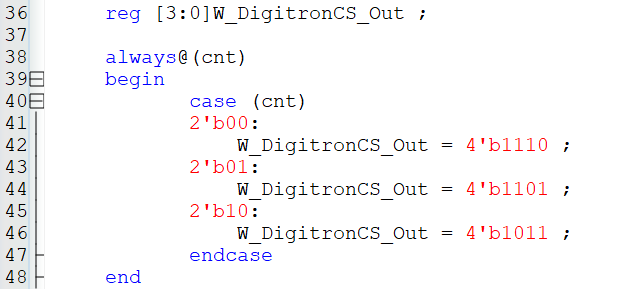
图22 Buzzer\_module的部分代码

17-31：通过“鸣笛使能位”控制送到蜂鸣器上的电压和电压变化频率。例如，当某个小组抢答成功时，“抢答成功鸣笛标志位” Buzzer\_Enable将被置为1且经过半秒后变回0，这半秒内将常量\_Answer的值（17'd95419）赋给Pulse\_x，半秒后Pulse\_x的值变为'd20000。实现抢答成功后蜂鸣器响半秒。如果没有组别抢答成功，而是倒计时自然结束，那么蜂鸣器Buzzer\_TimerOver将被置1，且经过半秒后变回0，这半秒内将常量\_TimeOver 的值（17'd50607）赋给Pulse\_x，半秒后Pulse\_x的值变为'd20000。实现倒计时自然结束之后蜂鸣器响半秒。

33-50：这是一个小型计数器。若Pulse\_x的值为“\_Answer”或“\_TimeOver”，则控制W\_buzzer的值以一定频率在“0”和“1”之间翻转，这样便形成了一定频率的脉冲信号。W\_buzzer的值即为将送给蜂鸣器的值，该脉冲信号的频率即控制蜂鸣器发声的频率。

图23是数码管显示模块：





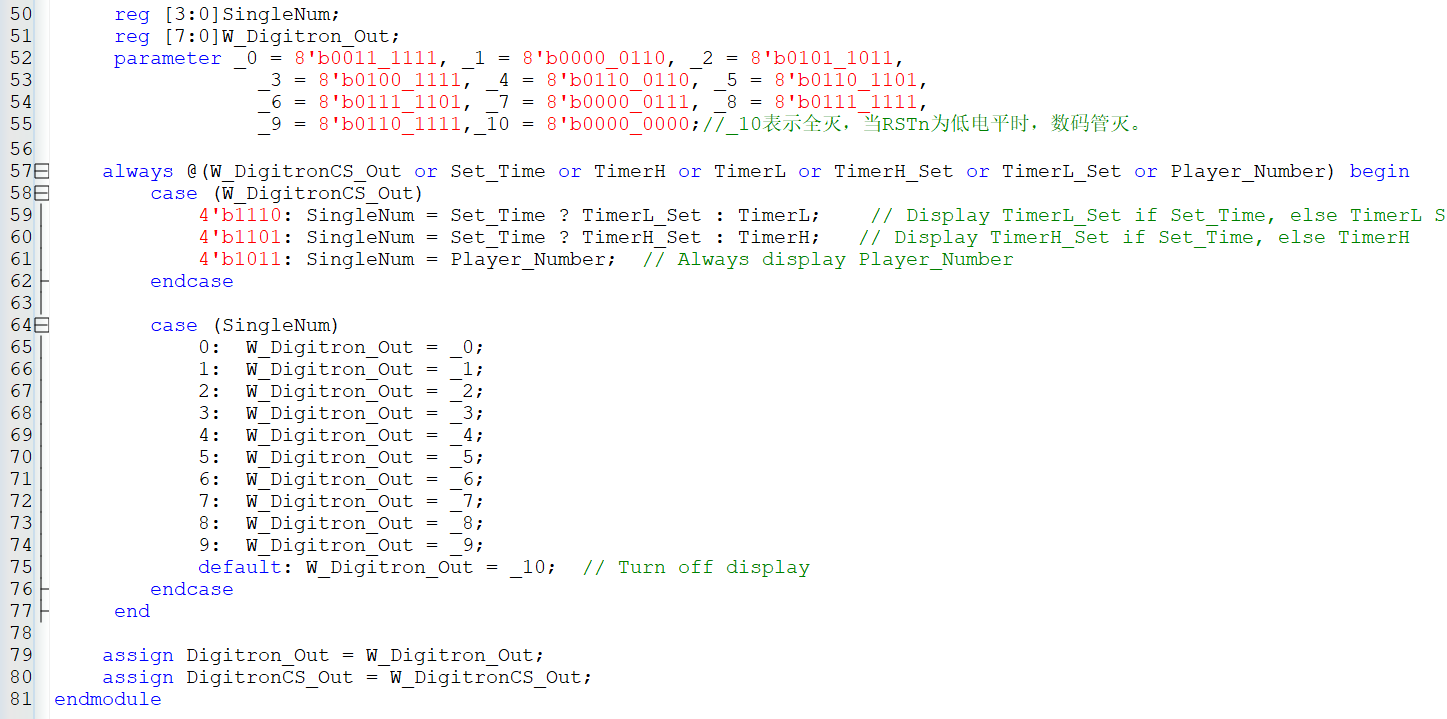


图23 数码管显示模块

17-34：分频程序，用于产生频率为250KHz的扫描信号,并在每个扫描周期对cnt在0-2循环计数。

36-48：片选信号DigitronCS\_Out的扫描程序。W\_DigitronCS\_Out[3:0]信号的值将在程序的最后赋给DigitronCS\_Out[3:0]信号，当它的某一位为0时，对应的数码管选中，反之则不选中。在本实验中使用数码管DIG1、DIG0显示显示倒计时数字，DIG2显示抢答号，因而DigitronCS\_Out[3:0]的值根据cnt计数结果在“1110,1101,1011”之间变换，变换频率为250KHz。

50-80: 根据SingleNum信号的值选择输送到数码管的常量，控制字码段的点亮情况。例如当SingleNum的值为4'b1000时，输送到数码管的常量为“\_8”, 即Digitron\_Out[7:0]的值为8'b0111\_1111，字码段A、B、C、D、E、F、G全部点亮，DP（小数点）熄灭，数码管显示数字“8”。

图14是调整倒计时时长模块的部分代码：

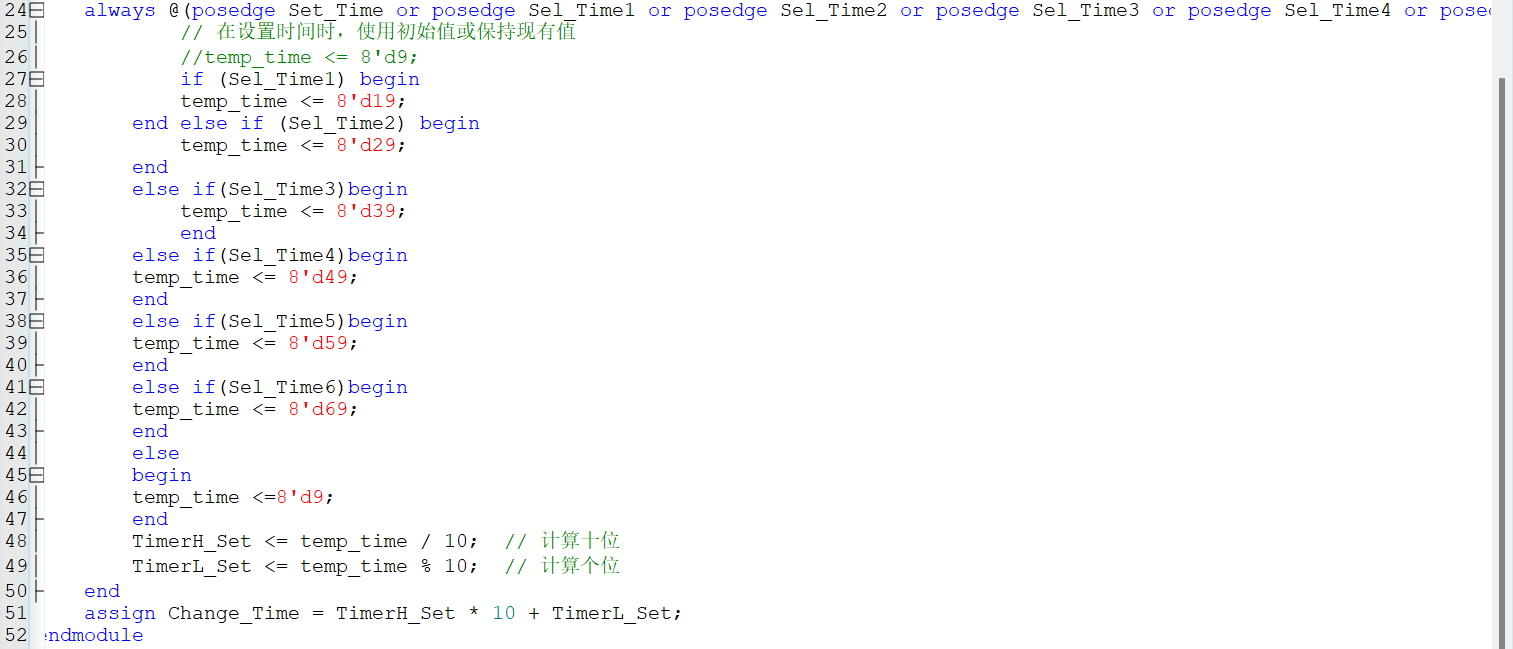


图24 countdown\_module部分代码

24-52:设置时间信号上升沿和各个上拨开关的上升沿为触发条件，如果SW1上拨，那么新的倒计时时长设置为19秒，其他同理。设置优先级是：SW1>SW2>…>SW7

图15是矩阵键盘消抖模块的部分代码：

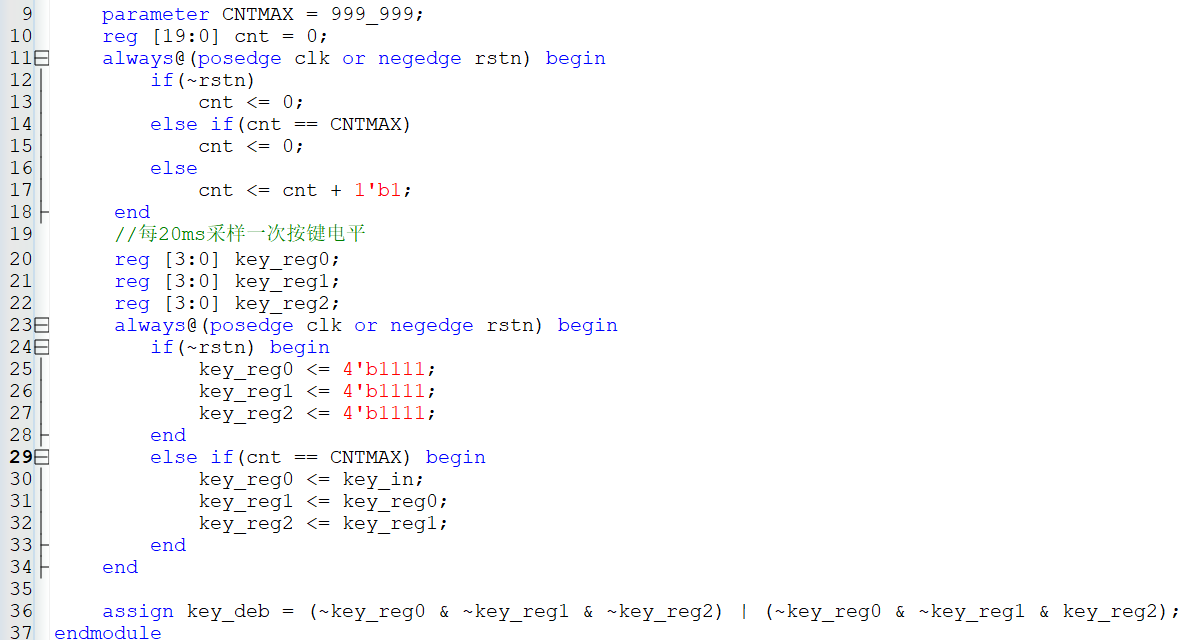


图25 key\_filter模块部分代码

9-37：通过FPGA的时钟信号对输入的按键信号进行采样和过滤。它利用20ms的定时器周期，对按键输入信号进行多次采样并存储在寄存器中，通过比较多次采样结果来判断按键是否稳定，从而生成去抖后的按键信号输出。

1. 运行结果、存在问题分析及改进方法

**最终运行结果：**

拨动开关SW1~SW7总共7个开关可以调节7档不同的计时时长：9秒，19秒，29秒，39秒，49秒，59秒，69秒。SW0为主持人控制开关，如果倒计时为9秒，则主持人直接通过控制开关控制抢答开始，若要调整不同的时间，主持人需要先下拉RST键，清零当前抢答，然后上推SW1进行时间调节，然后上推对应倒计时时长的开关，调节到对应时间，然后下拉SW1，将新的倒计时时长赋给倒计时计时器。时间设置好后，上推RST按钮，重新开始抢答，此时抢答倒计时调整为新的时间。

**结果分析：**

尝试使用矩阵键盘控制倒计时时长，但是可能由于矩阵键盘抖动问题，导致按下键盘之后键值不稳定，难以实现按一次按键时间自增1，而是会出现自增不确定值的情况，因此改用拨动开关进行。

**后续改进空间：**

当前矩阵键盘加入消抖之后理论上可以使用矩阵键盘实现倒计时时间的更细粒度调整。

另外，目前拨动开关仍有部分情况下出现不受控的倒计时数字，可能是因为设置倒计时时间的程序段有部分开关情况没有考虑到，导致寄存器出现没有赋值的情况，出现不受控的赋值操作。

**已经解决的问题：**

曾出现无论如何在程序中修改倒计时开始的时间，程序都总是从09秒开始倒计时，通过打印不同分支的进入情况，发现问题出在给定时器赋初值的语句和下方进行倒计时的语句的衔接逻辑不妥，导致每次要开始计时之后，程序会从09开始计时。改进方法：每次开始倒计时的信号为1时，会检查当前是不是开始计时之前的数码管状态，如果是，就赋初值。

**扩展模块的加入：**

新模块的输入信号：CLK,RSTn,Set\_Time,Sel\_Time(不同SW开关对应不同的倒计时时间),模块内部用一个temp\_time寄存器存储最新的倒计时时间，最后输出Change\_Time将这个模块的新的倒计时时间传给Timer模块，Timer模块用一个新的reg寄存器接收Change\_Time的值， 再变为高8位和低8位进行倒计时模式下（RSTn高电平）的显示。 显示部分需要增加两个输入信号，TimerH\_Set和TimerL\_Set使得数码管可以显示当前SW开关对应的时间长度。

# 五、课程设计心得体会

# （一）组员1心得体会

调试体会：

1.使用点亮一个LED的方式，解决了倒计时自然结束之后还可以继续抢答的问题。解决方法：在timer module里面增加了一个锁存器，使得自然计时结束之后，Block\_Sel的值恒为1，重置RSTn之后Block\_Sel的值变为0。如果不锁存的话，Block\_Sel信号不会被存储下来，导致下一次时钟上升沿到来之后，抢答按钮又可以工作。

2.多个 always 块中同一个锁存器的驱动条件需要相同，以确保在同一时刻对同一信号进行唯一且确定的赋值。如果在不同的 always 块中对同一信号进行赋值，并且赋值条件不同，会导致竞争和冲突，从而产生不确定的行为。在 Verilog 中，如果一个信号在多个地方被驱动，且驱动条件不同，会引发冲突。例如，以下情况是有问题的：

always @ (posedge CLK)

begin

if (condition1)

signal <= value1;

end

always @ (posedge CLK)

begin

if (condition2)

signal <= value2;

end

上述代码中，signal 在两个不同的 always 块中被驱动，并且驱动条件（condition1 和 condition2）不同。这会导致竞争和不确定的行为。

3.对于IP核的认识：IP核可以帮助设计人员在集成电路设计中节省时间和资源。IP核是预先设计和验证的功能模块，可以直接嵌入到更大的系统中。这些模块可以是软核（源代码或HDL代码）、硬核（已布线的物理设计）或固核（位于硬件中的中间形式）。 Modelsim与TD联合仿真可以没有IP核。IP核提高设计效率，类似于基本函数封装成库在高级语言程序设计中直接进行调用，不必从头造轮子。此外，IP核可以提高模块复用性，可以使得产品设计周期缩短，更快推向市场。

4.TD与Modelsim联合仿真：在TD软件内点击simulation，即可生成testbench文件，将这个文件进行修改，加入输入信号后，与其他.v源文件一同导入Modelsim即可仿真，仿真时需要选择work文件夹下的\_tb.v文件进行仿真，仿真步长需要合理设置，才能使波形更好地显示。联合仿真结果如下：

A screenshot of a computer

Description automatically generated

图26 TD与modelsim联合仿真波形图

仿真结果分析：时钟，RST，LED流水灯，预设输入波形如期变化，但是数码管波形一直是红线，可能原因：testbench文件中的仿真时间设置不够合理，使得整个仿真跑完之后仍没有触发数码管点亮。

Testbench文件内容如下：

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

图27 Testbench文件

1. 

图28 TD编译错误

由于不同模块可能在同一条件下使得同一个变量比如TimerH，携带不同的输出，因此为了拓展功能的实现，需要重新定义TimerH\_Set来存放设置倒计时的数值。

# 组员2心得体会

我主要负责Multisim理论设计仿真部分。完成理论仿真设计后，我觉得我对74LS系列芯片有了进一步的掌握，也很好地锻炼了我的电路逻辑分析能力。但同时，我也深深地感受到74LS系列与FPGA的差距。比如利用74LS系列组装时钟电路，我们要搭建1个基本的时钟振荡器(555单稳态电路)，然后利用计数器的组合调制出不同频率的时钟信号，要想改变时钟频率要在原理图上做很大的修改。但在FPGA中，只需要在Verilog代码中改变计时次数，就可以得到不同频率的时钟信号，便利性远强于74LS系列。我完成了理论设计的工作后，并不能很好地上手TD。

我们这届也是第一次使用FPGA开发板做电子设计课设，理论设计部分考察学生通过硬件构建逻辑电路，是承接了我们学习的模电、数电课程；TD设计部分则是要求我们掌握一门新的软件工具，更考察学生的创新探索能力。如何分配工作量能使学生同时掌握好这两方面的能力，仍是要探索的。

在Multisim仿真中我遇到了各种预期之外的结果，大多数是电路逻辑设计不严谨导致的，这部分我在理论设计部分也大致说明了。下面我想讲一个关于Multisim仿真时的问题：

在第一次Multisim仿真时，我发现仿真运行得太慢了。比如现实世界过了1s，Multisim仿真中只过了0.05s.



图29 Multisim实际时间

改进方法如下：

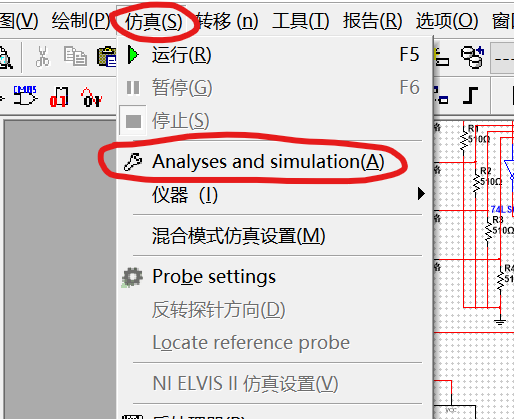


图30 选中“仿真”的“Analyses and simulation”

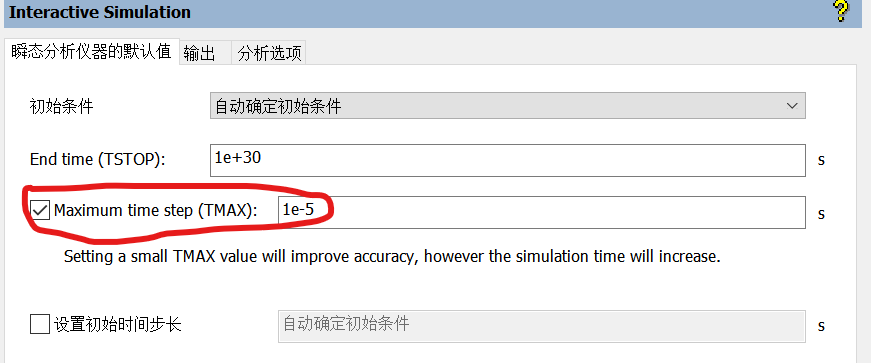


图31 将该部分数字改大即可

经过试验，将数字改至0.01--0.001之间，Multisim仿真时间与现实时间差不多

# 参考资料及网站

74LS192管脚功能介绍：

[www.51hei.com/chip/318.html](http://www.51hei.com/chip/318.html)

Multisim学习：

<https://www.bilibili.com/video/BV1az411b7aN/?spm_id_from=333.337.search-card.all.click&vd_source=7d08ea46774f06c88bc62c4c72ab28cc>

软件安装，破解与环境配置：

<https://www.bilibili.com/video/BV1WC4y1p72B/?spm_id_from=333.337.search-card.all.click&vd_source=23bb86bc8e22eff8e2bea3c693a1eed1>

<https://www.bilibili.com/video/BV1vt421b7BC?p=3&vd_source=23bb86bc8e22eff8e2bea3c693a1eed1>

例程学习：

硬木课堂学习资料

<https://www.yuque.com/yingmuketang/01/qha859>

Verilog语法学习：

<https://www.runoob.com/w3cnote/verilog-tutorial.html>

ChatGPT+例程学习

TD与Modelsim联合仿真：

<https://www.bilibili.com/video/BV1vt421b7BC?p=5&vd_source=23bb86bc8e22eff8e2bea3c693a1eed1>