# 电子电路设计实验 II 指南

叶险峰 施红军 邓靖靖 编著

二〇二五年二月

# 目录

# 01 函数信号发生器的设计、组装与调试研究

## 一、实验原理和电路设计

### 方波

采用 Arduino 产生简单的波形为方波，方波是高低逻辑电平之间简单循环。

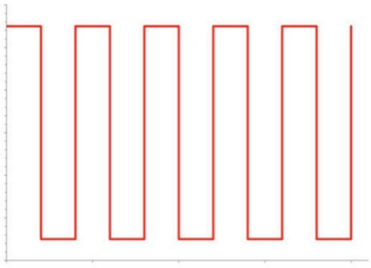

图 1.1 方波示意图

我们用来产生方波的电路几乎是可以想象的最平常的电路。

事实上，我们需要做的只是连接 Arduino 的一个数字输出引脚（这里用 8 脚）和地。代码也很简单：

```
void setup()
{
    pinMode(8，OUTPUT)；
}
void loop()
{
   // A total delay of 4 ms（frequency 250Hz）
    digitalWrite(8，HIGH)；
    delay(2)；
    digitalWrite(8，LOW)；
    delay(2)；
}
```

我们还可以使用 delaymicroseconds（）产生较小的延迟值。

事实上，Arduino 具有比这更容易产生一个方波的函数。例如，产生我们之前 250Hz 方波，我们可以使用：tone（8，250）代替 digitalWrite 语句。

让我们从在示波器上，看看这个方波到底是什么样子:

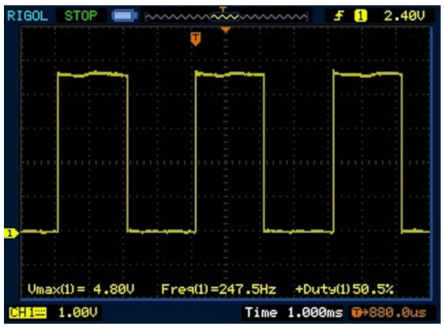


图 1.2 方波波形（占空比 50%）

除了波的频率，我们可能要修改一对其他主要特征参数：振幅（相对低电平，高电平有多高）和波的对称性，即所谓的占空比。

占空比只是测量高电平时间占总周期时间的比例。若方波是对称的：高低电平两种状态下都有相同的时间，即占空比为 50%。

如果我们要改变占空比，我们不能用 tone()功能的设计。我们需要使用 digitalwrite()和 delay()功能。例如，产生一个占空比为 25%的 250Hz 方波，在高电平后延迟 1 毫秒，然后低电平后延迟后 3ms。

使用示波器，我们就能看到波形的形状。

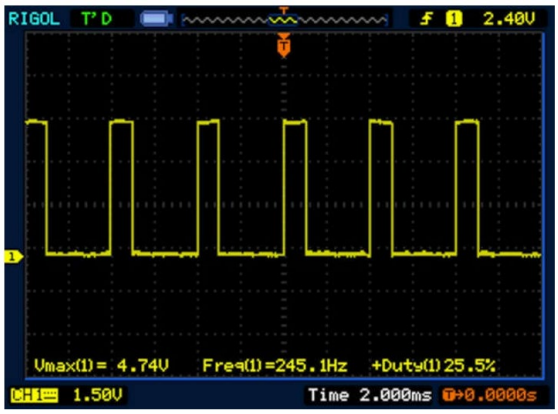


图 1.3 方波波形 （占空比 25%）

时间坐标每格 2ms，我们可以清楚地看到脉冲宽度 1ms，脉冲之间的间隙为 3ms。如万用表测量电压，一个 50%占空比的 250Hz 信号，平均值为 2.4V（正是 4.8 V 的一半，Arduino +5V 输出，测得的是 4.8V）。25%占空比我们得到 1.2V（四分之一高电平）。

## 三角波

如何产生一个三角形波形，可以采用某种数字模拟转换器（或 DAC）。

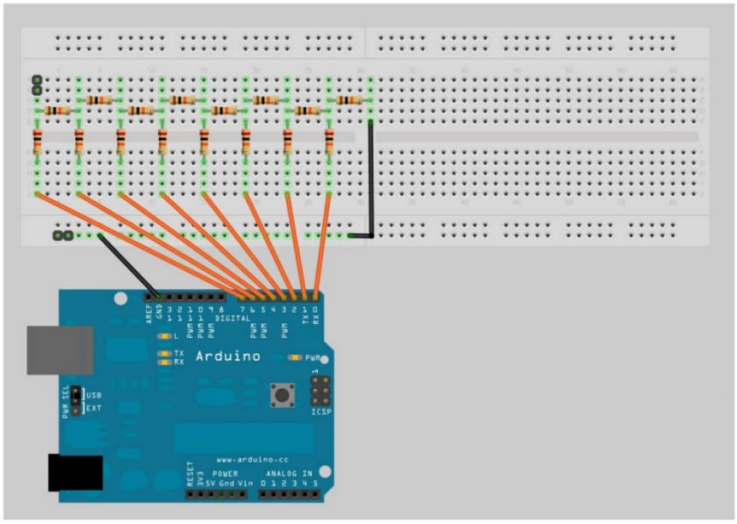


图 1.4 R-2R 电阻梯形网络

这是一个 8 位 DAC 电路称为 R-2R 电阻梯形网络。八位中的每一位对输出电压都有贡献。如果 8 位都为高电平，则输出约等于参考电压。如果最高位为低电平，就得到了大约参考电压一半。更确切地说，一个 8 位 DAC 如果 8 位都为高平，则得到（255 / 256）*参考电

压，最高位为低电平，我们得到（127 / 256）*参考电压。一般地， DAC 输出为（x/256）*参考电压，x 为 0 和 255 之间的值。

产生三角波形程序为：

```
void setup()
{
      pinMode (0, OUTPUT);
      pinMode (1, OUTPUT);
      pinMode (2, OUTPUT);
      pinMode (3, OUTPUT);
      pinMode (4, OUTPUT);
      pinMode (5, OUTPUT);
      pinMode (6, OUTPUT);
      pinMode (7, OUTPUT);
}
void loop()
{
       for (int i=0; i<255; i++)
      {
          PORTD=i ;
      }
         for(int i=255；i>0；i--)
      {
          PORTD=i ;
      }
}
```

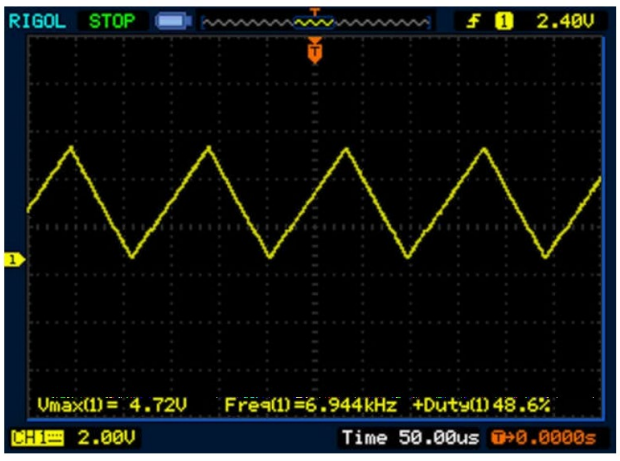


图 1.5 三角波波形

注意，这里我们使用 PORTD 代替单个输出引脚同时设置状态。这更快，并且确保所有的引脚在同一时间改变，这对产生平滑的模拟电压十分重要。

PORTD 是寄存器 D 端口，在这里其作用，是因为它映射到 0-7 数字引脚（8 位）；在一个操作，我们控制所有 8 引脚将二进制或十进制值的写入寄存器。例如，如果我们赋十进制

值 123（这相当于 B01111011），将置 6，5，4，3，1，0 为高电平。

虽然引脚 0、1 是用于串行收发，但使用 PORTD 可以同时操作 8 个输出引脚不影响串行通信。

另外，我们可以非常容易地修改我们的代码，通过简单地注释（或者删除）loop 中的第二个 for 语句来产生锯齿波形。

## 正弦波

对于正弦波。我们需要预先计算随着时间推移的输出电压值（Arduino 不足以快到实时获取它们）。

正弦波一个周期为 2π 弧度——所以为了产生一个每个周期 256 时间步长的正弦波，对每个 x 值，计算 y＝sin（（x／255）* 2π）。因为正弦函数在 1 和-1 间取值，而我们想要 0 和 255 之间的值，最简单的方法就是将浮点值乘以 128——给出一个-128 到 128 间的值，然后加上 128 从而数值落在正确的范围内。

在 setup()阶段，在全局数组存储结果。那么在 loop 中，我们需要做的是通过一次数组循环。程序代码如下：

```
#define PI 3.14
int sine[255];
void setup()
{
       pinMode (0，OUTPUT);
       pinMode(1，OUTPUT);
       pinMode(2，OUTPUT);
       pinMode(3，OUTPUT);
       pinMode(4，OUTPUT);
       pinMode(5，OUTPUT);
       pinMode(6，OUTPUT);
       pinMode(7，OUTPUT);
       float    x;
       float    y;
       for(int i=0；i<255；i++)
       {
            x = (float) i；
            y =sin((x ／ 255)*2*PI);
            sine[i]=int (y*128)+128;
       }
}
void loop()
{
     for(int i=0;i<255;i++)
     {
            PORTD=sine[ i ];
            delayMicroseconds(10);
     }
```

}

如果我们运行这个代码，并用 DAC(如 R-2R 电阻梯形网络)，在示波器上我们就会发现波形确实非常像正弦波。

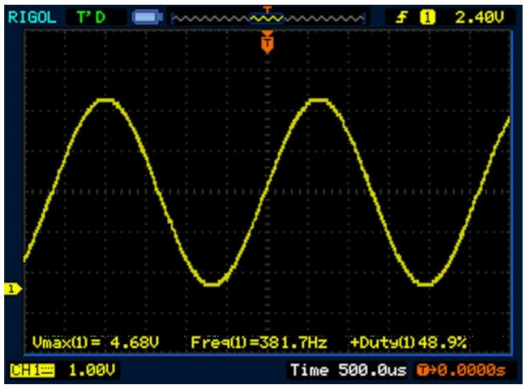


图 1.6 正弦波波形

## 二、实验任务与要求

1、用 Arduino UNO 设计一个函数信号发生器，要求如下：
（ 1）输出可在方波、三角波、正弦波间切换。
（ 2）输出频率自定。
（ 3）输出幅度 0.5~2V 可调（要求加输出缓冲）。
2、设计的电路，调整元件参数，直至满足任务要求。重点考虑：
（1）如何切换波形（硬件、软件方式）；
（2）DAC 方法（不限于 R-2R 电阻梯形网络）；
（3）单电源运放缓冲设计。
3、绘制程序流程图，编制与调试函数信号发生器程序。
4、制作 Arduino 扩展板，并将其与 UNO 板组装联调。

## 三、思考题

1、如何改变方波的占空比？
2、试分析简单的函数信号发生器中影响波形频率的因素。
3、简述 R-2R 电阻梯形网络实现 DAC 的原理。
4、若 D0 和 D1 用于串口通信，设计时避开这两个端口，应该如何设计？

## 四、设计举例

### 1、电路原理图

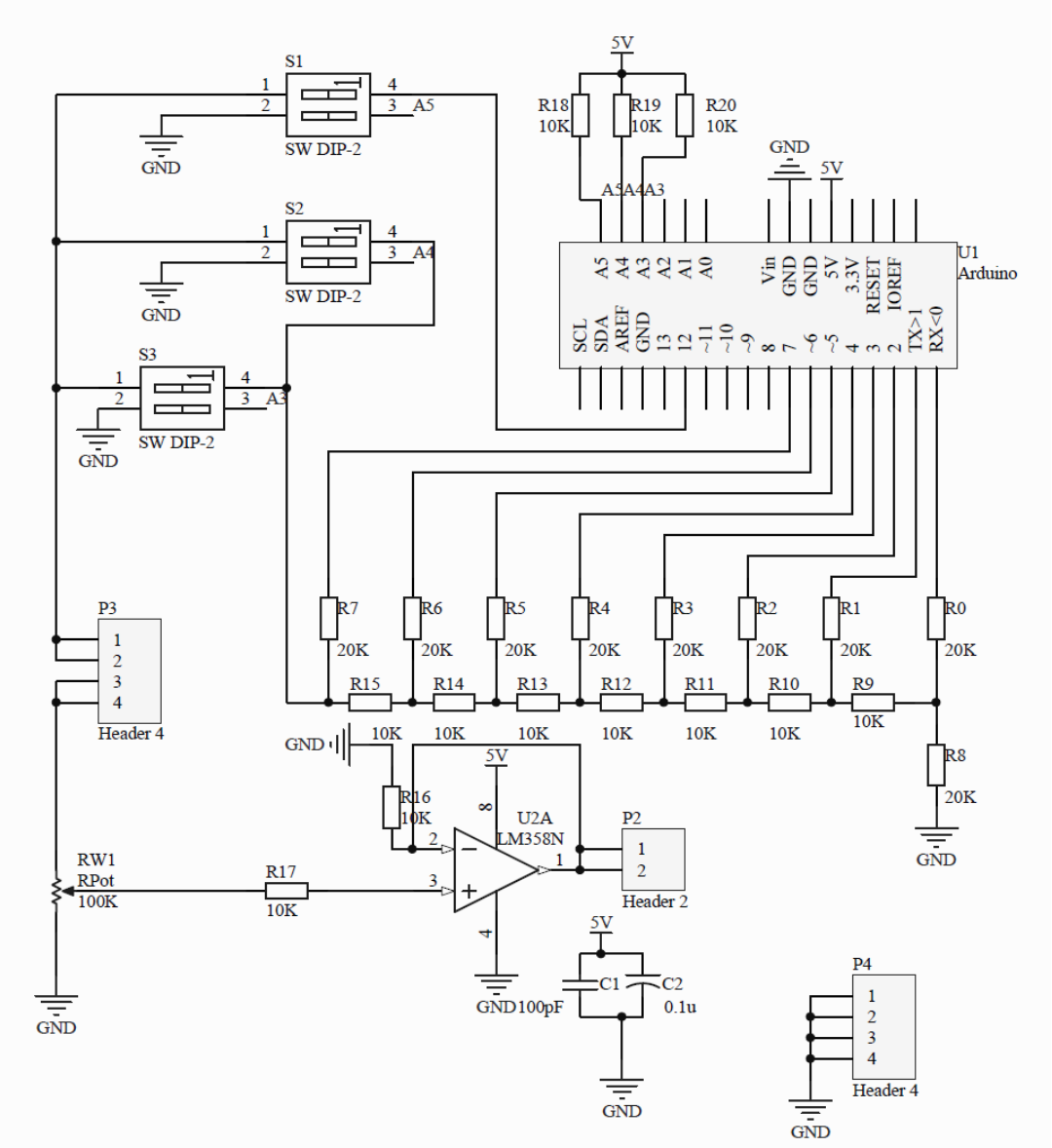


图 1.7 电路原理图

## 2、代码

```
#include <math.h>
#include <Arduino.h>

// 定义 DAC 连接引脚
#define DATA_PIN_START 0   // D0 开始
#define NUM_DATA_PINS 8    // 一共 8 个数据引脚
#define PI 3.14
int sine[255];
// 定义波形类型
enum WaveformType {
  SQUARE,
  SINE,
```

```
  TRIANGLE
};

// 读取引脚状态并返回当前波形类型
WaveformType readWaveformFromPins() {
  if (digitalRead(A5) == LOW & digitalRead(A4) == HIGH & digitalRead(A3) == HIGH) {
    return SQUARE;
  } else if (digitalRead(A4) == LOW & digitalRead(A3) == HIGH & digitalRead(A5) == HIGH)
{
    return SINE;
  } else if (digitalRead(A3) == LOW & digitalRead(A4) == HIGH & digitalRead(A5) == HIGH)
{
    return TRIANGLE;
  } else {
    // 默认情况下，我们可以选择一个波形或者保持一个默认状态
    return SQUARE; // 或者其他你希望的默认波形
  }
}
 void generateSquareWave() {
  // 方波：高低电平交替
  // A total delay of 4 ms（frequency 250Hz）
    digitalWrite(12,HIGH);
    delay(2);
    digitalWrite(12,LOW);
    delay(2);
}
 void generateSineWave() {
  // 正弦波：sine[]
   for(int i=0;i<255;i++)
    {
        PORTD=sine[i];
      delayMicroseconds(50);
    }
}

void generateTriangleWave() {
  // 三角波：先线性增加然后线性减少
 for (int i=0; i<255; i++)
    {
      PORTD=i ;
      delayMicroseconds(50);
    }
 for(int i=255;i>0;i--)
    {
```

```
        PORTD=i ;
        delayMicroseconds(50);
      }

}

void setup() {
  // 设置数据引脚为输出
  for (int i = DATA_PIN_START; i < DATA_PIN_START + NUM_DATA_PINS; i++) {
    pinMode(i, OUTPUT);
  }
  pinMode(12, OUTPUT);
  // 设置 A3, A4, A5 为输入模式
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);

  // sin 函数计算 sine[]
   float   x;
   float   y;
   for(int i=0;i<255;i++){
          x = (float) i;
          y =sin((x/255)*2*PI);
          sine[i]=int(y*128)+128;
   }
}

void loop() {
  // 读取引脚状态并确定当前波形类型
  WaveformType currentWaveform = readWaveformFromPins();
  // 根据当前波形类型生成波形
  switch (currentWaveform) {
    case SQUARE:
      generateSquareWave();
      break;
    case SINE:
      generateSineWave();
      break;
    case TRIANGLE:
      generateTriangleWave();
      break;
  }
}
```

# 02 Arduino 万用表的设计与制作

## 一、实验设计方案

本项目使用 Arduino UNO 设计万用表。硬件框图如图 1 所示，下面对各部分模块进行说明。

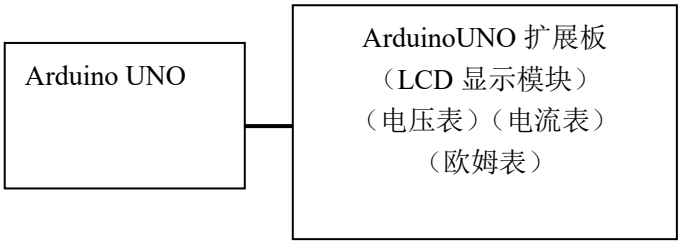


图 2.1　Arduino 万用表框图

## 二、硬件设计

万用表由电压表、电流表、欧姆表等功能模块组成，下面就各个部分进行说明：

1、 电压表的简化图。三个量程可以通过 Arduino 扩展板加按钮进行选择。在进行电压测量时，只有一个开关是闭合的。

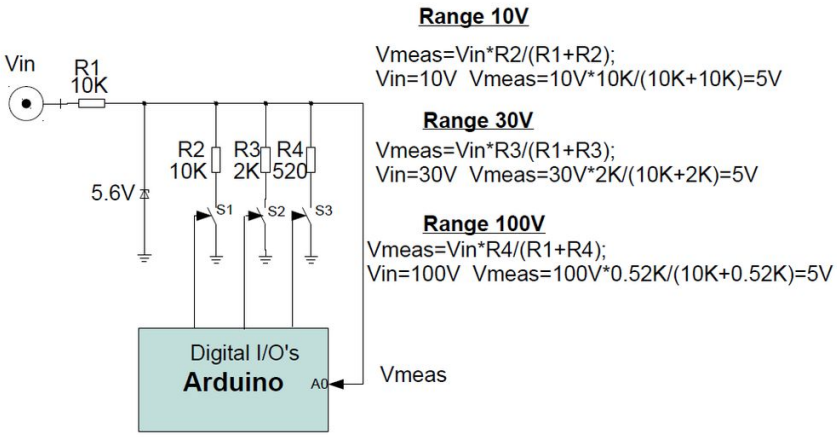


图 2.2　电压表电路简图

2、电流表的简化图。被测量电流流过 1Ω 电阻到地，其输出经放大后连接到 Arduino 的 A1 接口，放大器的增益为 10。为了进行过流保护，配置了一个 500mA 的复位 PTC。

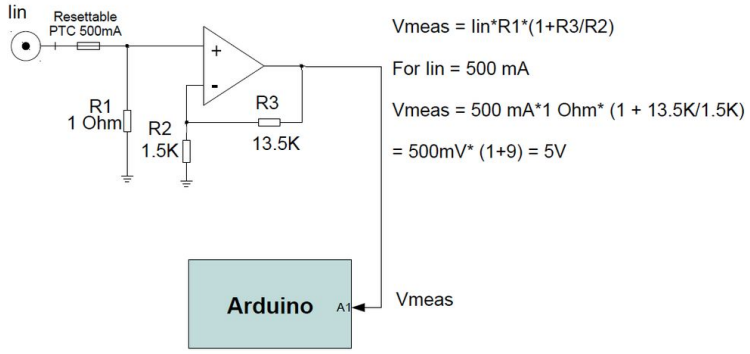


图 2.3 电流表电路简图

3、欧姆表的简化图。由齐纳二极管相对于正电压源产生基准电压，该电压被施加到由晶体管和运算放大器组成的电流转换器上，作为电压输入，晶体管的射极接可变电阻。此时齐纳二极管上的电压和该电阻上的电压相等。

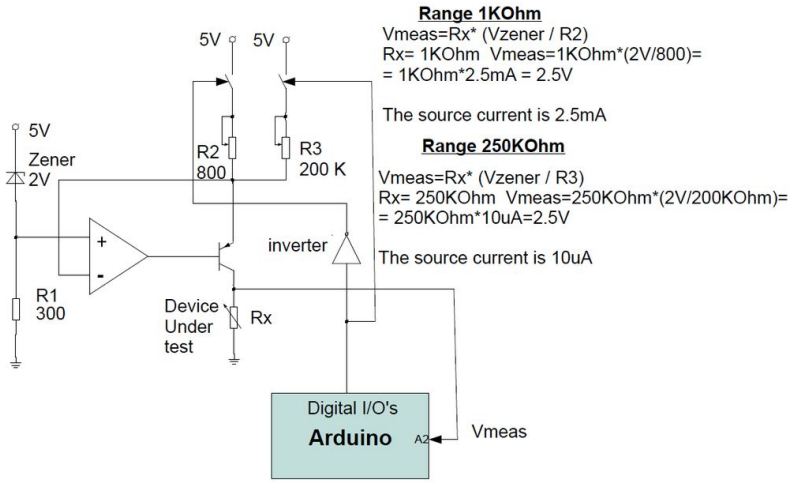


图 2.4 欧姆表的简图

“Arduino”板控制两个开关中的哪一个闭合，以这种方式决定两个电阻中的哪一个将流过电流。因此，两个可能的电流值是：10μA，2.5mA。这些电流可以非常精确地调节。这样产生的电流通过被测器件(例如电阻)，在被测器件(DUT)上电压降由“Arduino”板的模拟引脚 A2 检测。

## 三、实验任务与要求

1、用 Arduino Uno 板设计万用表，测量数据可以用 LCD 显示模块显示。该万用表应具备以下功能：

- 具备 3 个量程的电压表：0-10V，0-30V，0-100V
- 具备 1 个量程的电流表：0-500mA
- 具备 2 个量程的欧姆表：0-1kΩ，0-250kΩ

2、设计的电路，完成相应器件的选择和参数计算，制作 Arduino UNO 扩展板。

3、编制与调试万用表的程序。

4、将制作的 Arduino UNO 扩展板与 Arduino UNO 板组装后，进行系统联调。
5、用一个标准的万用表，对组装的万用表进行校准。

## 四、设计举例

1、电路原理图

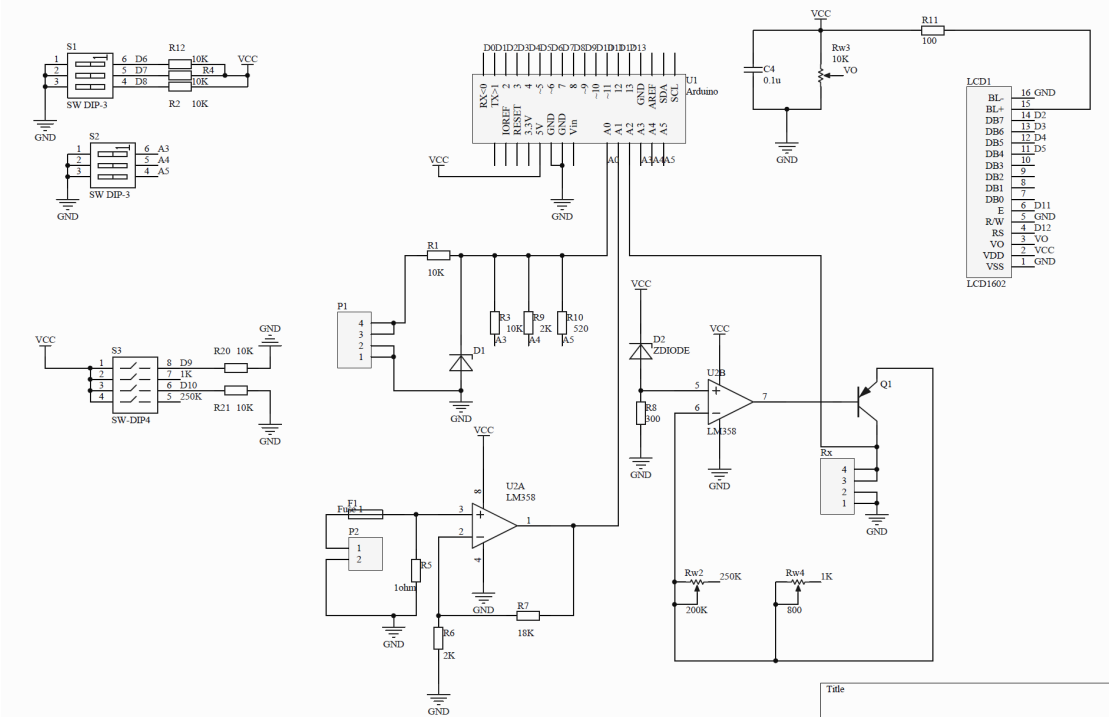


图 2.5 电路原理图

2、代码

```
#include <LiquidCrystal.h>  //声明 1602 液晶的函数库
// 初始化 LCD
LiquidCrystal lcd(12,11,5,4,3,2);  //4 数据口模式连线声明
const int V_Pin = 6;  // 电压表开关
const int VoltagePin = A0;// 电压引脚
const int I_Pin = 7;  // 电流表开关
const int currentPin = A1;// 定义电流引脚
const int R_Pin = 8;  // 欧姆表开关
// 定义电阻引脚
const int resistorPin = A2;
float   resistorFactor1 = 1.95;       // 电阻系数 1
float   resistorFactor2 = 0.489;       // 电阻系数 2
const int V1_Pin = A3;  // 电压量程开关 V1
float votageFactor1=2.0;
const int V2_Pin = A4;  // 电压量程开关 V2
float votageFactor2=6.0;
```

```
const int V3_Pin = A5;   // 电压量程开关 V3
float votageFactor3=20.0;
const int R1_Pin = 9;   // 欧姆表量程开关 1
const int R2_Pin = 10;   // 欧姆表量程开关 2
int reading;
void setup() {
  Serial.begin(9600);   // 设置引脚模式为输入
  pinMode(V_Pin, INPUT);
  pinMode(I_Pin, INPUT);
  pinMode(R_Pin, INPUT);

  pinMode(V1_Pin, INPUT);
  pinMode(V2_Pin, INPUT);
  pinMode(V3_Pin, INPUT);

  lcd.begin(16,2); //初始化 LCD 显示器
  lcd.clear();
}

void loop() {
  // 读取开关状态
  int V_State = digitalRead(V_Pin);
  int I_State = digitalRead(I_Pin);
  int R_State = digitalRead(R_Pin);
  // 读取电压档位
  int V1_L = digitalRead(V1_Pin);
  int V2_L= digitalRead(V2_Pin);
  int V3_L = digitalRead(V3_Pin);
   // 读取电阻档位
  int R1_L = digitalRead(R1_Pin);
  int R2_L= digitalRead(R2_Pin);
        // 检测开关状态并输出对应状态信息
     if (V_State == LOW && I_State == HIGH && R_State == HIGH ) {

    // 读取电压值
      if(V1_L==LOW) {
      reading = analogRead(VoltagePin) * (5.0 / 1023.0)*votageFactor1; // 0-10V
      lcd.setCursor(0, 1);
      lcd.print("Voltage: ");
      lcd.print(reading);
      lcd.print(" V");
      }
      else if(V2_L==LOW) {
      reading = analogRead(VoltagePin) * (5.0 / 1023.0)*votageFactor2; // 0-30V
```

```
  lcd.setCursor(0, 1);
  lcd.print("Voltage: ");
  lcd.print(reading);
  lcd.print(" V");
  }
  else if(V3_L==LOW){
  reading = analogRead(VoltagePin) * (5.0 / 1023.0)*votageFactor3; // 0-100V
  lcd.setCursor(0, 1);
  lcd.print("Voltage: ");
  lcd.print(reading);
  lcd.print(" V");
  }

  else
    delay(500);

}
else if(V_State == HIGH && I_State == LOW && R_State == HIGH)
{
  // 读取电流值
  int reading = analogRead(currentPin) * (0.489); //  假设最大 500mA
  Serial.println(reading);
  lcd.setCursor(0, 1);
  lcd.print("Current: ");
  lcd.print(reading);
  lcd.print(" mA");
  delay(500);
}
else if(V_State == HIGH && I_State == HIGH && R_State == LOW )
{
  // 读取电阻值
 if(R1_L==HIGH ) {
  Serial.println("!");
  reading = analogRead(resistorPin) * resistorFactor1; //
  lcd.setCursor(0, 1);
  lcd.print("Resistor: ");
  lcd.print(reading);
  lcd.print(" om");
  delay(500);

 }
  else if(R2_L==HIGH )
 {
  Serial.println("!!");
```

```
        lcd.clear();
        reading = analogRead(resistorPin) * resistorFactor2; //
        lcd.setCursor(0, 1);
        lcd.print("Resistor: ");
        lcd.print(reading);
        lcd.print("kom");
        delay(500);
      }
      else
     delay(500); //  稍作延迟以避免抖动
   }
}
```

# 03 温度检测显示控制器的设计与制作

## 一、实验设计方案

本项目使用 Arduino UNO 扩展板进行温度检测显示控制。硬件框图如图 1 所示，下面对各部分模块进行说明。

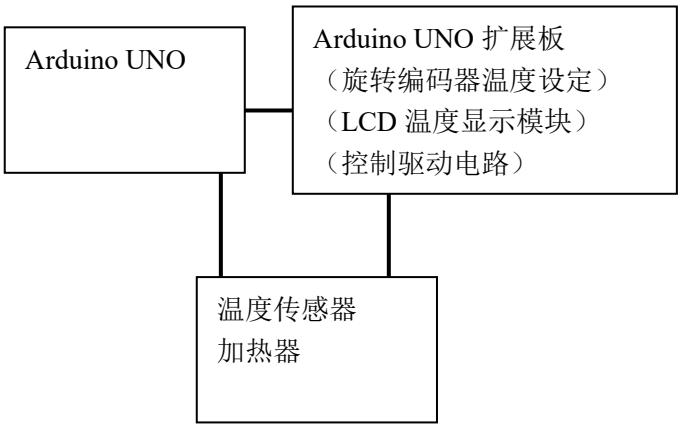


图 3.1 温度检测显示控制器框图

## 二、实验任务与要求

1、用 Arduino UNO 设计温度检测显示控制器，要求如下：

（1）选用合适的温度检测传感器，设计一个能实现测量范围 0°C~100°C，测量精度为 0.5°C 的温度检测转换电路；

（2）设计一个温度加热控制电路，实现用旋转编码器设定控制温度值，当检测温度值低于设定值时，加热电路工作；当检测温度值高于设定值时，加热电路停止加热。

2、设计的电路，完成相应器件的选择和参数计算，制作扩展板和控制加热电路。

3、编制与调试温度检测显示控制器程序。

4、将制作的扩展板、控制加热电路板与 Arduino UNO 板组装后，进行系统联调。

设计参考电路图如下：

## 三、设计举例

1、电路原理图

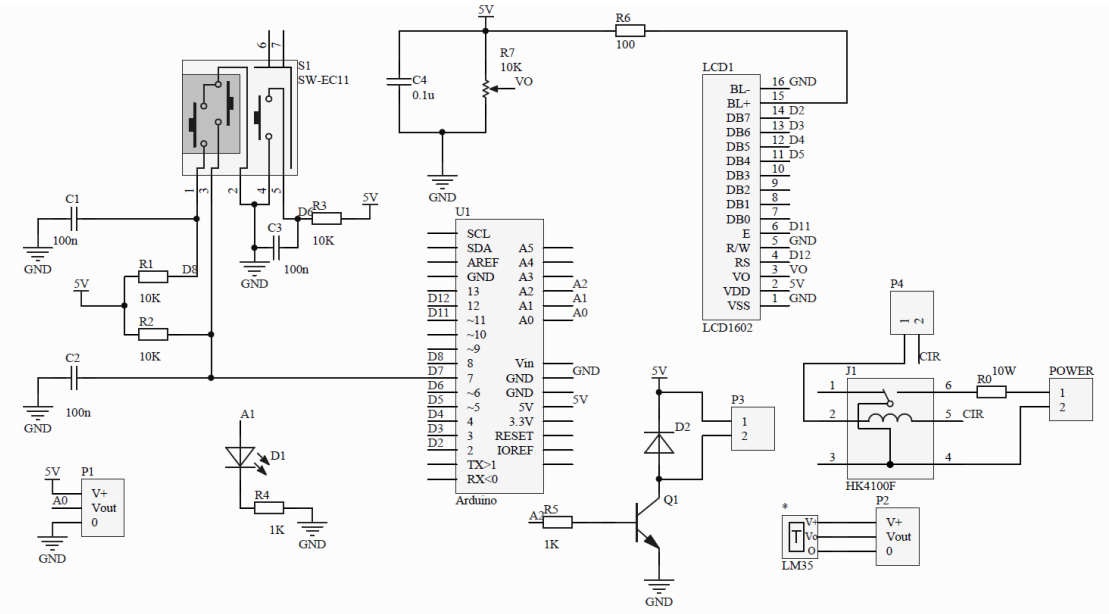


图 3.2 电路原理图

2、代码

```
#include <LiquidCrystal.h>
//LiquidCrystal display with:
//rs on pin 12
//enable on pin 11
//d4-7 on pins 5-2
LiquidCrystal lcd(12,11,5,4,3,2);

int ledPin = A1;
int relayPin = A2;
int aPin = 8;
int bPin = 7;
int buttonPin = 6;
int analogPin = A0;

float setTemp = 20.0;
float measureTemp;
char mode = 'C';   //can be changed to F
boolean override = false;
float hysteresis = 0.25;

void setup() {
  lcd.begin(2,20);
  pinMode(ledPin,OUTPUT);
  pinMode(relayPin,OUTPUT);
  pinMode(aPin,INPUT);
  pinMode(bPin,INPUT);
  pinMode(buttonPin,INPUT);
```

```
  lcd.clear();
}

void loop() {
  static int count = 0;
  measureTemp = readTemp();
  if(!digitalRead(buttonPin))
  {
    override = !override;
    updateDisplay();
    delay(500);//debounce
  }
 int change = getEncoderTurn();

  setTemp = setTemp + change *0.1;

  if(count == 1000)
  {
    updateDisplay();
    updateOutputs();
    count = 0;
  }
  count++;
  }

int getEncoderTurn()
{
  // return -1，0，or +1
  static int oldA=LOW;
  static int oldB=LOW;
  int result = 0;
  int newA=digitalRead(aPin);
  int newB=digitalRead(bPin);
  if (newA != oldA || newB != oldB)
  {
  // something has changed
   if(oldA == HIGH && newA==LOW)
  {
    result = -(oldB*2-1);
  }
  }
oldA = newA;
oldB = newB;
return result;
```

```
}
float readTemp()
{
  long a =analogRead(analogPin);
  float temp = 500.0*a/1023.0;
  return temp;
}

void updateOutputs()
{
  if(override||measureTemp<setTemp-hysteresis)
  {
    digitalWrite(ledPin,HIGH);
    digitalWrite(relayPin,HIGH);
  }
  else if(!override&&measureTemp>setTemp+hysteresis)
  {
    digitalWrite(ledPin,LOW);
    digitalWrite(relayPin,LOW);
  }
}
void updateDisplay()
{
  lcd.setCursor(0,0);
  lcd.print("Actual:");
  lcd.print(adjustUnits(measureTemp));
  lcd.print("o");
  lcd.print(mode);
  lcd.print(" ");

  lcd.setCursor(0,1);
  if(override)
  {
    lcd.print("OVERRIDE ON");
  }
  else
  {
    lcd.print("Set:");
    lcd.print(adjustUnits(setTemp));
  lcd.print("o");
  lcd.print(mode);
  lcd.print(" ");
  }
}
```

```
float adjustUnits(float temp)
{
  if(mode == 'C')
  {
    return temp;
  }
  else
  {
    return(temp*9)/5+32;
  }
}
```

## 四、注意事项

**控制系统中的迟滞问题考虑**

设计这样一个温度控制器时需考虑到要防止“摆动”情况的出现。一个简单的通断控制系统，就会出现摆动情况。当温度降至设定值之下时，电源接通，开始加热，直至温度高于设定值，电源断开。然后慢慢变冷，直至温度再次低于设定值，此时加热器又被接通，如此循环往复。但当温度恰好平衡在转变温度时，这种摆动会非常频繁，这种高频切换容易损坏器件。解决方法之一，就是引入迟滞（hysteresis）。图 4.17 展示如何利用一个迟滞值（迟滞变量设为 0.25°C）避免高频摆动的方法。

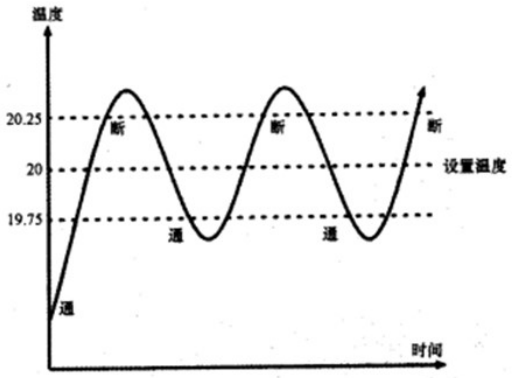


图 3.3　控制系统中的迟滞

温度随着电源接通而升高，它会到达设置点。然而，在温度超出设置点加上迟滞值之前电源并不会断开。同样，当温度下降时，降到低于设置点时并不会重新加电，而是只有当温度下降到低于设置点减去迟滞值时才会加电。

**显示闪烁问题考虑**

编程时不要连续刷新显示值，因为读数的任何细微变化都可能导致显示乱闪烁。因此不要在每次主循环中刷新显示，而是循环多次（如 1000 次）刷新一次。可引入一个变量（如 counter），该变量每个循环增加 1 次，当其增加到 1000 时，刷新显示并将 counter 变量复位到 0。

每次改变显示值都使用 lcd.clear()也会导致其闪烁。因此，可以采用将新温度值写在旧温度值的上面。

# 04 倒计时定时器的设计、制作与调试

## 一、实验设计方案

在日常生活中常常需要定时器，通过设定时间来通知我们一项任务完成或者下一项任务应该开始。本项目使用 Arduino UNO 与其扩展板来实现倒计时定时器。硬件框图如图 4.18 所示，下面对各部分模块进行说明。

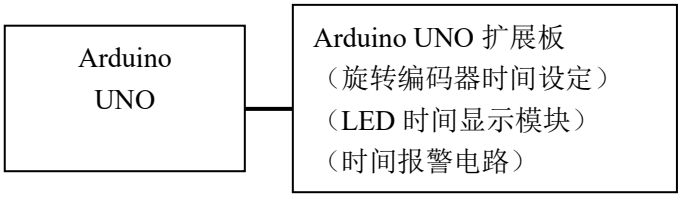


图 4.1 倒计时定时器框图

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，转动旋转编码器就可以改变定时时间；在定时运行状态，定时器倒计时。按下旋转编码器上的按钮可以对这两种状态进行切换。

转动旋转编码器时定时时间并不是以秒为步进长度来改变时间，我们有一个标准时间数组来实现。

EEPROM 库用于保存最后使用的时间，所以每当这个装置上电时，它将记住最后一次使用的时间。

## 二、实验任务与要求

1、用 Arduino UNO 设计倒计时定时器，要求如下：设定倒计时时间若干（设定标准时间数组），通过旋转编码器选择，时间到时报警。
2、设计电路，完成相应器件的选择，制作 Arduino UNO 扩展板。
3、编制与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、思考题

1、考虑倒计时定时器的应用，例如如何控制定时加热，简要介绍实现的电路。
2、如何通过修改程序，消除秒钟滴答声。

## 四、设计举例

1、电路原理图

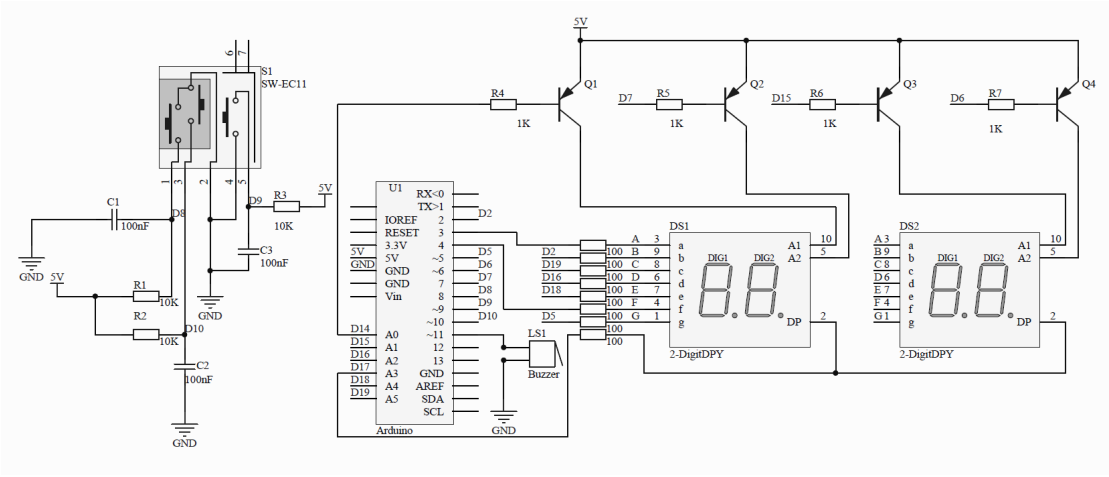


图 4.2 电路原理图

2、代码

下列代码是一个基于 Arduino 的计时器程序。这个程序利用了一块数码管显示器和一个旋钮编码器，可以设置定时器并开始/停止计时。下面是每个函数的功能注释：

setup(): 初始化函数，设置引脚模式和读取上次保存的定时设置。

loop(): 主循环函数，检测按钮状态并更新显示。

updateDisplay(): 更新数码管的显示，以显示当前的计时器数值。

process(): 处理函数，根据计时器状态决定是调整时间还是更新计时。

changeSetTime(int change): 改变设置时间的函数，根据旋钮编码器的转动改变时间设置。

updateCountingTime(): 更新计时时间的函数，每秒更新一次计时器的时间。

setDigit(int digit): 设置数码管显示的位数。

setSegments(int n): 设置数码管显示的数字。

getEncoderTurn(): 获取旋钮编码器的转动方向。

整体来说，这个程序实现了一个简单的定时器功能，可以通过旋钮编码器来设置时间，并通过按钮来开始/停止计时。

```
#include<EEPROM.h>
int segmentPins[] ={3,2,19,16,18,4,5,17};
int displayPins[] ={6,15,7,14};
int  times[]={5,10,15,20,30,45,100,130,200,230,300,400,500,600,700,800,900,1000,  1500,  2000,
3000};
int numTimes=20;
byte selectedTimeIndex;
int timerMinute;
int timerSecond;
int buzzerPin=11;
int aPin=8;
int bPin=10;
int buttonPin=9;

boolean stopped = true;
```

```
byte digits[10][8] ={
   //a   b   c   d   e   f   g .
   {1,1,1,1,1,1,0,0},//0
   {0,1,1,0,0,0,0,0},//1
   {1,1,0,1,1,0,1,0},//2
   {1,1,1,1,0,0,1,0},//3
   {0,1,1,0,0,1,1,0},//4
   {1,0,1,1,0,1,1,0},//5
   {1,0,1,1,1,1,1,0},//6
   {1,1,1,0,0,0,0,0},//7
   {1,1,1,1,1,1,1,0},//8
   {1,1,1,1,0,1,1,0}//9
  };

void setup()
{
for(int i=0;i<8;i++)
{
            pinMode(segmentPins[i],OUTPUT);
}
        for(int i=0;i<4;i++)
{
pinMode(displayPins[i],OUTPUT);
}
pinMode(buzzerPin,OUTPUT);
pinMode(buttonPin,INPUT);
pinMode(aPin,INPUT);
pinMode(bPin,INPUT);
selectedTimeIndex=EEPROM.read(0);
timerMinute=times[selectedTimeIndex]/100;
timerSecond=times[selectedTimeIndex]%100;
}

void loop()
{
     if(!digitalRead(buttonPin))
   {
          stopped = !stopped;
          digitalWrite(buzzerPin,LOW);
          while(!digitalRead(buttonPin)){};
          EEPROM.write(0,selectedTimeIndex);
   }
   updateDisplay();
}
```

```
void updateDisplay() //mmss
{
   int minsecs = timerMinute*100+timerSecond;
   int v = minsecs;
   for(int i=0;i<4;i++)
   {
        int digit=v%10;
        setDigit(i);
        setSegments(digit);
        v=v/10;
        process();
   }
   setDigit(5);// all digits off to prevent uneven illumination
}

void process()
{
   for(int i=0;i<100; i++)//tweak this number between flicker and blur
{
int change=getEncoderTurn();
if(stopped)
{
   changeSetTime(change);
}
   else
{
        updateCountingTime();
}
}
if(timerMinute == 0 && timerSecond == 0)
{
digitalWrite(buzzerPin,HIGH);
}
}

void changeSetTime(int change)
{
      selectedTimeIndex+=change;
      if(selectedTimeIndex<0)
      {
            selectedTimeIndex=numTimes;
      }
      else if(selectedTimeIndex>numTimes)
      {
```

```
            selectedTimeIndex = 0;
        }
        timerMinute=times[selectedTimeIndex]/100;
        timerSecond=times[selectedTimeIndex]%100;
}

void updateCountingTime()
{
        static unsigned long lastMillis;
        unsigned long m=millis();
        if(m>(lastMillis+1000) && (timerSecond>0 || timerMinute>0))
        {
           digitalWrite(buzzerPin,HIGH);
           delay(10);
           digitalWrite(buzzerPin,LOW);
           if(timerSecond==0)
           {
            timerSecond=59;
            timerMinute--;
           }
           else
           {
            timerSecond--;
           }
        lastMillis = m;
        }
}

void setDigit(int digit)
{
   for(int i=0;i<4;i++)
   {
      digitalWrite(displayPins[i],(digit!=i));
   }
}

void setSegments(int n)
{
   for(int i=0;i<8; i++)
   {
      digitalWrite (segmentPins[i], !digits[n][i]);
   }
}
```

```
int getEncoderTurn()
{
   // return -1，0，or +1
   static int oldA=LOW;
   static int oldB=LOW;
   int result = 0;
   int newA=digitalRead(aPin);
   int newB=digitalRead(bPin);
   if (newA != oldA || newB != oldB)
{
   // something has changed
     if(oldA == HIGH && newA==LOW)
   {
      result =-(2*oldB-1);
   }
}
oldA = newA;
oldB = newB;
return result;
}
```

# 05 Arduino 实时音频处理实验

## 一、实验目的

1、学习掌握用 Arduino UNO 进行实时音频信号处理；
2、学习掌握 PCB 电路板的设计和制作；
3、学习掌握 Arduino UNO 音频处理扩展板的设计与制作。

## 二、实验设计方案

数字信号处理，并不是数字信号处理器（DSP）独占的领域。Arduino 具有进行实时音频信号的处理的能力。

**1、 可调正弦波信号源**

Arduino 能产生的不仅数字信号，而且是高质量的波形。输出信号是连续的正弦波，可在较宽的频率范围内调整。压控振荡器（VCO）是电子合成器中最重要的元件之一。它被用来创建基本的音调，并由滤波器，混频器和包络变成了电子合成声。

可通过调节电压来改变信号源的频率。使用一个 10k 的电位器，使 UNO 板的 A0 的输入电压改变，从而改变输出信号的频率。得到的正弦波发生器，可以在许多音频领域中应用。

下面一段 Arduino 代码，实现了一个电压控制振荡器。它使用定时器 2 作为 PWM 定时器，并配置 ADC 来读取模拟信号。以下是代码的一些关键点：

在 setup 函数中，通过调用 calcSine 函数加载 sine 数组，设置 ADC 的采样率和通道，以及配置 Timer2 为快速 PWM 模式。

在 loop 函数中，等待 ADC 采样完成后，将当前的正弦值发送到 PWM 并递增计数器变量。

calcsine()，计算一个正弦波表。一系列正弦值不是外部预计算的，而是由单片机本身的一次计算得到。calcSine 函数计算 512 个值在 0 和 2 * pi 之间的正弦值，并将其存储在 sine 数组中。

ISR 函数（TIMER2_OVF_vect）在定时器 2 溢出时被触发，它通过中断服务程序读取 ADC 值并设置 adcSample 标志，以便 loop 函数可以执行下一次循环。函数每 16 μ s 被调用，以频率 62.5kHz 自动发生中断。中断函数第四次调用时，获取的 A0 模拟输入值被用于 loop（）中正弦波频率的设置。

总之，此代码实现了一个简单的电压控制振荡器，使用 PWM 输出正弦波，正弦波的频率根据从 ADC 读取输入电压来控制。

为了获得正常的正弦波，要在 PWM 输出端加滤波电路，可设计一个 LC 滤波器。

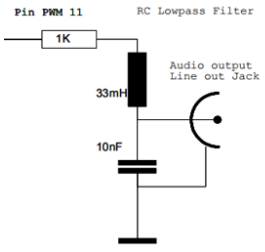


图 5.1　LC 滤波器

```
// Voltage-Controlled Oscillator
// 压控振荡器

#define sbi(PORT,bit)   (PORT |=1<<bit)   // 宏定义设置端口位为 1
#define cbi(PORT,bit)   (PORT &=~(1<<bit))   // 宏定义清除端口位为 0

const int pwmOutPin = 11; // PWM 输出引脚
const int testPin = 3;   // 测试引脚
const float pi = 3.141592;   // π值，用于计算正弦值
int indexCnt, interCnt;   //计数器计数值和中断计数器变量
byte sine[512];    // 存储正弦值的数组
// 中断控制变量
volatile boolean adcSample;   // ADC 采样标志
volatile byte bufferAdc0;   // ADC 缓冲区

void setup () {
   pinMode(pwmOutPin, OUTPUT);   // 将 PWM 引脚设置为输出
   pinMode(testPin, OUTPUT);      // 将测试引脚设置为输出
   calcSine( );    // 加载内存中的正弦表

// 设置 ADC 预分频器为 64，设置 8 位 ADC 和 VCC 参考电压，选择模拟通道 0
   ADCSRA |= 0b00000110;
   ADMUX |= 0b01100000;

// 设置 Timer2 为快速 PWM 模式
   TCCR2A |= 0b10000011;

// Timer2 无预分频
   TCCR2B |=0b00000001;
   cbi (TCCR2B, CS21) ;   // 清除 Timer2 的预分频位 CS21 和 CS22
   cbi (TCCR2B, CS22) ;

// 禁用 Timer0，启用 Timer2 的中断
   cbi(TIMSK0,TOIE0);
   sbi(TIMSK2,TOIE2);
}

void loop () {
   while (!adcSample) { } // 等待新的 ADC 样值
   digitalWrite (testPin, HIGH) ;   // 将测试引脚设置为高电平
   adcSample = false;   // 重置采样标志
   OCR2A = sine [indexCnt];   // 将当前值发送到 PWM
   indexCnt++;
   indexCnt += bufferAdc0;
```

```
  indexCnt &= 511;  // 将计数器计数值限制在 0 到 511 之间
  digitalWrite (testPin, LOW);  // 将测试引脚设置为低电平
}

// 计算包含 512 个值（在 0 到 2*pi 之间）的正弦数组
void calcSine () {
  for (int n =0; n <= 511; n++) {
    sine [n] = 127 * sin (pi * n / 255) + 128;
  }
}

// 获取电位器值
ISR (TIMER2_OVF_vect) {
  interCnt++; // 减少采样率，从 62.5 kHz 降到 15.625 kHz
  if (4 == interCnt) {
    bufferAdc0 = ADCH; // 获取 8 位 ADC 值
    adcSample = true;  // 设置 ADC 采样标志
    sbi (ADCSRA, ADSC) ;  // 开始下一次转换
    interCnt = 0;  // 重置中断计数器
  }
}
```

**2、数字信号处理**

除了产生高质量的模拟信号，Arduino 也可以作为一个信号处理器。在这一部分中，我们以混响发生器为例来说明。

以模拟技术产生人工混响一直赋有挑战性。高、低和带通滤波器的制作相对简单，但产生混响是困难的。

在微控制器中，混响效果可以使用循环缓冲存储器实现。为此，模拟信号首先像往常一样通过 ADC 数字化。Arduino 的 ADC 工作在输入电压 0 和 5V 之间，所以一个 2.5 V 的偏置电压必须被添加到输入信号上。通过耦合电容，将调整后在 0 至 5V 的范围内的信号发送到 ADC 输入端。图 5.2 为用于此目的的简单电路。

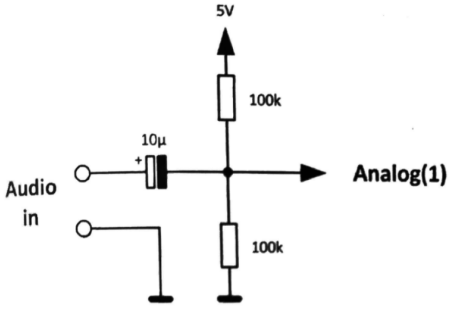


图 5.2 混响发生器输入电路

数字化值存储在数组中。为了产生的混响效果所需的延迟，反复读取循环缓冲区。延时用一单独的电位器调节。

由于 ATmega328 的资源有限，我们限制数组为 512 字节。

在环形缓冲区，这个存储器阵列构成一个循环中，一个指针用于访问写入数据的内存位

置和第二个指针来访问读取位置。完成一个步骤，两个指针都指向下一个位置。如果指针到达数组的结尾（511），指针又被设置为 0，旧的数据将被覆盖。读写指针之间的距离决定了时间延迟的长度。

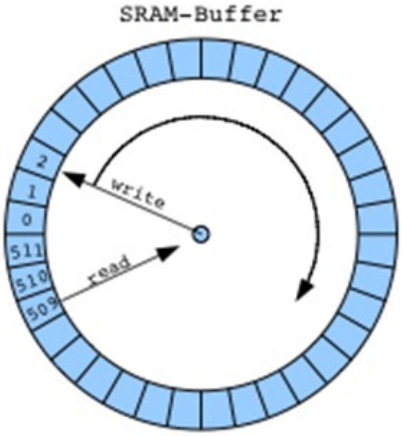


图 5.3 环形缓冲区

在微控制器中，混响效果可以使用循环缓冲存储器实现。数字化值存储在数组中。为了产生的混响效果所需的延迟，反复读取循环缓冲区。延时用一单独的电位器调节。

与前面在正弦波发生器相同，使用快速 PWM 模式，在 setup（）中的相关的设置也相同。

这里，电位器值和模拟输入信号都由中断程序采样。为此，交替地读出 A1（模拟输入）和 A0（电位器）。

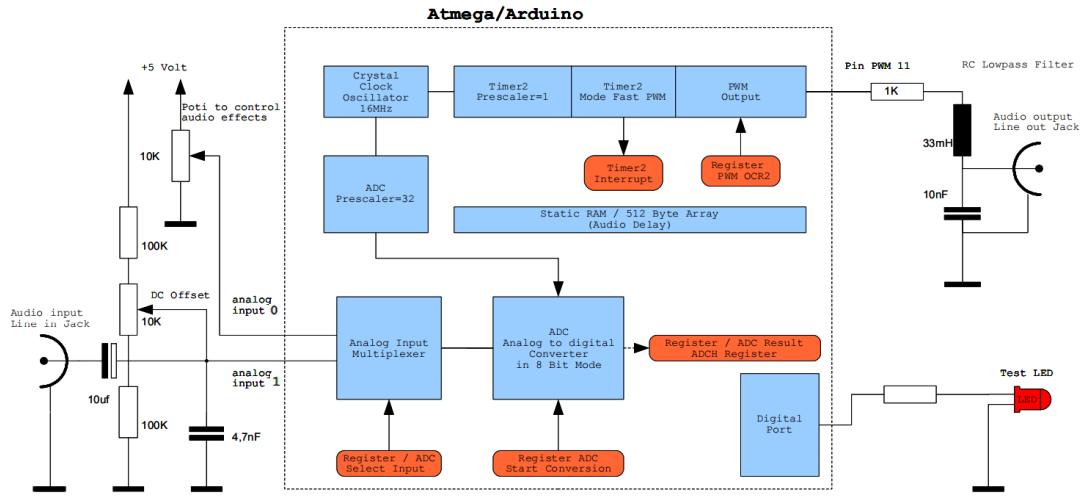


图 5.4 混响发生器原理图

在主程序中，循环缓冲器被写入并在延迟后读取，时间延迟与 A0 处的电位器电压成比例。

混响发生器示例程序

```
// Reverb Generator
#define sbi(PORT,bit)   (PORT |=1<<bit)   // 宏定义设置端口位为 1
#define cbi(PORT,bit)   (PORT &=~(1<<bit))   // 宏定义清除端口位为 0
```

```
const int pwmOutPin = 11; // PWM 输出引脚
const float pi = 3.1415926535;
const byte halfMax = 127;
boolean div2, selChan;   // 采样控制变量

//全局访问的中断变量
volatile boolean adcSample;
volatile byte potSampler, analogSample;

int indexCounter, valAnalog, valPot;
byte pwmBuf, ringbuff [512] ;   // Audio Ringbuffer Array   8-Bit

void setup () {
    pinMode (pwmOutPin, OUTPUT) ; // s PWM 输出

// 设置 ADC 预分频为 64，设置 8 位 ADC 和 VCC 参考电压，选择模拟通道 0
ADSRA |=0b00000110
ADMUX |=0b01100000

// 设置 Timer2 为快速 PWM 模式
  TCCR2A |= 0b10000011;

// 设置 Timer2 预分频
  TCCR2B |=0b00000001;
  cbi (TCCR2B, CS21) ;   // 清除 Timer2 的预分频位 CS21 和 CS22
  cbi (TCCR2B, CS22) ;

// 禁用 Timer0，启用 Timer2 的中断
  cbi(TIMSK0,TOIE0);
  sbi(TIMSK2,TOIE2);

valAnalog = analogSample;

void loop ( )   {
while (! adcSample) { }
adcSample = false;

// 根据电位器的值对延迟后的采样值进行缩放
valPot = (halfMax – ringbuff[indexCounter]) * potSample/255;
valAnalog = halfMax - analogSample;
valAnalog = valAnalog + valPot;

//限制模拟值的范围（最小值/最大值）
if (valAnalog < -halfMax) valAnalog = -halfMax;
```

```
if (valAnalog > halfMax) valAnalog = halfMax;

// PWM 信号上叠加一个偏置
pwmBuf = halfMax + valAnalog;
ringbuff [indexCounter] = pwmBuf;
indexCounter++;

// 将缓冲区索引限制在 0 到 511 的范围内
indexCounter = indexCounter & 511;

OCR2A = pwmBuf; // 将当前采样值写入 PWM 通道

//获取模拟值和电位器的值
ISR (TIMER2_OVF_vect) {
    PORTB = PORTB | 1;
    div2 =! div2; // 将采样频率减半
    if (div2) {
      selChan = ! selChan;
      if (selChan) {
          potSample = ADCH;
          sbi (ADMUX, MUX0); // 将多路复用器设置为 ADC1
      }
      else {
          analogSample = ADCH;
          cbi (ADMUX, MUX0); // 将多路复用器设置为 ADC0
          adcSample = true;
           }
      sbi (ADCSRA, ADSC); // 启动下一次转换
  }
}
```

**3、音频输出放大器**

使用扬声器可以获得较好的音频质量。问题是，其中一个是阻抗仅仅几欧姆，因此不能直接连接到 aduino 板的引脚，为此需要一个放大器，LM386 一个非常合适我们应用的放大器，它只需要几个额外元件构成电路，如图 5.5 所示。除了放大器，还具有音频滤波器的功能。2.2uF 电容器负责放大器的直流去耦，并且具有一定的高通特性，而 1μF 电容器提供低通滤波。当使用简单的脉宽调制产生音频信号时，高频 PWM 谐波将被很好地抑制，而在较低的千赫范围内所需的音频频率传递到放大器。

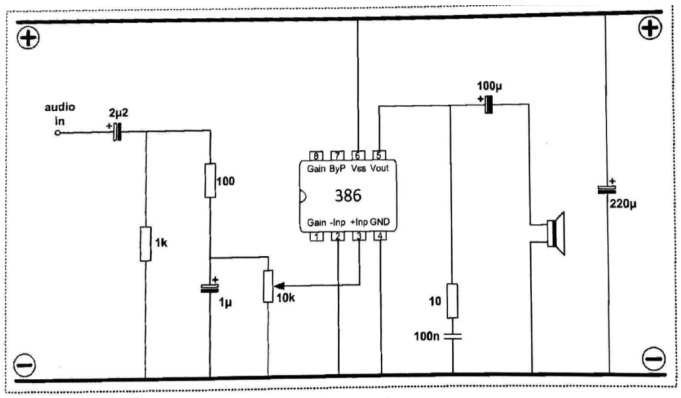


图 5.5　基于 LM386 音频放大器电路图

## 三、实验任务与要求

1、用 Arduino UNO 实现音频信号处理设计，

1）音频正弦波信号源+音频输出放大器

2）混响发生器+音频输出放大器

1）和 2）任选一项，形成一个完整的电路，其中的滤波器可以用 RC 滤波器或 LC 滤波器，也可设计有源滤波器。完成相应器件的选择，制作 Arduino UNO 扩展板。

3、编制与调试相关程序。

4、将制作的扩展板与 Arduino UNO 板组装后，进行系统联调。

## 四、设计举例

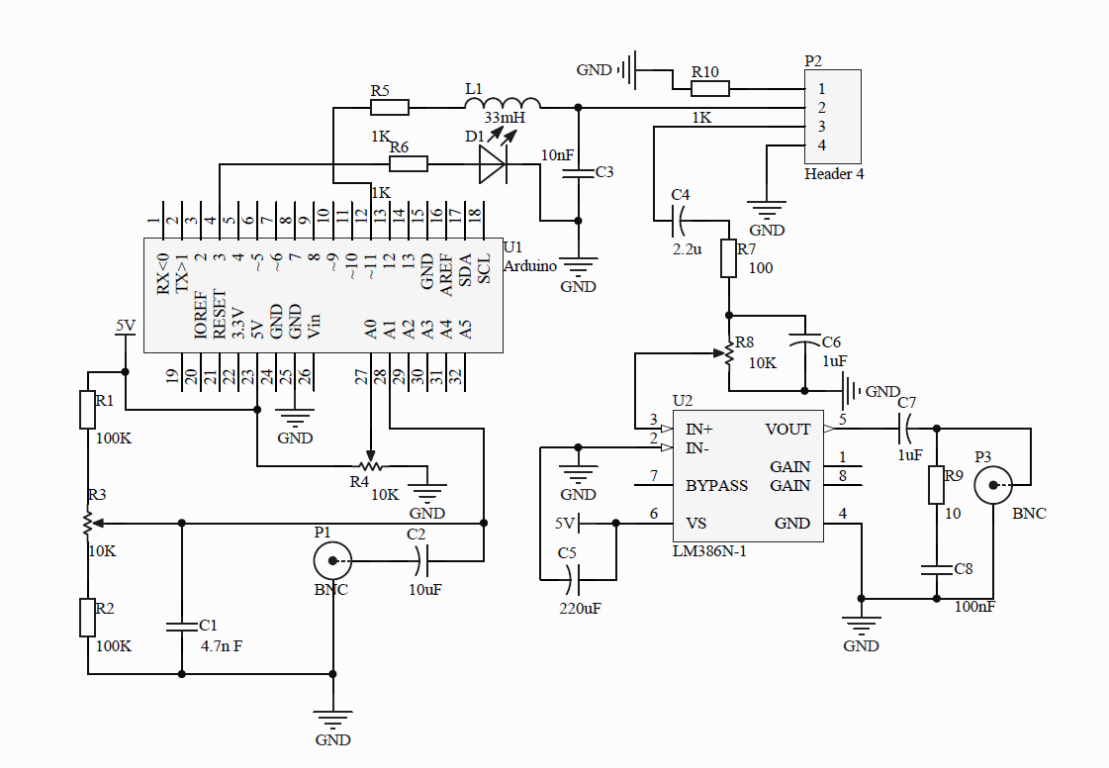


图 5.6 电路原理图

图 5.6 是参考设计的电路原理图。

# 06 多功能数字时钟的设计与制作

## 一、实验目的

1、学习掌握用 Arduino UNO 设计数字时钟；
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 DS1302 时钟芯片和 LCD1602 液晶显示屏的使用。

## 二、实验设计方案

本项目使用 Arduino UNO 设计多功能数字时钟。硬件框图如图 1 所示，下面对各部分模块进行说明。

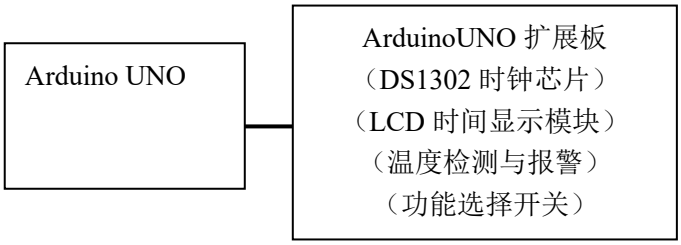


图 6.1　多功能数字时钟框图

**DS1302 时钟芯片**

DS1302 是美国 DALLAS 公司推出的一种高性能、低功耗的实时时钟芯片，附加 31 字节静态 RAM，采用 SPI 三线接口与 CPU 进行同步通信，并可采用突发方式一次传送多个字节的时钟信号和 RAM 数据。实时时钟可提供秒、分、时、日、星期、月和年，一个月小与 31 天时可以自动调整，且具有闰年补偿功能。工作电压宽达 2.5～5.5V。采用双电源供电（主电源和备用电源），可设置备用电源充电方式，提供了对后背电源进行涓细电流充电的能力。详细内容，可查阅芯片的 Datasheet。

**LCD 显示模块**

本次实验使用 Arduino UNO 直接驱动 1602 液晶显示字母和数字，LCD 显示模块有一个优势就是将驱动电路集成在模块中。LCD 显示模块具有标准，不同的生产商所生产的大多数模块都可以按相同方法来使用。1602 液晶在应用中非常广泛，最初的 1602 液晶使用的是 HD44780 控制器，现在各个厂家的 1602 模块基本上都是采用了与之兼容的 IC，所以特性上基本都是一致的。有关 LCD1602 模块的内容，详见附录 1。

**温度传感器**

温度传感器可采用 LM35，具体内容参见“温度检测显示控制器的设计与制作”。

## 三、实验任务与要求

1、用 Arduino UNO 设计多功能数字时钟，要求实现功能：
（1）显示时间、日期和星期
（2）断电保存时间
（3）通过按钮设置时间、日期
（4）整点响铃

（5）自定义闹钟
（6）显示温度
（7）自定义报警温度
（8）按键功能：按选择键进入设置时间功能；同时按 +- 键进入闹钟和报警温度设置功能；
（9）再按选择键光标跳动，光标跳到哪，当前的参数即可通过加减键修改。
2、设计的电路，完成相应器件的选择和参数计算，制作 Arduino UNO 扩展板。
3、编制与调试多功能数字时钟程序。
4、将制作的 Arduino UNO 扩展板与 Arduino UNO 板组装后，进行系统联调。

## 四、设计举例

1、电路原理图

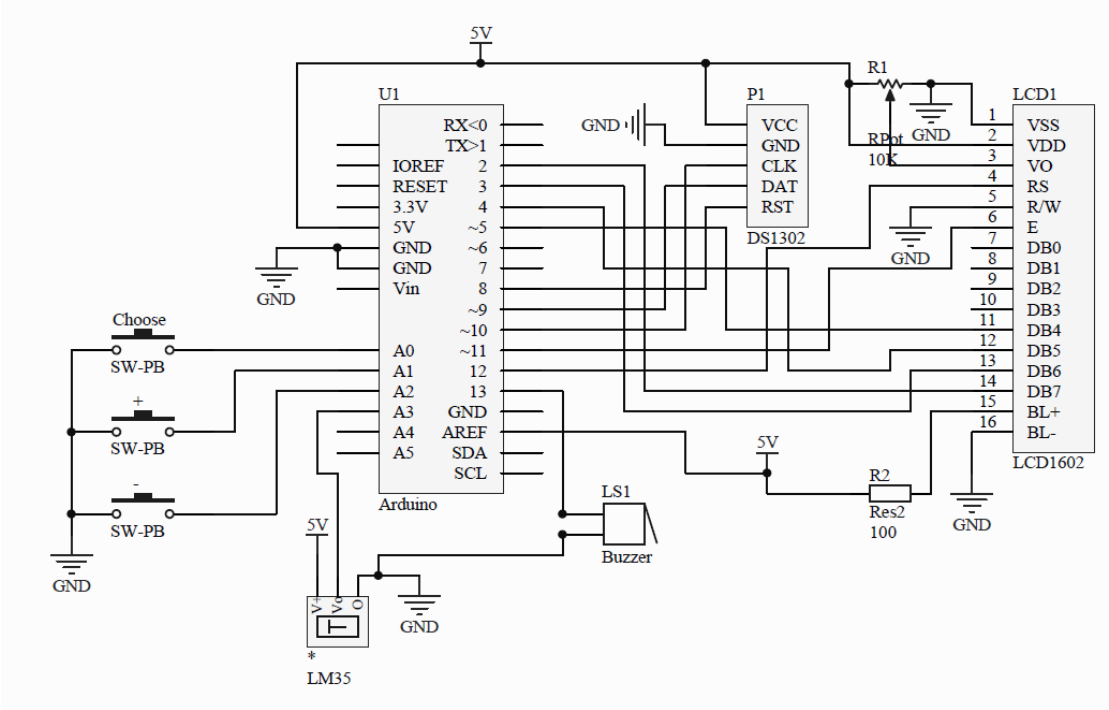


图 6.2　电路原理图

**2、代码**

设计参考程序如下：

```
/* *
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
```

```
 * */
#include <DS1302.h>
#include <LiquidCrystal.h>          //LCD1602 显示头文件

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

#define choose   A0    //选择端口
#define add   A1       //加
#define minus   A2     //减
#define Tone 13        //蜂鸣器端口
uint8_t CE_PIN    = 8;    //DS1302 RST 端口
uint8_t IO_PIN    = 9;    //DS1302 DAT 端口
uint8_t SCLK_PIN = 10;   //DS1302 CLK 端口
DS1302 rtc(CE_PIN, IO_PIN, SCLK_PIN);   //创建  DS1302  对象
unsigned long seconds;
int s = 0, m = 0, h = 0, d = 0, mon = 0, y = 0;    //时间进位
int second = 0, minute = 0, hour = 0, day = 0, month = 0, year = 0;   //当前时间
int SECOND = 0, MINUTE = 0, HOUR = 0, DAY = 0, MONTH = 0, YEAR = 0;   //初始时间
int chose = 0, alarm_choose = 0 ,ButtonDelay = 20, frequence = 2093;
int alarm_hour = 7, alarm_minute = 30, alarm_second = 0;   //闹钟时间
float Temperatures, Temp_Alarm = 30 ;

/**  格式化输出  */
void FormatDisplay(int col, int row,int num){
    lcd.setCursor(col, row);
    if(num < 10)    lcd.print("0");
    lcd.print(num);
}

/**  计算时间  */
void time() {
    second = (SECOND + seconds) % 60;    //计算秒
    m = (SECOND + seconds) / 60;          //分钟进位
    FormatDisplay(6,1,second);

    minute = (MINUTE + m) % 60;   //计算分钟
    h = (MINUTE + m) / 60;         //小时进位
    FormatDisplay(3,1,minute);

    hour = (HOUR + h) % 24;    //计算小时
    d = (HOUR + h) / 24;       //天数进位
    FormatDisplay(0,1,hour);

    lcd.setCursor(2, 1);
```

```
    lcd.print(":");
    lcd.setCursor(5, 1);
    lcd.print(":");
}

/** 根据年月计算当月天数 */
int Days(int year, int month){
    int days = 0;
    if (month != 2){
        switch(month){
            case 1: case 3: case 5: case 7: case 8: case 10: case 12: days = 31;  break;
            case 4: case 6: case 9: case 11:  days = 30;  break;
        }
    }else{    //闰年
        if(year % 4 == 0 && year % 100 != 0 || year % 400 == 0){
            days = 29;
        }
        else{
            days = 28;
        }
    }
    return days;
}

/** 计算当月天数 */
void Day(){
    int days = Days(year,month);
    int days_up;
    if(month == 1){
        days_up = Days(year - 1, 12);
    }
    else{
        days_up = Days(year, month - 1);
    }
    day = (DAY + d) % days;
    if(day == 0){
        day = days;
    }
    if((DAY + d) == days + 1 ){
        DAY -= days;
        mon++;
    }
    if((DAY + d) == 0){
        DAY += days_up;
```

```
            mon--;
        }
        FormatDisplay(8,0,day);
}

/**  计算月份  */
void Month(){
        month = (MONTH + mon) % 12;
        if(month == 0){
            month = 12;
        }
        y = (MONTH + mon - 1) / 12;
        FormatDisplay(5,0,month);
        lcd.setCursor(7, 0);
        lcd.print('-');
}

/**  计算年份  */
void Year(){
        year = ( YEAR + y ) % 9999;
        if(year == 0){
            year = 9999;
        }
        lcd.setCursor(0, 0);
        if(year < 1000){
            lcd.print("0");
        }
        if(year < 100){
            lcd.print("0");
        }
        if(year < 10){
            lcd.print("0");
        }
        lcd.print(year);
        lcd.setCursor(4, 0);
        lcd.print('-');
}

/**  根据年月日计算星期几  */
void Week(int y,int m, int d){
        if(m == 1){
            m = 13;
        }
        if(m == 2){
```

```
        m = 14;
    }
    int week = (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7+1;
    String weekstr = "";
    switch(week){
        case 1: weekstr = "Mon. ";    break;
        case 2: weekstr = "Tues. ";   break;
        case 3: weekstr = "Wed. ";    break;
        case 4: weekstr = "Thur. ";   break;
        case 5: weekstr = "Fri. ";    break;
        case 6: weekstr = "Sat. ";    break;
        case 7: weekstr = "Sun. ";    break;
    }
lcd.setCursor(11, 0);
lcd.print(weekstr);
}

/**  显示时间、日期、星期  */
void Display() {
    time();
    Day();
    Month();
    Year();
    Week(year,month,day);
}

/**  显示光标  */
void DisplayCursor(int rol, int row) {
    lcd.setCursor(rol, row);
    lcd.cursor();
    delay(100);
    lcd.noCursor();
    delay(100);
}
/**  设置初始时间  */
void set(int y, int mon, int d, int h, int m, int s){
    YEAR = y;
    MONTH = mon;
    DAY = d;
    HOUR = h;
    MINUTE = m;
    SECOND = s;
}
```

```
/** 通过按键设置时间 */
void Set_Time(int rol, int row, int &Time){
    DisplayCursor(rol, row);
    if(digitalRead(add) == LOW){
        delay(ButtonDelay);
        if(digitalRead(add) == LOW){
            Time ++;
        }
        Display();
    }
    if(digitalRead(minus) == LOW){
        delay(ButtonDelay);
        if(digitalRead(minus) == LOW){
            Time --;
        }
        Display();
    }
}

/** 按键选择 */
void Set_Clock(){
    if(digitalRead(choose)==LOW){
        lcd.setCursor(9, 1);
lcd.print("SetTime");
        while(1){
            if(digitalRead(choose) == LOW){
                delay(ButtonDelay);
                if(digitalRead(choose) ==LOW){
                    chose++;
                }
            }
            seconds = millis()/1000;
            Display();
            if(chose == 1){
                Set_Time(1, 1, HOUR);        //SetHour
            }else if(chose == 2){
                Set_Time(4, 1, MINUTE);     //SetMinute
            }else if(chose == 3){
                Set_Time(7, 1, SECOND);     //SetSecond
            }else if(chose == 4){
                Set_Time(9, 0, DAY);          //SetDay
            }else if(chose == 5){
                Set_Time(6, 0, MONTH);       // SetMonth
            }else if(chose == 6){
```

```
                Set_Time(3, 0, YEAR);        //SetYear
            }else if(chose >= 7) {
                chose = 0;
                break;
            }
        }
        delay(150);//新增（2024）
    }
}

/** 设置闹钟小时 */
void Set_Alarm_Hour(){
  DisplayCursor(1, 1);
  if(digitalRead(add) == LOW){
      delay(ButtonDelay);
      if(digitalRead(add) == LOW){
          alarm_hour ++;
          if(alarm_hour == 24){
              alarm_hour = 0;
          }
          FormatDisplay(0,1,alarm_hour);
      }
  }
   if(digitalRead(minus) == LOW){
           delay(ButtonDelay);
           if(digitalRead(minus) == LOW){
               alarm_hour --;
               if(alarm_hour == -1){
                   alarm_hour = 23;
               }
               FormatDisplay(0,1,alarm_hour);
           }
    }
}

/** 设置闹钟分钟 */
void Set_Alarm_Minute(){
  DisplayCursor(4, 1);
  if(digitalRead(add) == LOW) {
      delay(ButtonDelay);
      if(digitalRead(add) == LOW){
           alarm_minute ++;
           if(alarm_minute == 60){
               alarm_minute = 0;
```

```
            }
            FormatDisplay(3,1,alarm_minute);
        }
    }
    if(digitalRead(minus) == LOW){
            delay(ButtonDelay);
            if(digitalRead(minus) == LOW){
                alarm_minute --;
                if(alarm_minute == -1){
                    alarm_minute = 59;
                }
                FormatDisplay(3,1,alarm_minute);
            }
    }
}

/**  设置报警温度  */
void Set_Alarm_Temp(){
    DisplayCursor(10, 1);
    if(digitalRead(add) == LOW) {
        delay(ButtonDelay);
        if(digitalRead(add) == LOW){
            Temp_Alarm ++;
        }
    }
    if(digitalRead(minus) == LOW){
        delay(ButtonDelay);
        if(digitalRead(minus) == LOW){
            Temp_Alarm --;
        }
    }
}

/**  进入报警设置  */
void Set_Alarm(){
    if(digitalRead(add) == LOW && digitalRead(minus) == LOW){
        alarm_hour = hour;
        alarm_minute = minute;
        //alarm_choose   = 1;
        lcd.setCursor(0, 0);
        lcd.print("set alarm          ");
        lcd.setCursor(6, 1);
        lcd.print("00");            //闹钟秒数
        while(1){
```

```
            if(digitalRead(choose) == LOW){
                delay(ButtonDelay);
                if(digitalRead(choose) == LOW){
                    alarm_choose++;

                }
            }
            lcd.setCursor(9, 1);
            lcd.print(Temp_Alarm);
            lcd.setCursor(14, 1);
            lcd.print((char)223);     //显示 o 符号
            lcd.setCursor(15, 1);
            lcd.print("C");            //显示字母 C
            if(alarm_choose == 1){
                Set_Alarm_Hour();
            }else if(alarm_choose == 2){
                Set_Alarm_Minute();
            }else if(alarm_choose == 3){
                Set_Alarm_Temp();
            }else if(alarm_choose >= 4){
                alarm_choose = 0;
                break;
            }
        }
        delay(150);
    }
}

/**  正点蜂鸣  */
void Point_Time_Alarm(){
    if(minute == 0 && second == 0){
        tone(Tone,frequence);
        delay(500);
        noTone(Tone);
    }
}

/**  闹钟  指定时间蜂鸣  */
void Clock_Alarm(){
  if(hour == alarm_hour && minute == alarm_minute && second == alarm_second){
        tone(Tone,frequence);
        delay(5000);
        noTone(Tone);
  }
```

```
}

/**  获取 LM35 温度  */
void GetTemperatures(){
    long a = analogRead(A3); // Get temperatures
    Temperatures = 500.0*a/1023.0;
    lcd.setCursor(9, 1) ;
    lcd.print(Temperatures); //获取温度
    lcd.setCursor(14, 1);
    lcd.print((char)223); //显示 o 符号
    lcd.setCursor(15, 1);
    lcd.print("C"); //显示字母 C
}

/**  超过指定温度报警  */
void Temperatures_Alarm(){
    if(Temperatures >= Temp_Alarm){
        tone(Tone,frequence);
        delay(500);
        noTone(Tone);
    }
}

void setup(){
    for(int i = 2;i <= 13; i++){
        pinMode(i,OUTPUT);
    }
    digitalWrite(add, HIGH);
    digitalWrite(minus, HIGH);
    digitalWrite(choose, HIGH);
    lcd.begin(16, 2);   //初始化 LCD1602
     //读取 DS1302 芯片的时间
    rtc.writeProtect(false);   //  关闭 DS1302 芯片写保护
    rtc.halt(false);              //为 true 时 DS1302 暂停
    Time t;
    t=rtc.getTime();
    set(t.year,t.mon,t.date,t.hour,t.min,t.sec);   //设置 DS1302 芯片初始时间
    //      Time t;
    //      t.hour=10;t.min=47;t.sec=59;
    //      t.year=2025;t.mon=2;t.date=12;
    //      rtc.setTime(t.hour,t.min,t.sec); //设置 DS1302  芯片初始时间
    //      rtc.setDate(t.year,t.mon,t.date);
    //      set(t.year,t.mon,t.date,t.hour,t.min,t.sec);
    }
```

```
void loop() {
    seconds = millis()/1000;     //获取单片机当前运行时间
    Display();          //显示时间
    Set_Clock();       //设置时间
    Set_Alarm();       //设置闹钟
    Point_Time_Alarm();     //正点蜂鸣
    Clock_Alarm();             //闹钟时间蜂鸣
    GetTemperatures();       //获取 LM35 温度
    Temperatures_Alarm();   //超过指定温度报警

    //将单片机的当前时间写到 DS1302 芯片中
    rtc.setTime(hour,minute,second);
    rtc.setDate(day,month,year);

    }
```

# 附录 1

1602LCD 主要技术参数：

显示容量为 16×2 个字符；

芯片工作电压为 4.5～5.5V；

工作电流为 2.0mA（5.0V）；

模块最佳工作电压为 5.0V；

字符尺寸为 2.95×4.35（W×H）mm。

1602 采用标准的 16 脚接口，其中:

第 1 脚：VSS 为地电源

第 2 脚：VDD 接 5V 正电源

第 3 脚：V0 为液晶显示器对比度调整端，接正电源时对比度最弱，接地电源时对比度最高，对比度过高时会产生“鬼影”，使用时可以通过一个 10K 的电位器调整对比度

第 4 脚：RS 为寄存器选择，高电平时选择数据寄存器、低电平时选择指令寄存器。

第 5 脚：R/W 为读写信号线，高电平时进行读操作，低电平时进行写操作。当 RS 和 RW 共同为低电平时可以写入指令或者显示地址，当 RS 为低电平 RW 为高电平时可以读信号，当 RS 为高电平 RW 为低电平时可以写入数据。

第 6 脚：E 端为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令。

第 7～14 脚：D0～D7 为 8 位双向数据线。

第 15 脚：背光电源正极

第 16 脚：背光电源负极

LCD 可按 4 位模式连接，如图 1 所示。

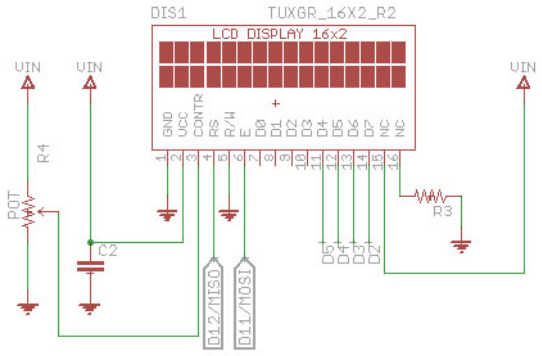


图 F.1　LCD 4 位连接模式

程序模版如下：

```
#include <LiquidCrystal.h>
LiquidCrystal iLCD (12, 11, 5, 4, 3, 2);
void setup ( )
{
    iLCD.begin (16, 2 );
    // your code here...
}
void loop ()
{
```

```
    // your code here ...
}
```

在 Arduino 的安装目录下\libraries\LiquidCrystal 可以查看到函数的原型

LiquidCrystal()——定义你的 LCD 的接口：各个引脚连接的 I/O 口编号，格式为

LiquidCrystal(rs, enable, d4, d5, d6, d7)  //4 位连接模式

LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)  //4 位连接模式

LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)  //8 位连接模式

LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)  //8 位连接模式

begin()——定义 LCD 的长宽（n 列×n 行），格式 lcd.begin(cols, rows)

clear()——清空 LCD，格式 lcd.clear()

home()——把光标移回左上角，即从头开始输出，格式 lcd.home()

setCursor()——移动光标到特定位置，格式 lcd.setCursor(col, row)

write()——在屏幕上显示内容（必须是一个变量，如”Serial.read()”），格式 lcd.write(data)

print()——在屏幕上显示内容（字母、字符串，等等），格式 lcd.print(data)

lcd.print(data, BASE)

cursor()——显示光标（一条下划线），格式 lcd.cursor()

noCursor()——隐藏光标，格式 lcd.noCursor()

blink()——闪烁光标，格式 lcd.blink()

noBlink()——光标停止闪烁，格式 lcd.noBlink()

display()——（在使用 noDisplay()函数关闭显示后）打开显示（并恢复原来内容），格式 lcd.display()

noDisplay()——关闭显示，但不会丢失原来显示的内容，格式为 lcd.noDisplay()

scrollDisplayLeft()——把显示的内容向左滚动一格，格式 lcd.scrollDisplayLeft()

scrollDisplayRight()——把显示的内容向右滚动一格，格式为 lcd.scrollDisplayRight()

autoscroll()——打开自动滚动，这使每个新的字符出现后，原有的字符都移动一格：如果字符一开始从左到右（默认），那么就往左移动一格，否则就向右移动，格式 lcd.autoscroll()

noAutoscroll()——关闭自动滚动，格式 lcd.noAutoscroll()

leftToRight()——从左往右显示，也就是说显示的字符会从左往右排列（默认），但屏幕上已经有的字符不受影响，格式 lcd.leftToRight()

rightToLeft()——从右往左显示，格式 lcd.rightToLeft()

# 附录 2

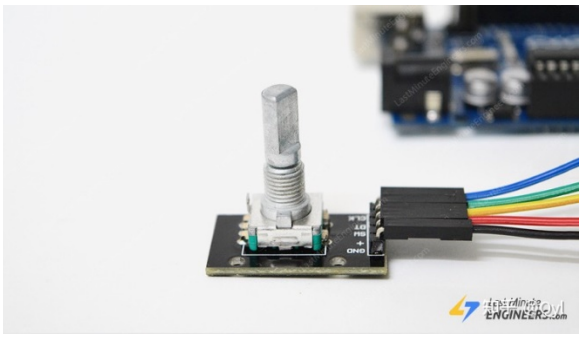

图 F.2 旋转编码器

旋转编码器是一种位置传感器，可将旋钮的角位置（旋转）转换为用于确定旋钮旋转方向的输出信号。

旋转编码器是如何工作的呢？

编码器内部是一个槽形磁盘，该磁盘连接到公共接地引脚 C 以及两个接触针 A 和 B，如下所示。

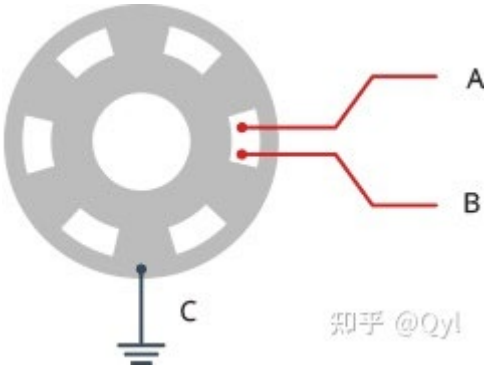


图 F.3 旋转编码器内部原理图

旋转旋钮时，A 和 B 根据旋转旋钮的方向以特定顺序与公共接地引脚 C 接触。

当它们接触公共接地时，它们会产生信号。当一个引脚先于另一引脚接触时，这些信号就会彼此错开 90° 。这称为正交编码。

图 F.4 旋转编码器工作原理图

顺时针旋转旋钮时，首先连接 A 引脚，然后连接 B 引脚。逆时针旋转旋钮时，首先连接 B 引脚，然后连接 A 引脚。

通过跟踪每个引脚何时与地面连接或与地面断开，我们可以使用这些信号变化来确定旋钮的旋转方向。您可以通过在 A 更改状态时观察 B 的状态来做到这一点。

当 A 更改状态时：

如果 B！＝A，指示旋钮顺时针旋转。

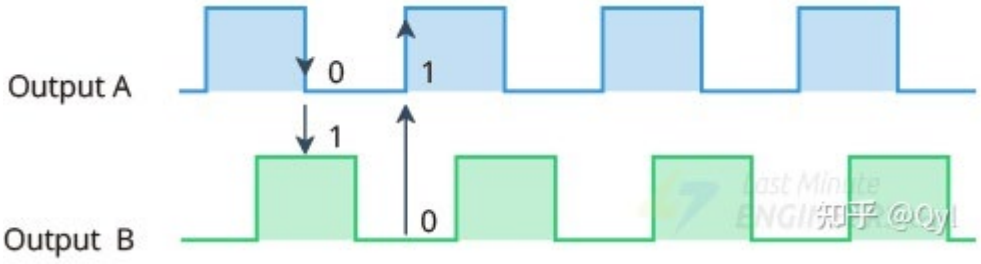


图 F.5 旋转编码器顺时针旋转波形图

如果 B＝A，指示旋钮逆时针旋转。

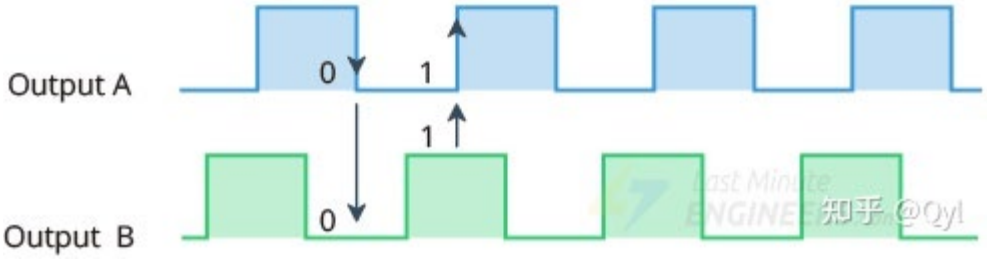


图 F.6 旋转编码器逆时针旋转波形图

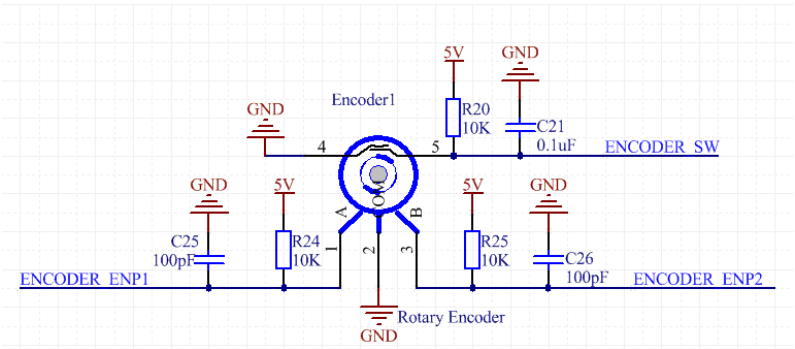


图 F.7 旋转编码器接线图

图 F.7 为 EC11 旋转编码器接线图，1、3 脚为 A、B 信号，4、5 脚为开关，2 脚接地。