倒计时定时器的设计、制作与调试

胡东东 3240102873

一、实验目的

1、学习掌握用 Arduino UNO 设计倒计时定时器;

2、学习掌握 PCB 电路板的设计和制作;

3、学习掌握 Arduino UNO 扩展板的设计与制作;

4、学习掌握旋转编码器的使用。

二、 实验任务与要求

1、用 Arduino UNO 设计倒计时定时器，通过旋转编码器选择设定倒计时时间（设定标准时间数组），时间到时蜂鸣器报警;

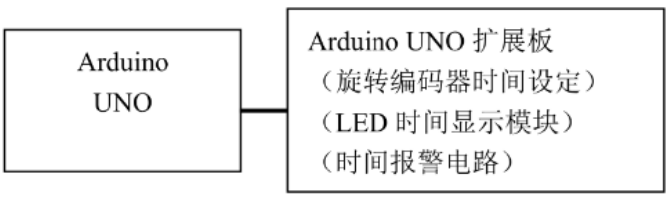
2、设计电路，完成相应器件的选择，制作 Arduino UNO 扩展板;

3、编写并调试倒计时定时器程序;

4、将制作的扩展板与 Arduino UNO 板组装后，进行系统联调。

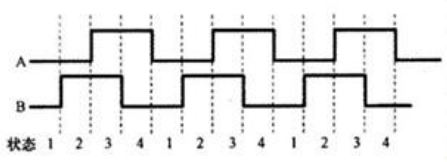
三、实验原理

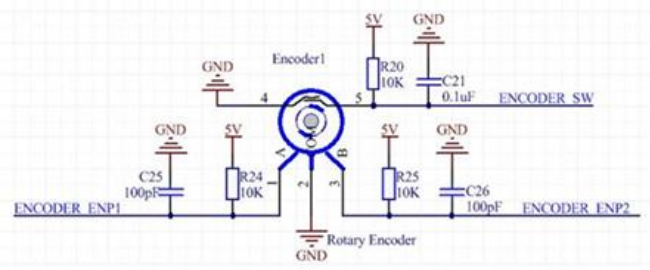
本项目使用 Arduino UNO 与扩展板实现倒计时定时器，下面对各部分模块进行说明。



（1）旋转编码器

旋转编码器是一种数字器件，这种器件有两个输出(A 和 B)。当旋转旋钮时，输出会有变化，这个变化表明旋钮是顺时针旋转还是逆时针旋转。实验中可根据两个脉冲波形变化情况判断旋钮旋转方向，从而更新时间设定。

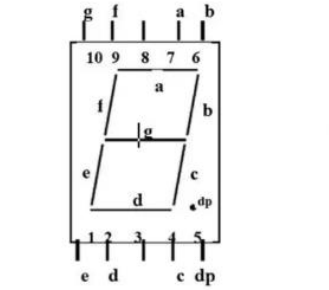


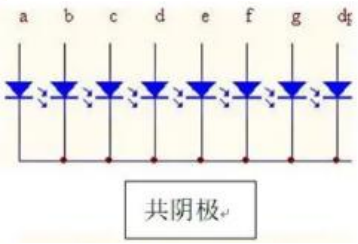


（2）LED 时间显示模块

①7 段数码管结构

7 段数码管一般由 8 个发光二极管组成，其中由 7 个细长的发光二极管组成数字显示，另外一个圆形的发光二极管显示小数点。当发光二极管导通时，相应的一个点或一个笔画发光。发光二极管的阳极连在一起的称为共阳极数码管，阴极连在一起的称为共阴极数码管（本次设计要求使用共阴极数码管）。





7 段数码管内部字段 LED 和引脚

②7 段数码管驱动方法

本次设计的 7 段数码管的显示方法采用动态显示，动态显示驱动是将所有数码管的 8 个显示笔划"a,b,c,d,e,f,g,dp "的同名端连在一起，另外为每个数码管的公共极 COM 增加位元选通控制电路，该位元就能够显示出字形，没有选通的数码管就不会亮。透过分时轮流控制各个 LED 数码管的 COM 端，就使各个数码管轮流受控显示，这就是动态驱动。在轮流显示过程中，每位元数码管的点亮时间为 1～2ms，由于人的视觉暂留现象及发光二极管的余辉效应，只要扫描的速度足够快，给人的印象就是一组稳定的显示，不会有闪烁感。动态显示能够节省大量的 I/O 口，而且功耗更低。

（3）时间报警电路

报警可采用蜂鸣器。蜂鸣器按工作原理可分为压电式和电磁式：前者利用压电陶瓷的压电效应带动金属片振动发声；后者利用电磁原理，通过电磁力与振动膜弹力的交替作用产生声音。按是否内置振荡源，蜂鸣器又分为有源和无源两类。此处的“源”指振荡源而非电源。有源蜂鸣器内部集成振荡电路，接入额定直流电压即可发声，程序控制简便；无源蜂鸣器则需外部提供 2kHz 至 5kHz 方波驱动，其优点是成本低、频率可控，能实现不同音阶效果，且在特定场合可与 LED 复用同一控制端口。

四、实验方案设计

（1）硬件电路设计

① 电路总体设计

定时器总是处于两种状态之一：停止定时与定时运行。在停止定时状态，转动旋转编码器可从预设的标准时间数组中选择倒计时时间；在定时运行状态，定时器进行倒计时，归零后蜂鸣器报警。按下旋转编码器上的按钮可对这两种状态进行切换。编写 Arduino 代码时引入 EEPROM 库，用于保存最后一次使用的时间，确保重新上电后能恢复断电前的设定值。

硬件电路的设计可分为以下四个模块：

旋转编码器模块：根据 A、B 两相电平变化正确识别旋转方向，并检测按钮的按下动作。

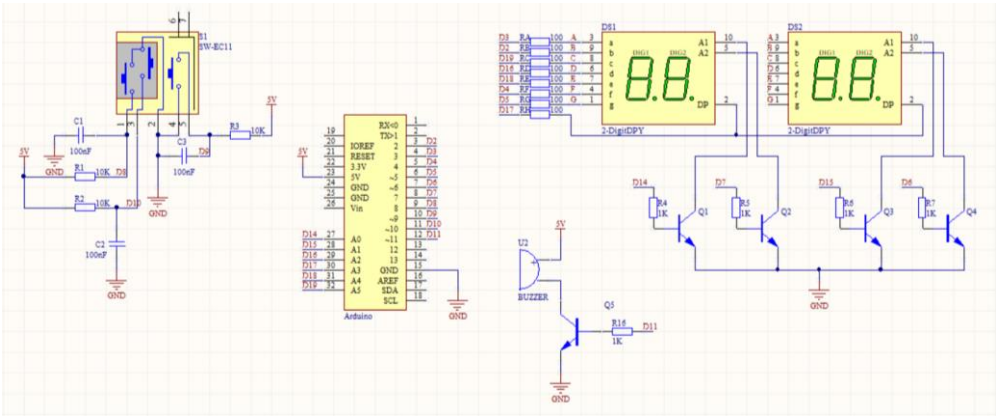
数码管位选控制模块：通 NPN 型三极管控制四个数码管的共阴极通断，实现分时选通；

数码管段选控制模块：控制当前选通的数码管显示对应数字的段码；

蜂鸣器模块：倒计时归零时控制蜂鸣器发出报警提示；

设计要求使用共阴数码管，故位选驱动采用 NPN 型三极管，发射极共地连接。为优化 PCB 布线，引脚分配遵循物理位置就近原则：段选信号（a 至 dp）依次分配至 D3、D2、D19、D16、D18、D4、D5、D17，位选信号（四个数码管共阴极控制）分配至 D14、D7、D15、D6，以上引脚在 Arduino UNO 的物理封装上分别集中在相邻区域，便于 PCB 走线时减少交叉与绕线。蜂鸣器由 D11 端口控制，旋转编码器的 A、B 相信号和按钮信号分别由 D8、D10、D9 端口读取，均就近连接至对应外设。该分配方案兼顾了原理图逻辑清晰与 PCB 布局布线的便利性。

（2）电路原理图（本人设计）：



原理图中使用的电容电阻值均已在原理图中注明：即$RA\sim RH=100\Omega$，$R1\sim R3=10k\Omega$，$R4\sim R7=1k\Omega$，$C1\sim C3=100nF$。

（3）各模块设计说明：

①数码管位选控制模块

由四个 NPN 型三极管组成，每个三极管的基极通过限流电阻与 Arduino UNO 的一个数字输出接口相连，发射极共同接地，集电极分别连接四个数码管。Q1 至 Q4 依次控制两个双位数码管从左向右的四个数字位。以 Q1 为例，其集电极接数码管 DS1 的位选引脚，基极接 Arduino UNO 的 D14 端口。当 D14 端口输出高电平时，Q1 导通，对应数码管的共阴极被拉至低电平，该位数码管进入显示状态，此时可通过控制段选端口电平来点亮对应数字；当 D2 端口输出低电平时，Q1 截止，该位数码管熄灭。其余三个位选控制工作模式同理。四个位选端口通过分时扫描的方式轮流导通，利用人眼视觉暂留效应实现四位数码管的稳定显示。

②数码管段选显示模块

选用 Arduino UNO 的 8 个数字输出接口，分别控制每个数码管的八个发光二极管段。当某位数码管被位选信号选中进入工作状态后，将待显示数字对应的段选端口置于高电平，即可点亮相应笔段。例如显示数字“0”时，需要点亮 a、b、c、d、e、f 段而熄灭 g 段，此时相应端口输出高电平或低电平。段选信号通过 8 个限流电阻接入数码管对应引脚，以保护数码管和 Arduino 端口。

③蜂鸣器控制模块

选用有源蜂鸣器，通过 NPN 三极管控制蜂鸣器，三极管基极连接至 Arduino UNO 的 D11 端口，发射极接地。当倒计时归零时，端口输出高电平，三极管导通，蜂鸣器通电即发声报警；其余时间 D11 端口保持低电平，蜂鸣器不工作。

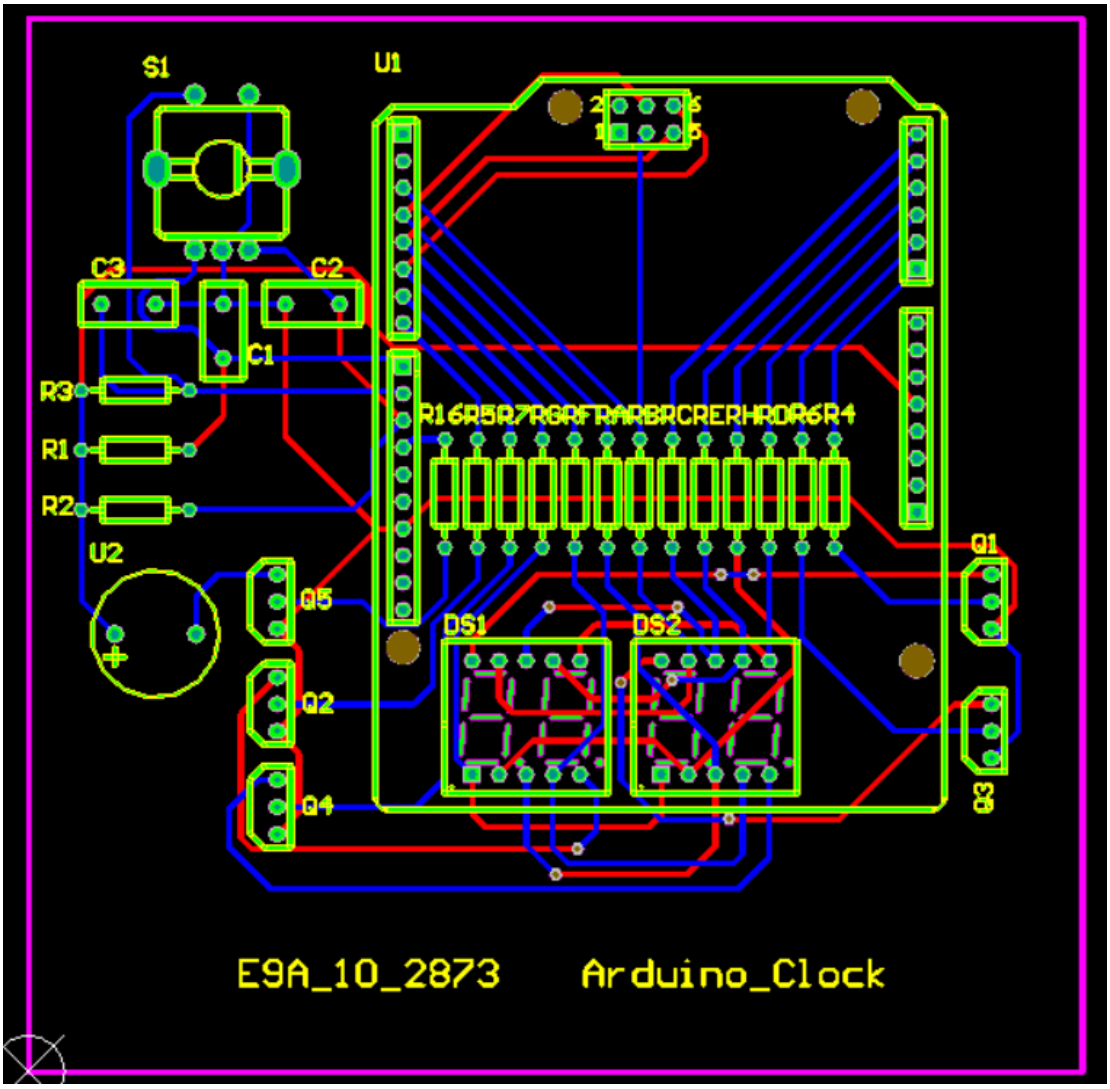
④旋转编码器模块

D9 端口连接旋转编码器的按钮引脚，用于检测按键动作，按下按钮即可在停止定时与定时运行两种状态之间切换。D8 和 D10 端口分别连接旋转编码器的 A、B 相信号，Arduino 代码通过检测两路电平变化的相位关系，判断旋转方向，从而在停止定时状态下改变所选的倒计时时间。旋转编码器未按下按钮时，D9 端口被上拉电阻拉至高电平；按下时接地变为低电平，代码以此检测按键动作。

（4）PCB 版图（本人设计）

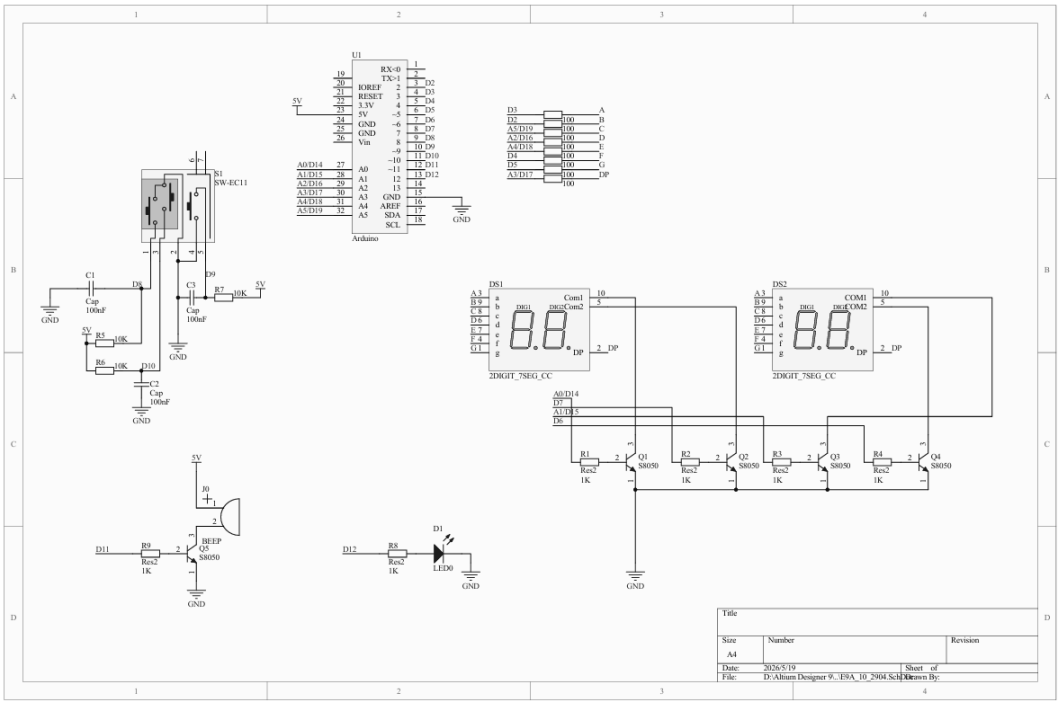
下图为本设计的 PCB 版图。整体布局紧凑合理，走线清晰流畅，充分考虑了模块化与可读性，体现了良好的硬件设计思路。本板的外形尺寸为 99×99mm，完全符合项目规定的尺寸要求。

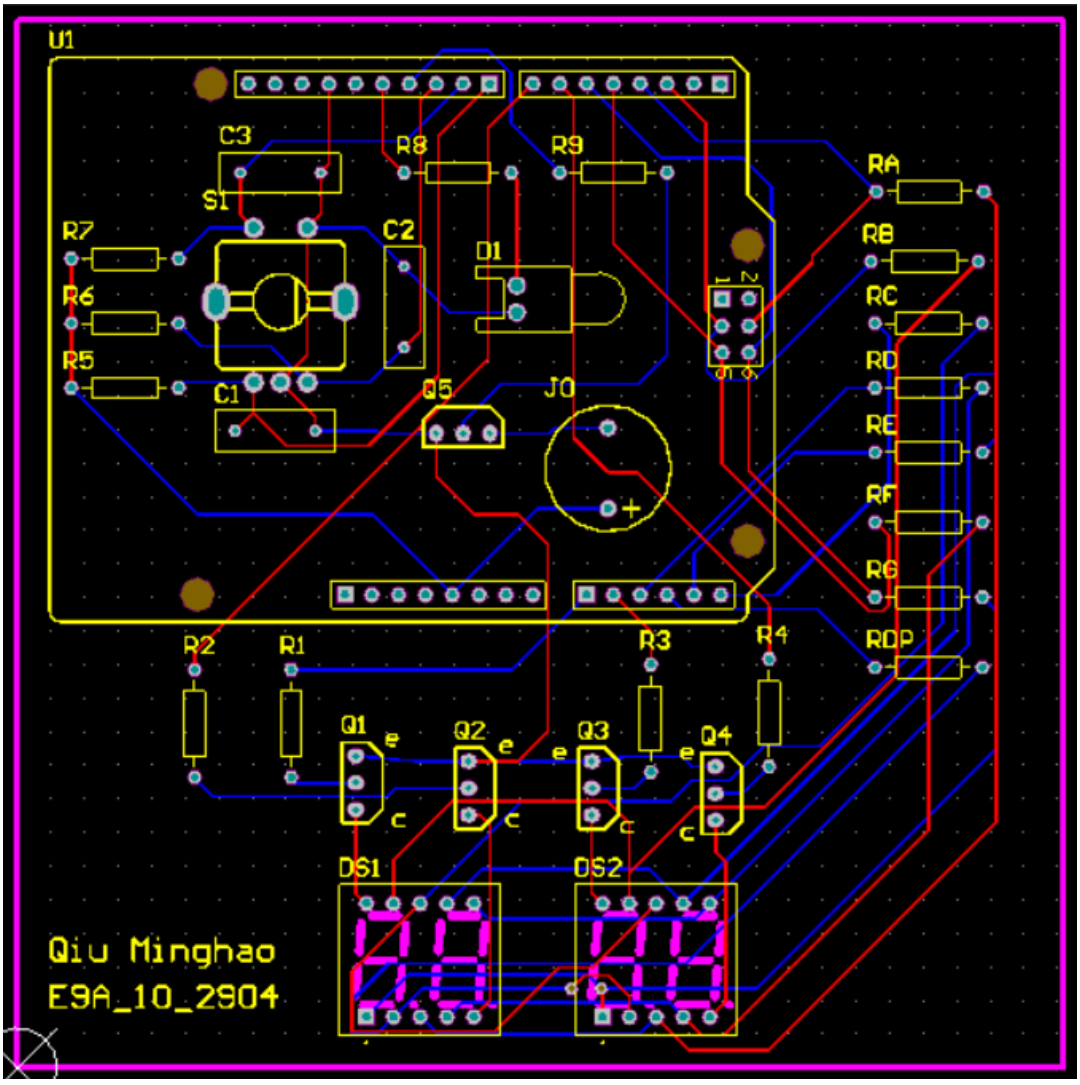
在布线过程中，为了优化走线路径、减少过孔数量并提高整体布线效率，我对不同模块的限流电阻位置进行了适当调整。这一改动使得实际布线更为顺畅、整齐，但需要特别留意：由于电阻位置与原始原理图并非一一对应，焊接时务必对照阻值标识进行安装，避免因错放电阻导致数码管段亮度异常或端口损坏。此外，为了进一步方便布线，数码管位置采用了双面布线的方法，提高了走线灵活性。



备注：本次设计项目实际参与制板与调试工作的 PCB 是队友设计的，上述介绍的设计则由个人完成，后面的代码部分是依照队友的设计编写的。

(5) 队友设计的原理图与 PCB 版图（队友设计）





（6）软件部分设计（本人设计）

本设计采用共阴极数码管配合 NPN 三极管位选驱动，并在功能上改为连续调秒的模式，代码在原始参考程序的基础上作出以下修改：

改动一：段码表（共阳极改为共阴极）

原始代码的段码表按共阳极数码管编写，共阴极数码管需反转电平。即共阳极数码管低电平点亮，共阴极数码管高电平点亮，因此需将段码表中的 0 和 1 全部互换。

```
// 改动 1：共阴极段码表（1=点亮，0=熄灭）
// 段码顺序：{a, b, c, d, e, f, g, dp}
byte digits[10][8] = {
  {1,1,1,1,1,1,0,0},  // 0：a,b,c,d,e,f 亮
  {0,1,1,0,0,0,0,0},  // 1：b,c 亮
  {1,1,0,1,1,0,1,0},  // 2：a,b,d,e,g 亮
  {1,1,1,1,0,0,1,0},  // 3：a,b,c,d,g 亮
  {0,1,1,0,0,1,1,0},  // 4：b,c,f,g 亮
  {1,0,1,1,0,1,1,0},  // 5：a,c,d,f,g 亮
  {1,0,1,1,1,1,1,0},  // 6：a,c,d,e,f,g 亮
  {1,1,1,0,0,0,0,0},  // 7：a,b,c 亮
  {1,1,1,1,1,1,1,0},  // 8：全部段亮
  {1,1,1,1,0,1,1,0}   // 9：a,b,c,d,f,g 亮
};
```

改动二：段选输出函数（去除取反操作）

原始代码为适配共阳极，在输出段码时进行了取反操作。共阴极数码管在段选引脚输出高电平时对应笔段点亮，因此直接输出段码值即可，无需逻辑取反。

```
// 改动 2：段选函数（共阴极：高电平点亮，无需取反）
void setSegments(int n) {
  for (int i = 0; i < 8; i++) {
    digitalWrite(segmentPins[i], digits[n][i]);  // 直接输出段码
  }
}
```

改动三：位选输出函数（电平反转）

原始代码位选低电平有效，本设计采用 NPN 型三极管驱动共阴极数码管，基极高电平时三极管导通、对应位数码管工作，因此位选信号需改为高电平有效。

```
// 改动 3：位选函数（共阴+NPN 三极管：高电平选通）
void setDigit(int digit) {
  for (int i = 0; i < 4; i++) {
    digitalWrite(displayPins[i], (digit == i));  // 只给选中的位送高电平
  }
}
```

改动四：时间设定方式（数组跳选改为连续调秒）

原始代码通过预设的标准时间数组进行步进选择。本设计改为连续调节方式，编码器每转一格加减 1 秒，分钟和秒自动进位/借位。将当前分、秒转换为总秒数，编码器每产生一个脉冲总秒数相应加减 1，然后再转换回分钟和秒。该方式可实现任意时间的精确设定，时间调节范围 0 秒至 99 分 59 秒。

```
// 改动 4：连续调节时间（以秒为单位增减）
void changeSetTime(int change) {
  int totalSeconds = timerMinute * 60 + timerSecond;  // 转换为总秒数
  totalSeconds += change;                              // 编码器转一格加减 1
秒
  if (totalSeconds < 0) totalSeconds = 0;              // 最小 0 秒
  if (totalSeconds > 5999) totalSeconds = 5999;        // 最大 99 分 59 秒
  timerMinute = totalSeconds / 60;                     // 转换回分钟
  timerSecond = totalSeconds % 60;                     // 转换回秒
}
```

改动五：EEPROM 存储方式（存储索引改为存储时间值）

原始代码在 EEPROM 中保存的是时间数组的索引值。连续秒调模式不再使用数组，改为直接存储分钟数和秒数。读取时增加合法性检查，防止 EEPROM 未初始化时出现 255 等非法数值。

```
// 改动 5：从 EEPROM 读取上次保存的时间（分钟和秒分别存储）
timerMinute = EEPROM.read(0);   // 地址 0 存储分钟数
timerSecond = EEPROM.read(1);   // 地址 1 存储秒数
// 防止读取到非法值（EEPROM 初始值可能是 255）
if (timerMinute > 99) timerMinute = 0;
if (timerSecond > 59) timerSecond = 0;
// 改动 5：将当前时间（分钟和秒）保存到 EEPROM
EEPROM.write(0, timerMinute);  // 地址 0 存储分钟
EEPROM.write(1, timerSecond);  // 地址 1 存储秒
```

（7）软件部分设计（队友设计）

这部分将结合队友设计的代码，说明我们组最终的设计项目能够实现的功能。

功能一：正常倒计时模式（MODE_NORMAL）

功能描述：从预设的时间列表中选择一个倒计时时长（如 5 秒、10 秒、1 分钟、5 分钟、30 分钟等）。旋转编码器调整时间档位，短按启动倒计时。倒计时运行时每秒更新显示，剩余 60 秒和最后 10 秒时会发出短促蜂鸣提示。倒计时结束进入报警状态，蜂鸣器周期性鸣叫，显示“donE”。长按编码器按键可随时返回模式选择菜单。

核心代码：

```
// 预设时间档位（单位：秒，格式为 M*100+SS）
const int timeCodes[] = {
    5, 10, 15, 20, 30,
    45, 100, 130, 200, 230,
    300, 400, 500, 600, 700,
    800, 900, 1000, 1500, 2000, 3000
};
// 启动倒计时（普通模式或 LOAD 模式）
void startTimer(unsigned long now) {
    currentState = STATE_RUNNING;
    pomodoroRest = false;
    alarmMuted = false;
    lastReminderSecond = -1;
    lastTick = now;
    remainingSeconds =
secondsFromTimeCode(timeCodes[selectedTimeIndex]);
    digitalWrite(loadPin, currentMode == MODE_LOAD ? HIGH : LOW);
    saveSettings();
}
// 每秒递减剩余时间
void updateCountdown(unsigned long now) {
    if (currentState != STATE_RUNNING) return;
    while (now - lastTick >= 1000 && currentState == STATE_RUNNING) {
        lastTick += 1000;
        if (remainingSeconds > 0) {
            remainingSeconds--;
            maybeReminderBeep(now);   // 60 秒或≤10 秒时短鸣
        }
        if (remainingSeconds <= 0) {
            remainingSeconds = 0;
            finishTimer(now);
        }
    }
}
// 倒计时结束处理
void finishTimer(unsigned long now) {
    digitalWrite(loadPin, LOW);
    if (currentMode == MODE_POMODORO && !pomodoroRest) {
        startPomodoroRest(now);   // 番茄工作结束自动休息
        return;
    }
    currentState = STATE_ALARM;
    alarmMuted = false;
```

```
    showTemporaryText("donE", 1000, now);
}
```

功能二：番茄钟模式（MODE_POMODORO）

功能描述：进入模式后短按直接开始一个 25 分钟的工作计时（无需选择时间档位）。显示临时提示“P-25”1.2 秒，然后倒计时。工作计时结束后自动切换为 5 分钟休息计时，显示“rESt”2 秒并短鸣一声。休息计时结束后进入报警状态，显示“donE”并周期鸣叫。任何长按可退出番茄模式返回主菜单。

核心代码：

```
// 启动番茄工作（25 分钟）
void startPomodoroWork(unsigned long now) {
    currentState = STATE_RUNNING;
    currentMode = MODE_POMODORO;
    pomodoroRest = false;
    alarmMuted = false;
    lastReminderSecond = -1;
    lastTick = now;
    remainingSeconds = 25 * 60;          // 25 分钟 = 1500 秒
    digitalWrite(loadPin, LOW);
    showTemporaryText("P-25", 1200, now);
    saveSettings();
}
// 启动番茄休息（5 分钟）
void startPomodoroRest(unsigned long now) {
    currentState = STATE_RUNNING;
    pomodoroRest = true;
    alarmMuted = false;
    lastReminderSecond = -1;
    lastTick = now;
    remainingSeconds = 5 * 60;           // 5 分钟 = 300 秒
    digitalWrite(loadPin, LOW);
    startBeep(now, 250);                 // 短鸣 250ms
    showTemporaryText("rESt", 2000, now);
}
// 在 finishTimer() 中自动切换工作→休息
void finishTimer(unsigned long now) {
    digitalWrite(loadPin, LOW);
    if (currentMode == MODE_POMODORO && !pomodoroRest) {
        startPomodoroRest(now);   // 工作结束 → 自动开始休息
        return;
    }
    // 普通模式或休息结束 → 报警
```

```
    currentState = STATE_ALARM;
    alarmMuted = false;
    showTemporaryText("donE", 1000, now);
}
```

功能三：LOAD 负载模式（MODE_LOAD）

功能描述：该模式下，倒计时运行时，引脚 loadPin（12 号）输出高电平，可用于驱动外部继电器或负载（本次实验中采用 LED 呈现功能）。倒计时结束、暂停、返回模式选择时，loadPin 恢复低电平。负载模式与普通倒计时共用时间档位和倒计时逻辑，仅增加负载控制功能。适合需要定时关闭或开启外部设备（如灯光、电机、加热器等）的应用场景。

核心代码：

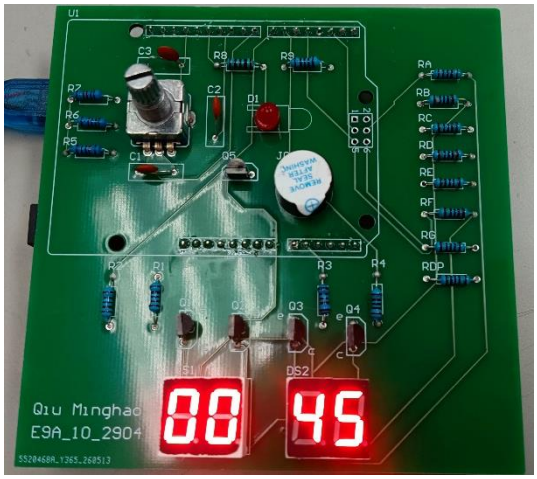
```
// 启动倒计时时根据模式设置负载引脚
void startTimer(unsigned long now) {
    currentState = STATE_RUNNING;
    // ... 其他初始化
    digitalWrite(loadPin, currentMode == MODE_LOAD ? HIGH : LOW);
    // 仅当 MODE_LOAD 时 loadPin = HIGH，其余模式 LOW
}
// 暂停倒计时时关闭负载
void pauseTimer() {
    currentState = STATE_PAUSED;
    digitalWrite(loadPin, LOW);
    digitalWrite(buzzerPin, LOW);
    beepUntil = 0;
}
// 倒计时结束或返回主菜单时关闭负载
void finishTimer(unsigned long now) {
    digitalWrite(loadPin, LOW);
    // ... 后续处理
}
void enterModeSelect() {
    currentState = STATE_MODE_SELECT;
    digitalWrite(loadPin, LOW);   // 退出任何运行状态时确保负载关闭
}
```

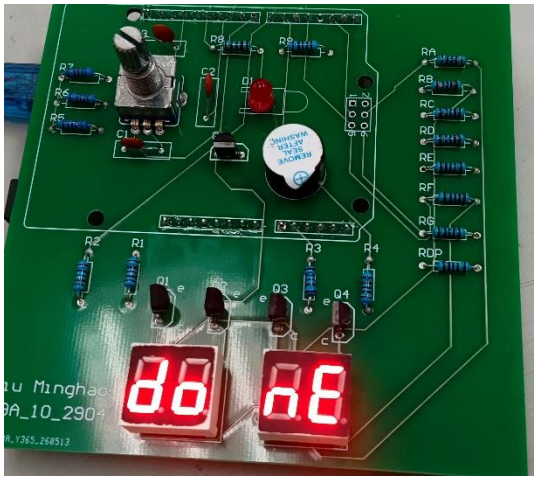
五、实验结果记录

（1）正常倒计时模式功能呈现：

上电后默认进入正常倒计时模式，数码管显示当前设定的时间。旋转编码器可在预设时间数组中选择倒计时时长，短按编码器按键启动倒计时。运行期间数码管动态

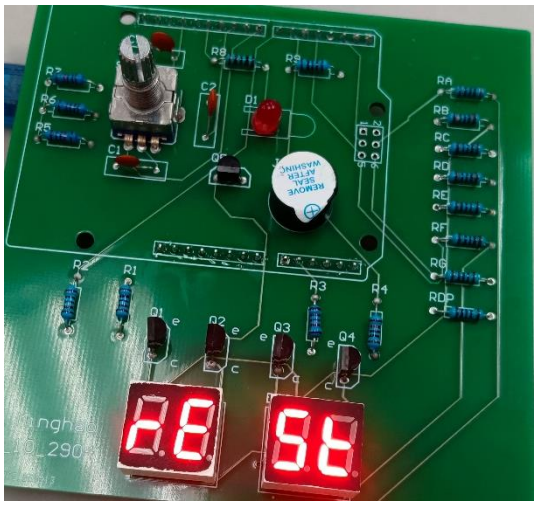
显示剩余时间；当剩余 60 秒及最后 10 秒时，蜂鸣器发出短促提示音。倒计时归零后，蜂鸣器进入周期性报警模式，数码管显示“donE”。长按编码器按键可随时返回模式选择主菜单。经实测，时间显示准确，蜂鸣提示清晰，按键响应可靠，各预设档位均能正常工作。下图为正常倒计时模式运行及报警状态的实物照片。

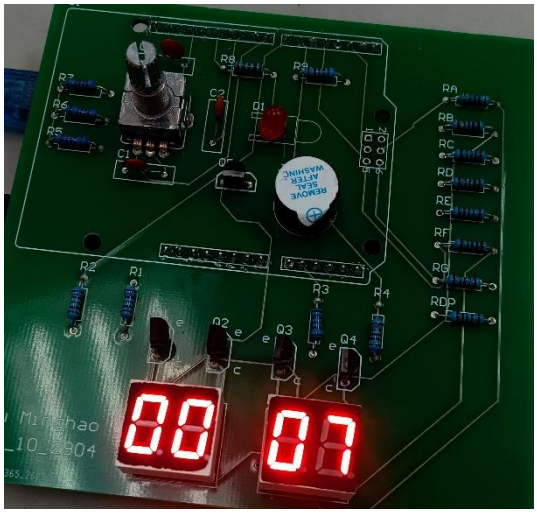
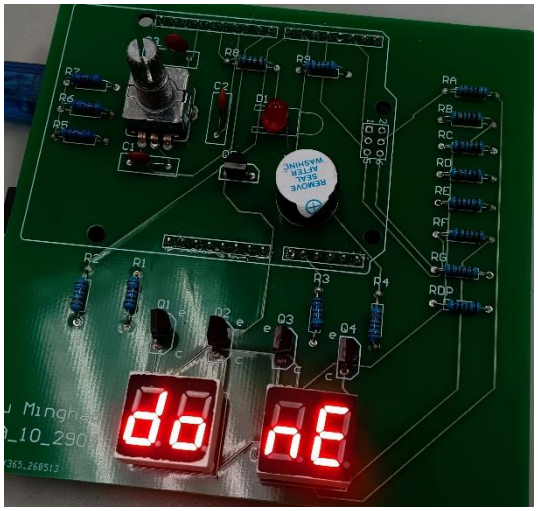




（2）番茄钟模式功能呈现：

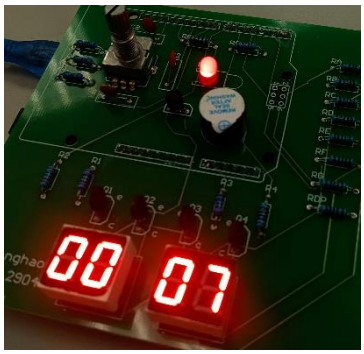
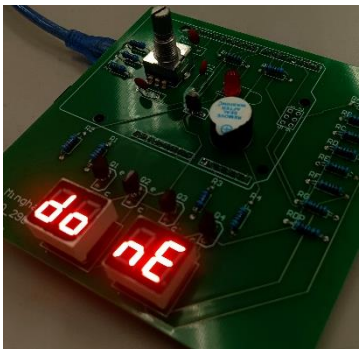
通过旋转编码器将模式切换至“P-25”（番茄工作模式），短按按键后数码管短暂显示“P-25”提示，随即开始 25 分钟工作倒计时。工作计时结束时，蜂鸣器短鸣 0.25 秒，数码管显示“rESt”2 秒，随后自动切换为 5 分钟休息倒计时。休息计时结束，蜂鸣器周期报警，数码管显示“donE”。由于默认设定的番茄钟倒计时时间太长，调试过程中我们缩短了计时时间。实验过程中，工作→休息→报警的状态切换完全自动完成，无需人工干预。长按编码器按键可随时退出番茄模式。下图为番茄工作模式启动、工作倒计时、休息倒计时及报警状态的实物记录。



（3）LOAD 负载模式功能呈现

切换至“HEAt”（负载模式）后，短按按键进入时间选择界面，旋转编码器设定所需倒计时时长，再次短按启动倒计时。倒计时运行期间，Arduino UNO 的 D12（loadPin）引脚输出高电平，驱动扩展板上的 LED 指示灯点亮，模拟外部负载（继电器、加热器等）工作。倒计时结束、按暂停或长按返回主菜单时，LED 立即熄灭。实验证明，负载输出与倒计时状态严格同步，负载控制逻辑正确，可用于实际设备定时通断控制。下图为 LOAD 模式倒计时运行时 LED 点亮及倒计时结束时 LED 熄灭的对比照片。

六、总结与体会

通过本次倒计时定时器的设计与实现，我在硬件电路设计、嵌入式编程、PCB 设计与调试等多个方面获得了宝贵的实践经验。本次实验涵盖了从功能需求分析、元器件选型、电路原理图设计、PCB 布局布线，到 Arduino 程序编写与系统联调的全过程。通过亲手设计扩展板、焊接调试、编写控制程序，我加深了对嵌入式系统开发流程的理解，尤其是如何将理论知识转化为实际可运行的硬件系统。

在 PCB 设计过程中，我学会了根据引脚物理位置就近分配信号，优化走线路径，减少过孔和交叉。虽然最终制板工作由队友完成，但我在自己的设计方案中也进行了完整的布局尝试，体会到合理布局对焊接、调试和系统稳定性的重要性。旋转编码器的使用是本项目的一大亮点。通过检测 A、B 两相的电平变化，我学会了如何判断旋转方向并实现时间的连续调节。同时，数码管的动态扫描显示也让我深入理解了多路复用显示的原理，掌握了如何用有限的 I/O 口驱动多个显示单元。在适配共阴极数码管的过程中，我重新设计了段码表和位选逻辑，明确了高电平点亮、低电平熄灭的控制方式，并通过 NPN 三极管实现位选导通。这一过程让我对数字电路中的电平匹配、驱动能力、限流保护等实际问题有了更直观的认识。程序编写过程中，我从共阳数码管的参考代码出发，逐步修改为适配共阴极的逻辑，并实现了连续调秒、EEPROM 断电保存、状态切换等功能。尤其是在时间调节部分，采用“总秒数”中间变量的方式简化了分钟和秒的进位借位逻辑，提升了代码的可读性和可维护性。

在系统联调阶段，调试过程整体非常顺利。 将烧录好程序的 Arduino UNO 与焊接完成的扩展板组装后，上电测试，各模块功能均一次通过：数码管显示清晰、旋转编码器响应准确、按键长短按识别可靠、蜂鸣器报警正常、负载输出控制无误。唯一遇到的小问题是偶尔出现供电不稳定的现象，表现为数码管亮度闪烁、程序异常复位。经排查，原因是 USB 供电线内部接触不良，导致电压波动。更换一根质量较好的 USB 线后，供电恢复稳定，整个定时器长时间运行正常，未再出现任何异常。这一经历让我认识到，在嵌入式系统调试中，除了关注电路设计和代码逻辑，供电质量同样至关重要，稳定的电源是系统可靠运行的基础。

通过本次实验，我不仅掌握了倒计时定时器的实现方法，也对嵌入式系统设计产生了更浓厚的兴趣。未来我希望进一步学习更多传感器与执行器的使用，探索更复杂的控制系统设计，并提升 PCB 设计的美观性与抗干扰能力。

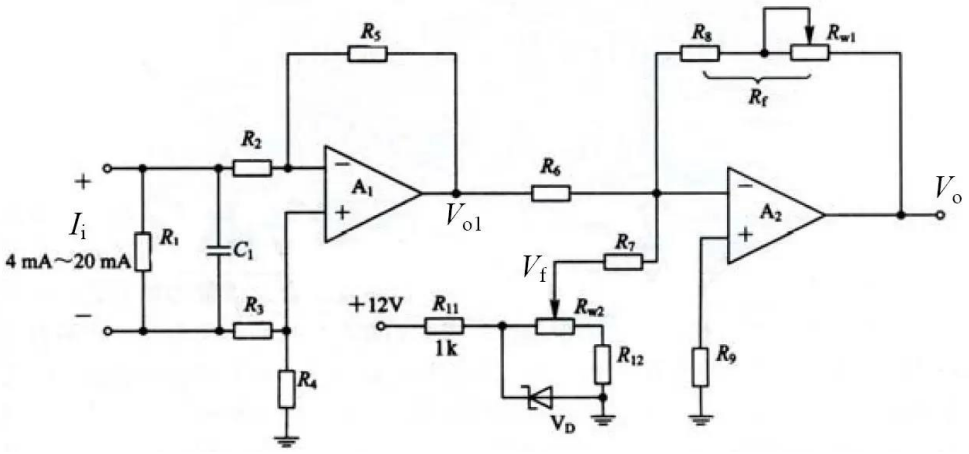
电压电流转换器设计

一、设计任务与要求

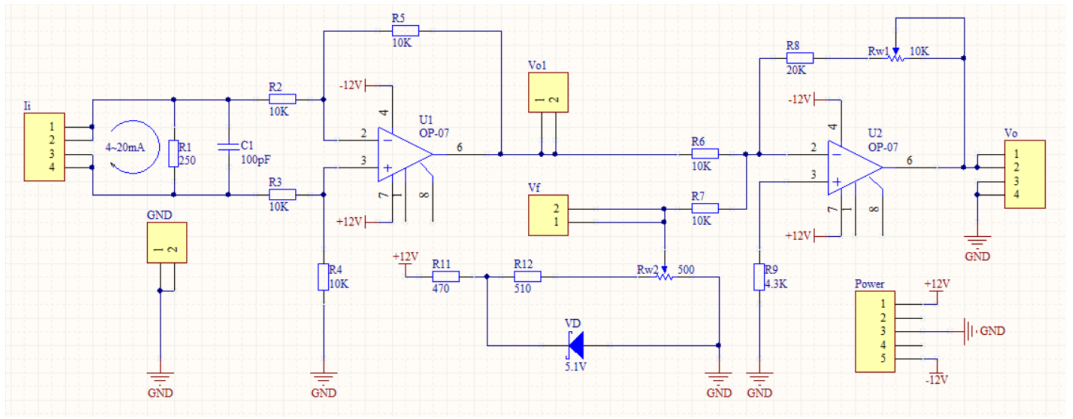
设计电流电压转换电路，将 4~20mA 标准电流信号转换为 0~10V 的标准电压信号。

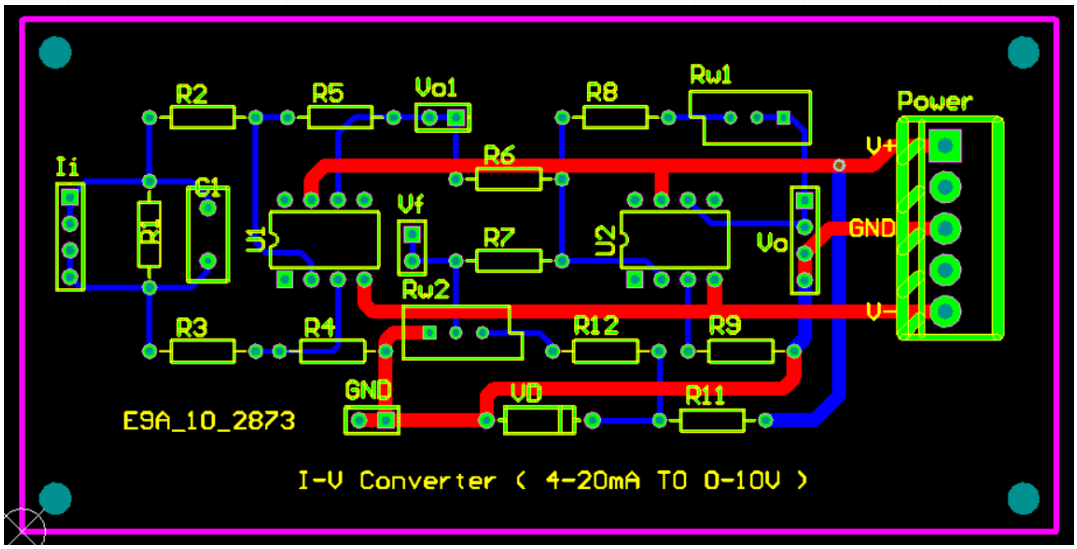
要求：1、参考如下电流-电压转换电路原理图，确定各元件值。

2、用 Altium Designer 软件设计原理图和 PCB 图。



二、原理图与 PCB





三、电路板调试与实验数据

电流/mA	实测电压/V	理论电压/V
4	0.00	0.0
8	2.49	2.5
12	5.01	5.0
16	7.48	7.5
20	10.02	10.0

实测电压与理论电压十分吻合，电路板调试成功。

四、总结与体会

本次电压电流转换器实验是本课程的前期基础训练，主要目的是帮助我们熟悉 Altium Designer 软件的操作流程，掌握原理图绘制、PCB 布局布线以及电路板焊接调试的基本方法，为后续 Arduino UNO 扩展板的完整设计奠定实践基础。

本实验采用典型的反相比例放大电路结构，将 4~20mA 电流信号经采样电阻转换为电压后，再通过运放调理至 0~10V 输出范围。我学会了根据输入输出关系计算反馈电阻值，并理解了运放在信号调理中的关键作用。

在 Altium Designer 中，我练习了元件库的调用、原理图连线、网络标签的使用，以及 PCB 中元件的合理布局、手工布线和规则检查。尤其是单面板布线时，如何避免飞线、减少跳线，提高了我的空间规划能力。焊接完成后，通过逐级测试输入输出关系，实测数据与理论值高度吻合，验证了电路设计的正确性。这一过程让我学会了使用万用表进行故障排查，例如检查虚焊、电源短路或运放供电异常等问题。

本实验虽然功能简单，但涵盖了一个完整电子项目的基本环节。在后续的倒计时定时器设计中，我能够更加熟练地进行原理图绘制、PCB 布局以及程序调试，正是因为有了这次前置训练的铺垫。总之，电压电流转换器实验虽然规模小，但让我初步建立了硬件开发的整体概念，增强了动手信心，为顺利完成 Arduino 扩展板项目做好了充分准备。