LED 发光二极管

发光二极管简称为 LED, 是一种能发光的半导体电子元件，是将电能转换为光能的组件。最常用于室内外 LED 照明、LED 显示屏、交通信号灯、汽车用灯、显示屏的背光源、灯饰、光纤通信等等。

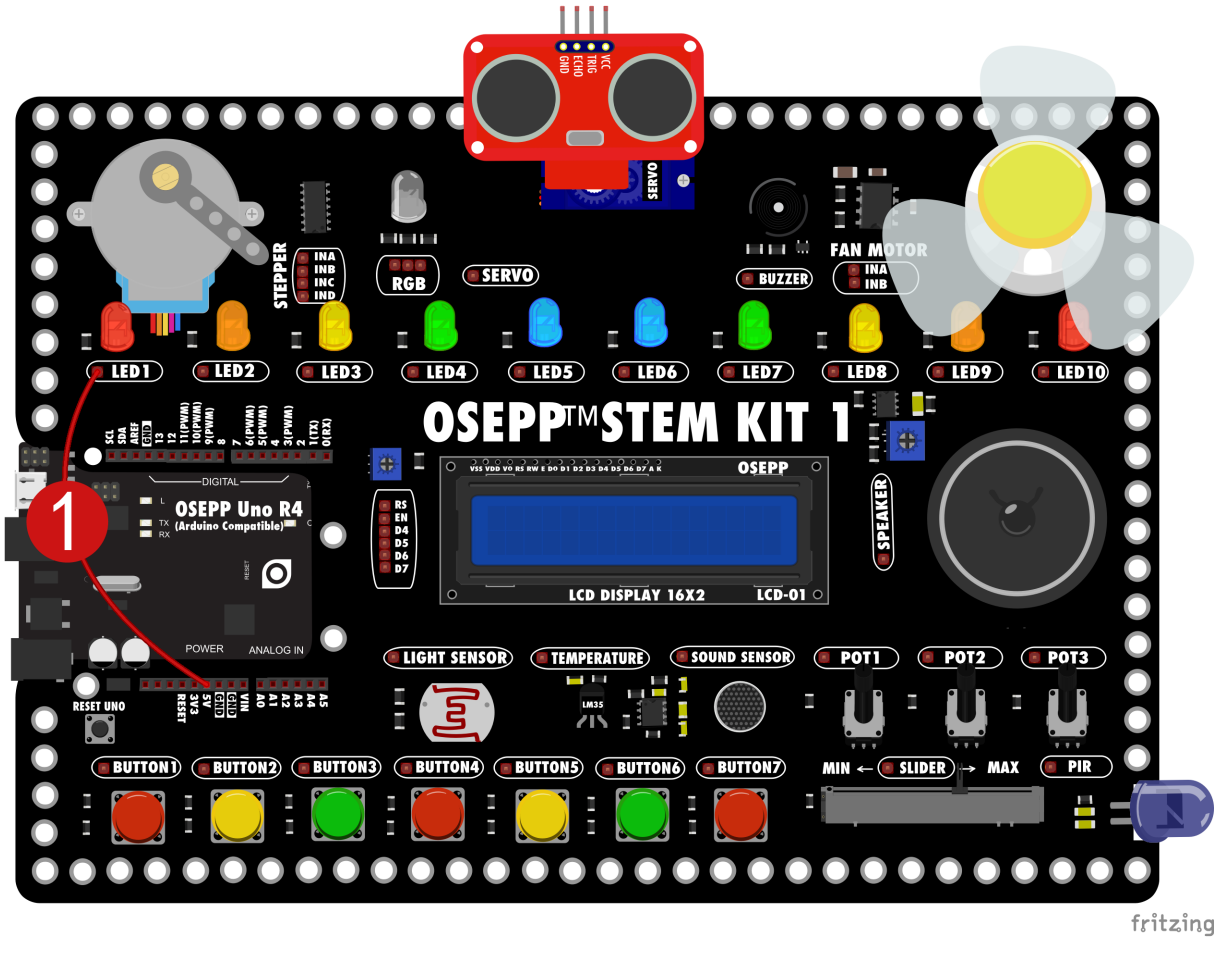
发光二极管具有效率高、寿命长、不易破损、反应速度快、可靠性高等传统光源不及的优点。

应用 1: 点亮 LED

如何点亮一个 LED 呢?
如果要让 **STEM KIT** 学习板上面的 LED 发光，我们只需要把 LED 的端子用导线连接到 **5v** 电源正极端口，然后给 **STEM KIT** 学习板接上电源。LED 就会发光了。

部署

如下图中使用杜邦线连接 **LED1** 端子到 **OSEPP UNO** 控制板的 **5V** 引脚，插上 USB 接头后给 **STEM KIT** 供电后，LED 就会发光了。



运行结果

在实验中，LED 连接到 5V 电压时，LED 就会发光。

解析

一个典型的红色 LED 一般只需要 1.5V 的正向电压，若阳极上的电压低于阴极 1.5V，将没有电流流过 LED，也不会发光。若阳极的电压比阴极高于 1.5V，LED 导通，但基本变成短路状态（电流会很大）。所以连接 LED 时必须使用电阻限制电流，否则 LED 将损坏。

一般 LED 的最大连续电流在 25mA，为了不让 LED 烧坏，**STEM KIT** 学习板上面的 LED 接了限流电阻，即使直接把导线接到 5v 也不会损坏 LED。

应用 2: Arduino 控制 LED

在实验开始之前，让我们花点时间安装编程软件及了解它的基本操作。

Arduino IDE

如果您有编程经验，喜欢使用文本编程软件，
请参考以下链接安装 **Arduino IDE**:
https://cn.osepp.com/tutorial/robopro/arduino_ide_install

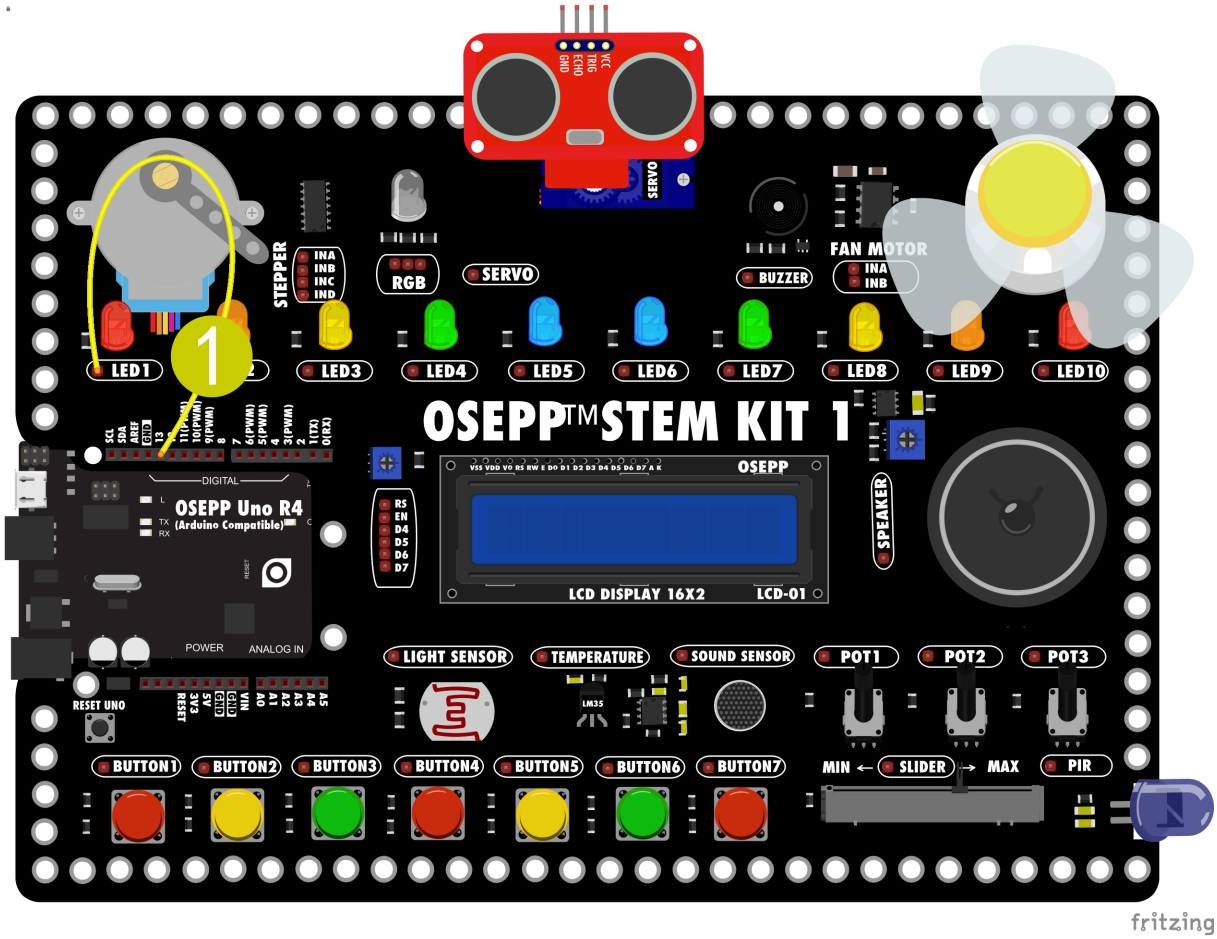
oseppBlock IDE

如果您是编程初学者，我们建议您先使用 **oseppBlock IDE** 软件。 **oseppBlock IDE** 是为初学者编写的一个基于图形的编程软件，基于 **Scratch2.0** 二次开发。通过拖放积木代码块组成程序，并实时转换成文本代码。
请参考以下链接安装 **oseppBlock IDE**:
下载地址: https://cn.osepp.com
安装教程: https://cn.osepp.com/tutorial/robopro/oseppblock_ide_install

Arduino 的数字引脚可以输出高电平 **5v** 也可以输出低电平 **0V**。在上次的实验中，如果将 LED 端子连接到电源负极 **GND**，LED 就没有电流通过，也就不会发光。
通过程序控制 **Arduino** 的引脚输出高电平，或者低电平，就可以控制 LED 的亮和灭，相当于用程序将 LED 端子接到了电源正极，或者电源负极。

部署

1. 将 LED1 端子接在 OSEPP UNO 的 13 号引脚上。



2. usb 一端连电脑，一端连接在 OSEPP UNO 的主板接口上。

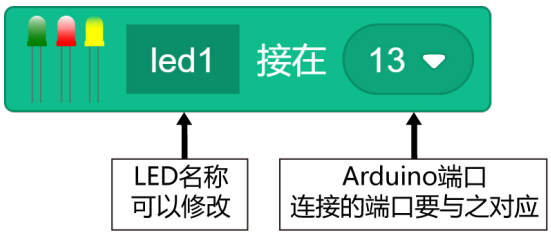
程序搭建

1. 在开始编程之前，我们先来了解 Arduino 程序。
最基本的 Arduino 程序由 setup 和 loop 两部分组成。

开机运行：void setup() 函数
setup 中的代码只在通电或者复位时执行一次，用于初始化变量，配置针脚的输出 / 输入类型 , 配置串口。

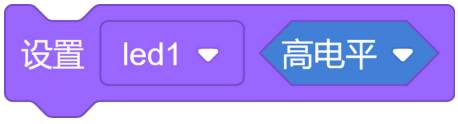
重复执行：void loop() 函数
loop 中的代码将会不断地被重复执行，通常程序的功能将在这里实现。

2. 在左侧目录区中的**输出模块**内找到 **LED 模块**积木：



将积木放入工作区空白区域。表示您有一个 LED，接在 13 号引脚。
右侧代码区中 `void setup()` 函数内会自动生成代码 `pinMode(13,OUTPUT)`，表示将控制板的 13 号引脚设置为输出模式，以驱动 LED。
定义模块类的积木只需要放置到工作区中就会产生代码。

3. 这时目录区中的会出现对应**设置 LED** 积木：



设置 LED 状态积木：**LED1** 对应上图的 **LED 模块**积木的名称。如果有多个 LED，请点击名称处的三角箭头，从拉列表选择 所要设置的 LED。
LED 状态设置有高电平和低电平两种模式，对应代码：

```
digitalWrite(13,HIGH)//13 号引脚输出高电平
digitalWrite(13,LOW)//13 号引脚输出低电平
```

此类形状的积木，需要放入开机运行或者重复执行中，才会产生代码。

4. 目前的程序将连接到 LED 的引脚 , 设置为输出模式并输出高电平，LED 将会被点亮。您还需要添加更多的积木 / 程序，目录区的选择硬件，积木区中拖出一个暂停积木到设置 LED 积木的下面。

暂停积木：将程序暂停一段时间。代码：

```
delay(1000);
```

是将程序的执行暂停一段时间 (单位为毫秒)，1000 毫秒 =1 秒。

并在设置 LED 积木上点击鼠标右键（触摸屏为长按），选择复制，将复制出来的积木连接到程序中。

最终的程序如图：

oseppBlock 程序

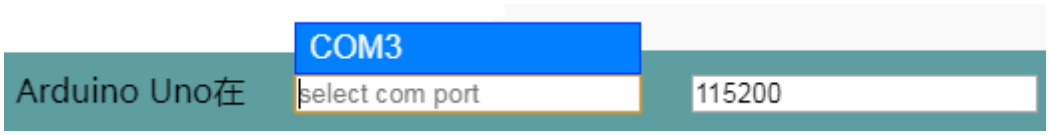
代码区中完整的

Arduino 程序

```
void setup()
{
    //led1
    pinMode(13, OUTPUT);          // 定义 13 号引脚为输出模式
}
void loop()
{
    digitalWrite(13, HIGH);     //13 号引脚输出高电平
    delay(1000);                // 延时 1000 毫秒
    digitalWrite(13, LOW);      //13 号引脚输出低电平
    delay(1000);                // 延时 1000 毫秒
}
```

上传程序

软件当前窗口下方选择端口号，



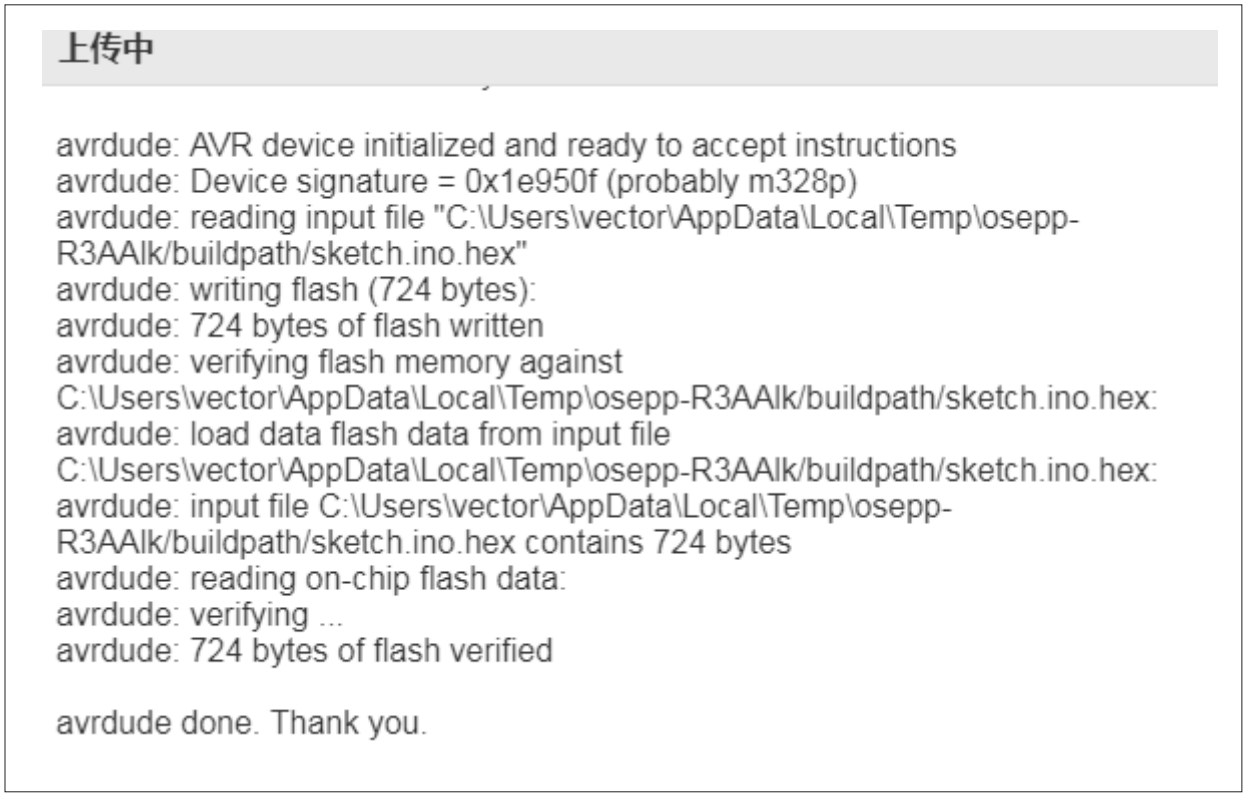
注意：您电脑上并不一定显示的是 COM3，也可能是其他数字。

然后点右上角的

上传按钮上传程序，这时会弹出一个窗口，

当窗口最后内容出现 `...Thank you.` 字样时，表示程序已经上传完成并写入到 **OSEPP UNO** 控制板中。

如下图：



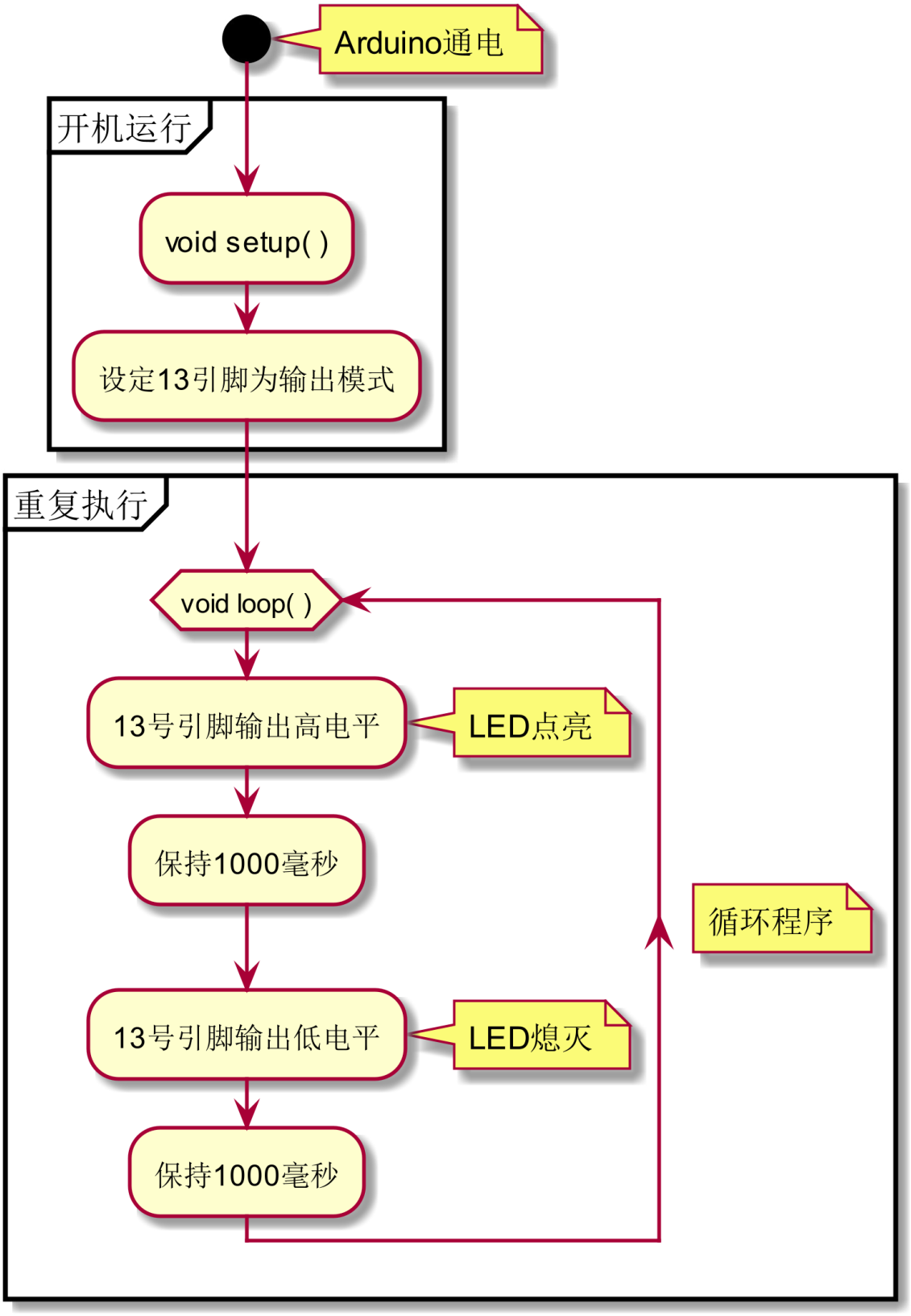
上传成功后可以关闭此窗口。

运行结果

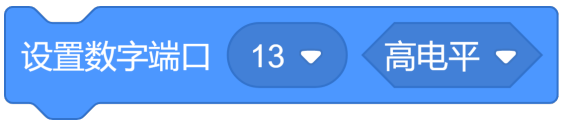
这时主板上的 LED 就会按照我们程序设定的模式点亮一秒，然后熄灭一秒，不停循环。

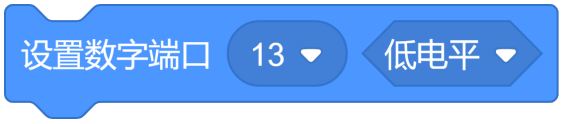
解析

控制板按照程序执行，运行流程如下图：



数字引脚输出模式：

<table>
<tr>
<td>


代码：
<code>digitalWrite(13,HIGH);</code>
</td>
<td rowspan="2">
<code>digitalWrite</code> 是一个函数，其作用为设置引脚的输出电压为高 \ 低电平。
该函数无返回值，有两个参数 <code>pin</code> 和 <code>value</code>，<code>pin</code> 参数表示所要设置的引脚，<code>value</code> 参数表示输出的电压 (<code>HIGH</code> 为高电平 <code>LOW</code> 为低电平)。
在使用 <code>digitalWrite(pin, value)</code> 函数之前要将引脚设置为输出 <code>OUTPUT</code> 模式。
</td>
</tr>
<tr>
<td>


代码：
<code>digitalWrite(13,LOW);</code>
</td>
</tr>
</table>

当 **Arduino** 的**数字引脚**设置为输出模式 `OUTPUT` 时，只能输出两种状态：高电平 `HIGH` 或者低电平 `LOW`。通常将这种输出称为**数字输出** (对于两种状态，有时称为二进制)。

这两个状态通常称为高电平 `HIGH` 和低电平 `LOW`。高电平 `HIGH` 等于说“这里有电压！”，低电平 `LOW` 表示“这个引脚上没有电压！”。

当您使用 `digitalWrite()` 的命令将 `OUTPUT` 引脚设置为高电平 `HIGH` 时，相当于在芯片内部把引脚连接到了电源正极。测量引脚与电源负极之间的电压，电压表显示 `5V`。

当您将 `OUTPUT` 引脚设为低电平 `LOW` 时，引脚相当于连接到了电源负极，再次测量引脚与电源负极，电压表没有输出。而测量电源正极与引脚之间的电压，电压表显示 `5V`。

按钮开关

按钮 (Button) 开关是一种电子开关，其内部结构是靠金属弹片受力变化来实现通断。

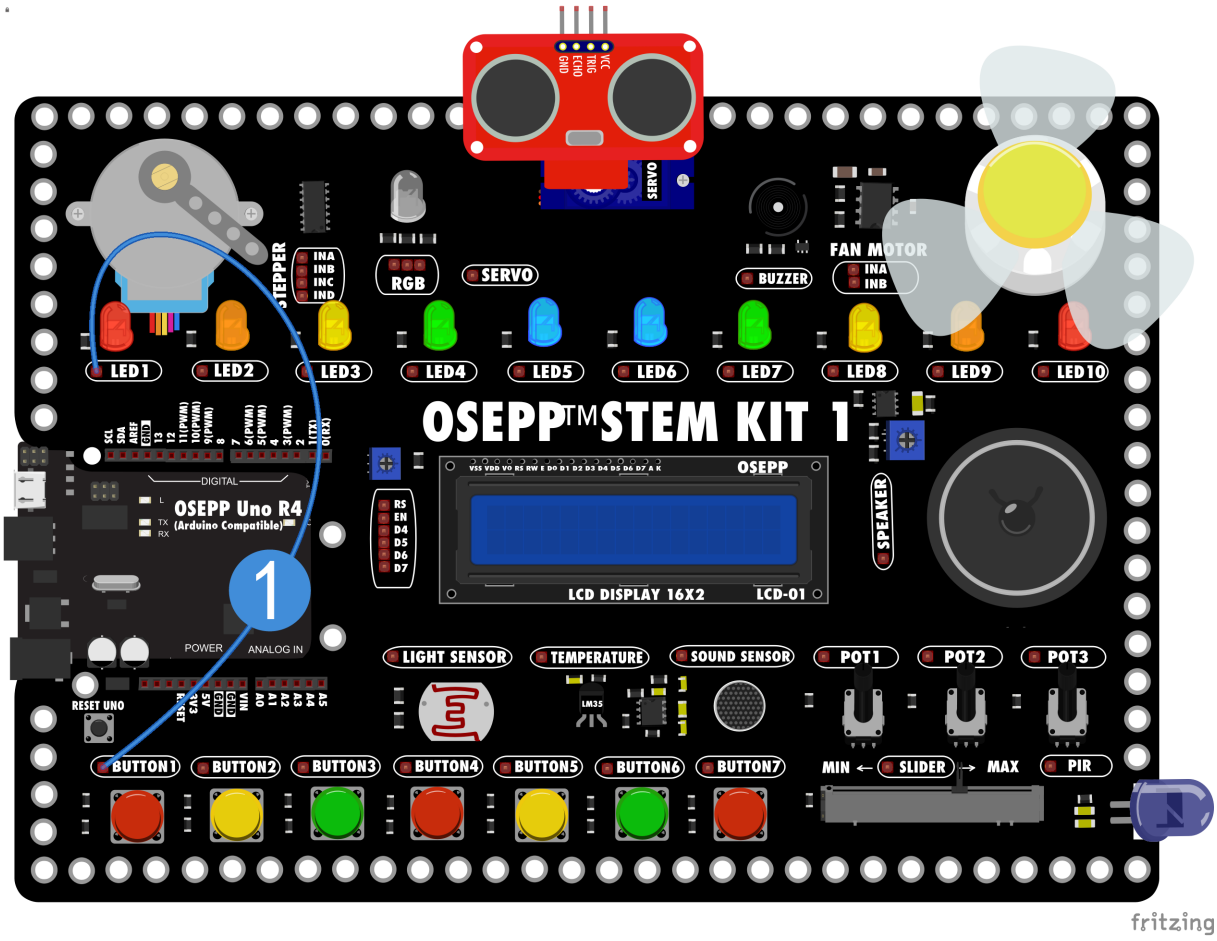
按钮开关只是开关的一种。开关在我们的日常生活中随处可见，我们家里的每一盏灯都是有一个或者两个开关来控制，每一个电器都会有一个开关来控制电源的通断。

应用 1：按钮开关控制 LED

在 STEM KIT 学习板上面有七个按钮开关，下面实验怎样用学习板上面的开关来控制 LED。

部署

将按钮 Button1 的端子接到 LED1 的端子上。



运行结果

STEM KIT 学习板通电后 LED 就被点亮，当按钮 (Button1) 按下时 LED 熄灭，松开时 LED 点亮。

解析

这看起来和我们平常认识的按下就点亮，松开时熄灭的效果相反。这是 **STEM KIT** 学习板设计的电路造成的，为了使用起来安全可靠，即使接错线了也不会损坏开发板和上面的元件，所以学习板开关采用了这种电路设计。

当我们把开关到 LED，并给学习板通电，LED 就会有电流通过。开关按下时，电流通过开关直接流到负极，LED 没有电流，所以 LED 不会点亮。

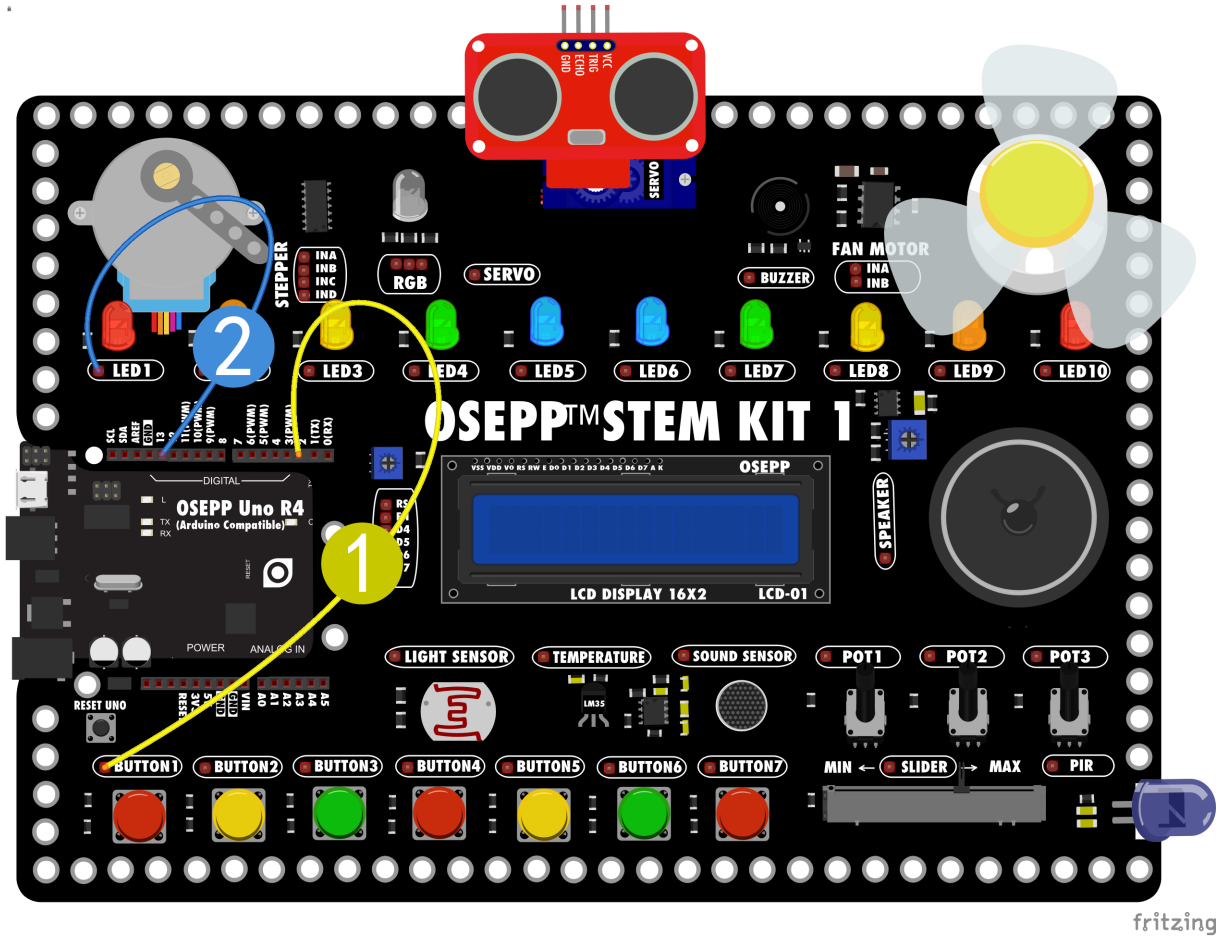
应用 2： 按钮开关连接 Arduino 控制 LED

我们可以把 LED 接到 **Arduino** 上面，也可以把开关接到 **Arduino** 上面。LED 是输出器件，能表示输出的状态。而开关是输入器件，**Arduino** 可以通过程序判断开关的状态。
本实验通过程序读取按钮的状态再控制 LED 的输出状态。

部署

1. 把按钮 Button1 连接在 **OSEPP UNO** 的 2 号引脚，引脚 0 和 1 用于与计算机通信和程序下载，通常不在应用中使用它们。

2. 把 LED1 连接在 **OSEPP UNO** 的 13 号引脚。

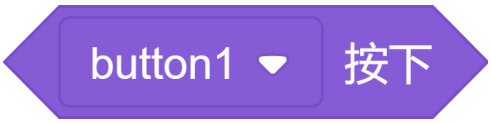


程序搭建

1. 在左侧的**积木区**模块里面选择**输入模块**里面的开关

	按钮开关积木，左图表示 Button1 接在 **Arduino** 的 2 号引脚。

2. 拖入到右边**工作区**空白处，这时在这个模块下面就出现了

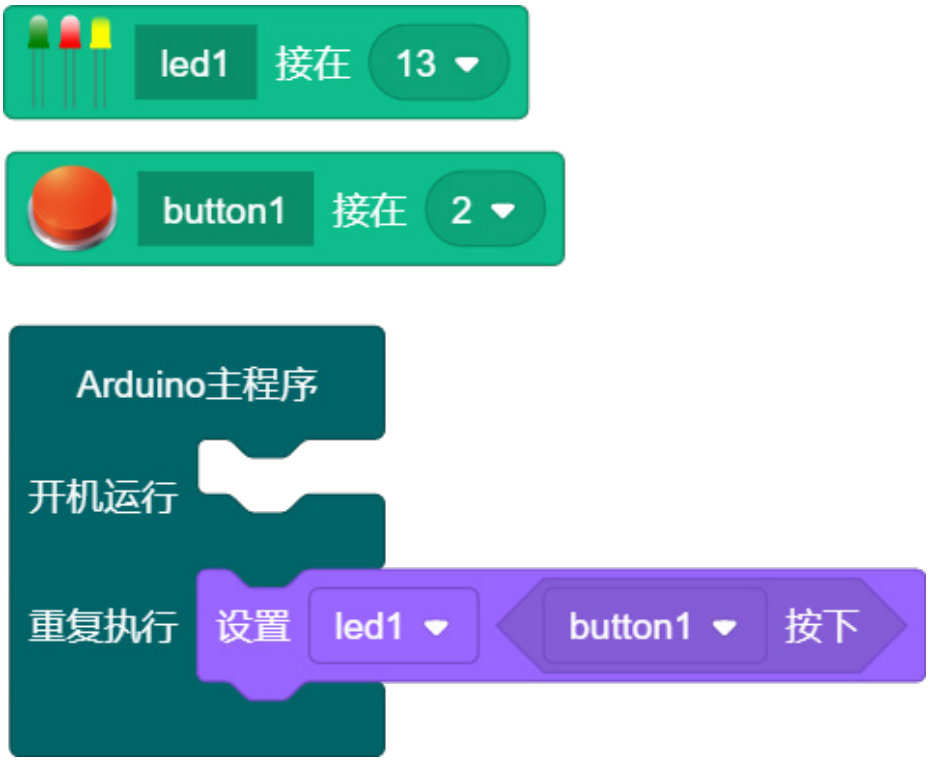
	按钮开关状态积木，左图表示 Button1 开关按下时状态。工作区中需要有按钮积木模块，此积木才会出现。

3. 将这个图标拖入到 LED1 的电平选择框里面。

	此时积木表示开关按下时，点亮 LED。程序通过判断开关的状态来决定是否点亮 LED。

最终的程序：

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //button1
    pinMode(2, INPUT); // 定义 2 号引脚为输入模式
    //led1
    pinMode(13, OUTPUT); // 定义 13 号引脚为输出模式
}

void loop()
{
    digitalWrite(13, digitalRead(2) == LOW);
    // 开关按下时 LED 点亮
}
```

运行结果

按钮 (Button1) 按下时 LED 发光，松开时 LED 熄灭。

解析

数字引脚输入状态：

当 **Arduino** 的数字引脚设置为输入时 INPUT ，只能读取两种状态：高电平 HIGH 或者低电平 LOW。

函数 digitalRead() 用来监视输入引脚 INPUT 的电压，如果电压是高电平，它将返回 HIGH，如果是低电平就返回 LOW。但实际中，一般来说高于芯片供电电压的一半就被认为是高电平，而小于这个电压就被认为是低电平。如果该引脚没连接 (悬空)，那从 digitalRead() 返回的值是不确定的，可能是 HIGH 也可能是 LOW。

定义输入还有另一种模式：

	定义引脚为带上拉电阻的输入模式时，芯片内部的上拉电阻将会连接到该引脚，从而为引脚提供默认高电平状态。

代码：

```
void setup()
{
    pinMode(3, INTPUT_PULLUP)    // 定义 3 号引脚为带上拉电阻的输入模式
}
```

应用 3 延时开关

延时开关是为了节约电力资源而开发的一种新型的自动延时电子开关，省电、方便。主要用于楼梯间，走廊过道等场所。下面我们就来制作一个延时开关。

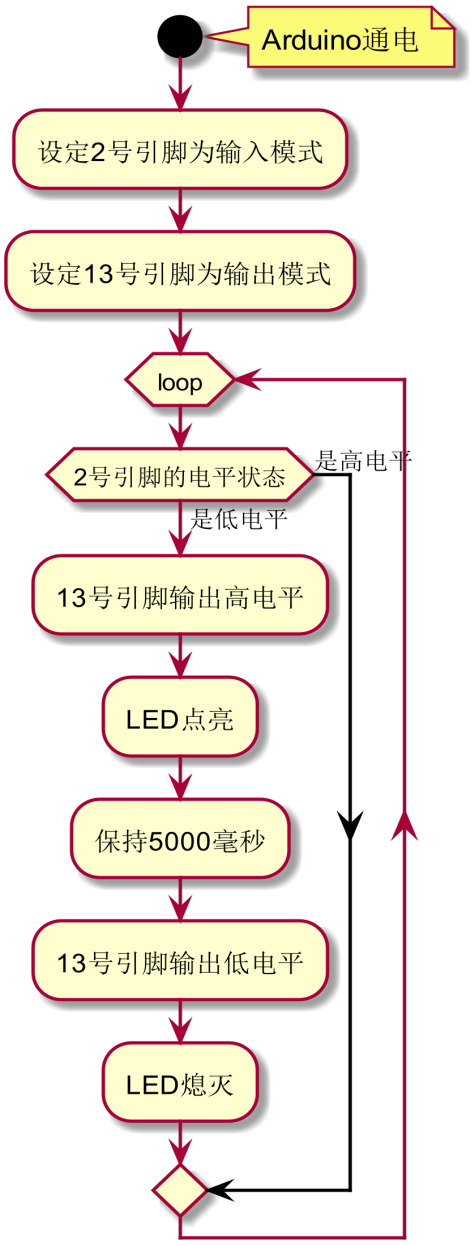
本实验通过程序实现：按钮 Button1 按下，点亮 LED，5 秒后熄灭。

部署

继续使用上一个实验中的线路，只需修改一下程序。

程序搭建

我们首先来了解一下 **Arduino** 的程序工作流程：



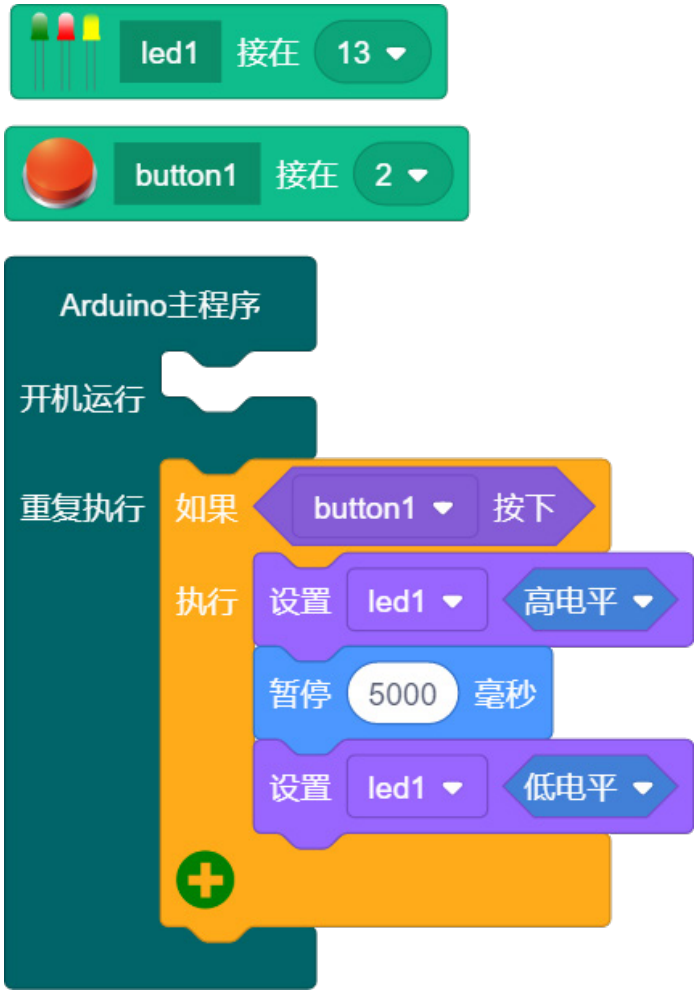
开关有两种状态按下时和没按时。按下时是**低电平**，没按时是**高电平**（1. 学习板上的开关接入了上拉电阻，开关没有被按下时默认是**高电平**）。程序可以通过判定开关的这两种状态来决定是否点亮 LED。

开关接到 2 号引脚，当开关按下时，2 号引脚是**低电平**，此时程序执行点亮 LED，也就是 13 号引脚输出**高电平**。点亮之后程序暂停 5 秒，然后 13 号引脚输出**低电平**，LED 熄灭。

如果开关没有被按下，2 号引脚始终都是**高电平**。程序会直接跳过该段代码。

依照流程图，最终得到程序

oseppBlock 程序



Arduino 程序

```
void setup() {
    // button1
    pinMode(2, INPUT);            // 定义 2 号引脚为输入模式；
    // led1
    pinMode(13, OUTPUT);          // 定义 13 号引脚为输出模式；
}
void loop() {
    if (digitalRead(2) == LOW) {    // 如果开关被按下
        digitalWrite(13, HIGH);     // 13 号引脚输出高电平
        delay(5000);                // 延时 5000 毫秒
        digitalWrite(13, LOW);      // 13 号引脚转换为低电平
    }
}
```

运行结果

如果开关按下时，LED 就会点亮 5 秒，然后熄灭，直至下一次开关按下 LED 才会点亮。

解析

程序判断语句，根据条件采取行动：

判断语句 `if` ，如果 ... 执行 ....。

如果按钮开关按下就点亮 LED, 点亮 LED 就是执行动作。

`if` 语句是指编程语言中用来判定所给定的条件是否满足，根据判定的结果（真或假）决定是否执行下面的程序，如果为真就执行，否则就跳过此部分。

3 电位器

电位器是一种可变电阻器。通常是由电阻体与转动或滑动系统组成，即靠一个动触点在电阻体上移动，获得部分电压输出。

电位器应用很广泛，在很多产品上都会有所应用，比如音响上的音量调节，风扇的速度调节，灯光的亮度调节等等。

电位器的作用——调节电压和电流的大小。

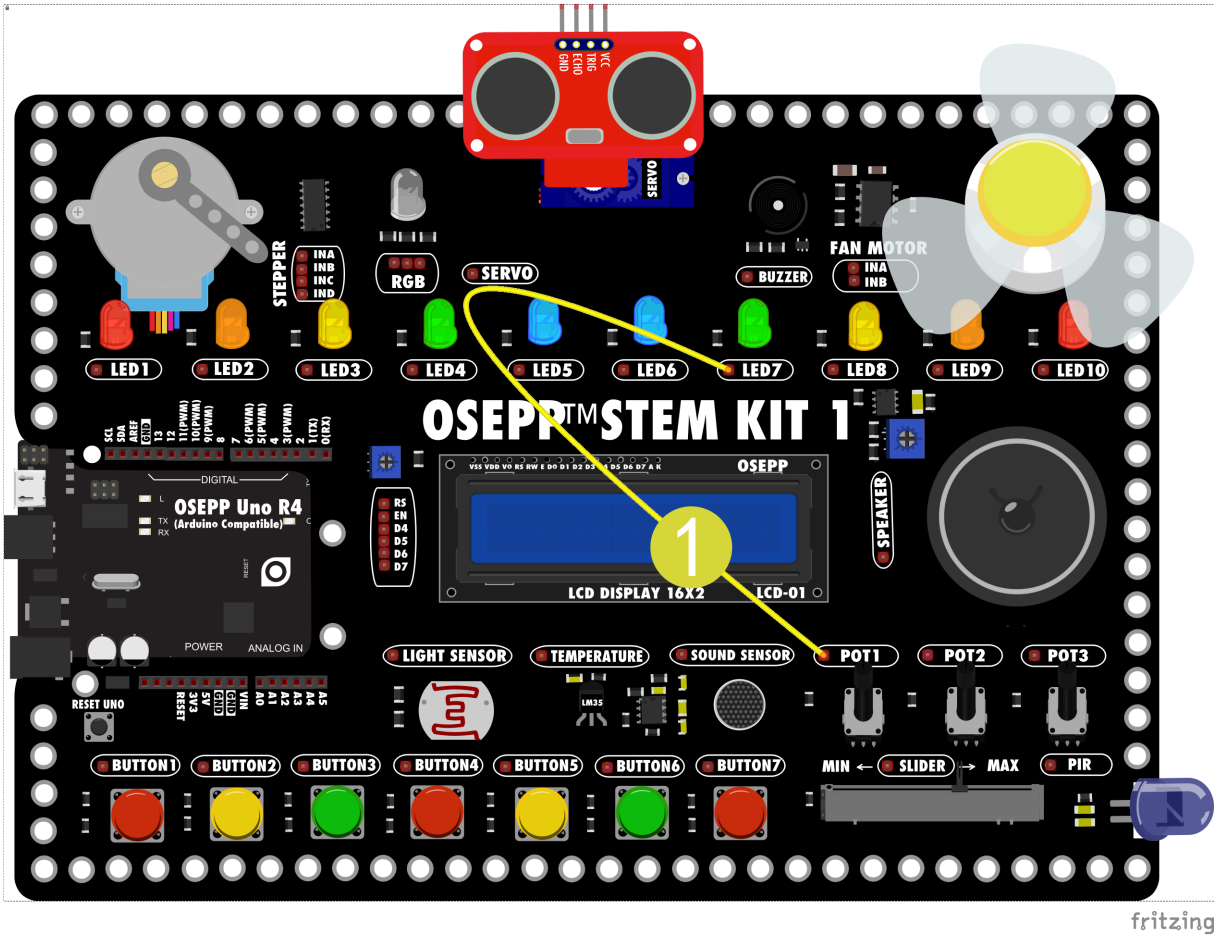
电位器的结构特点——电位器的电阻体有两个固定端，通过手动调节转轴或滑柄，改变动触点在电阻体上的位置，则改变了动触点与任一个固定端之间的电阻值，从而改变了电压与电流的大小。

应用 1 电位器调节 LED

本实验中将电位器直接连接 LED，然后转动电位器，从而能直观地了解电位器的特性。

部署

用一条母对母的杜邦线将 LED7 的端子和电位器 POT1 的端子连接起来。



运行结果

当您转动电位器 POT1 时，LED 的亮度也随之改变。

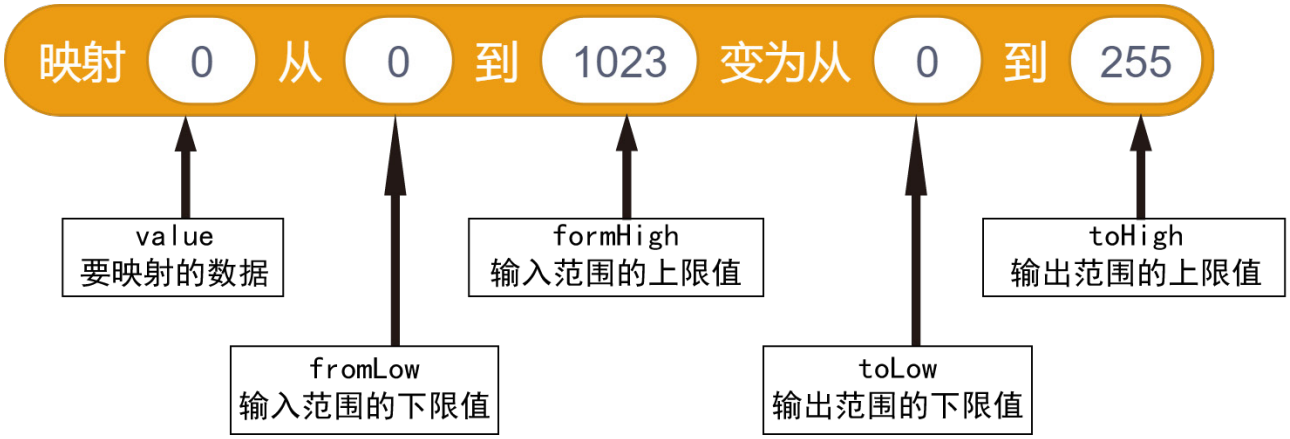
通过此次实验，让我们了解了电位器。电位器简单点解释就是可变电阻器，调节电位器就是调节了线路当中的电阻，从而改变了加载在负载端的电压和电流。

应用 2 电位器连接 Arduino

本实验将电位器连接到 **Arduino**，通过**映射函数**来调节 LED 的亮度。

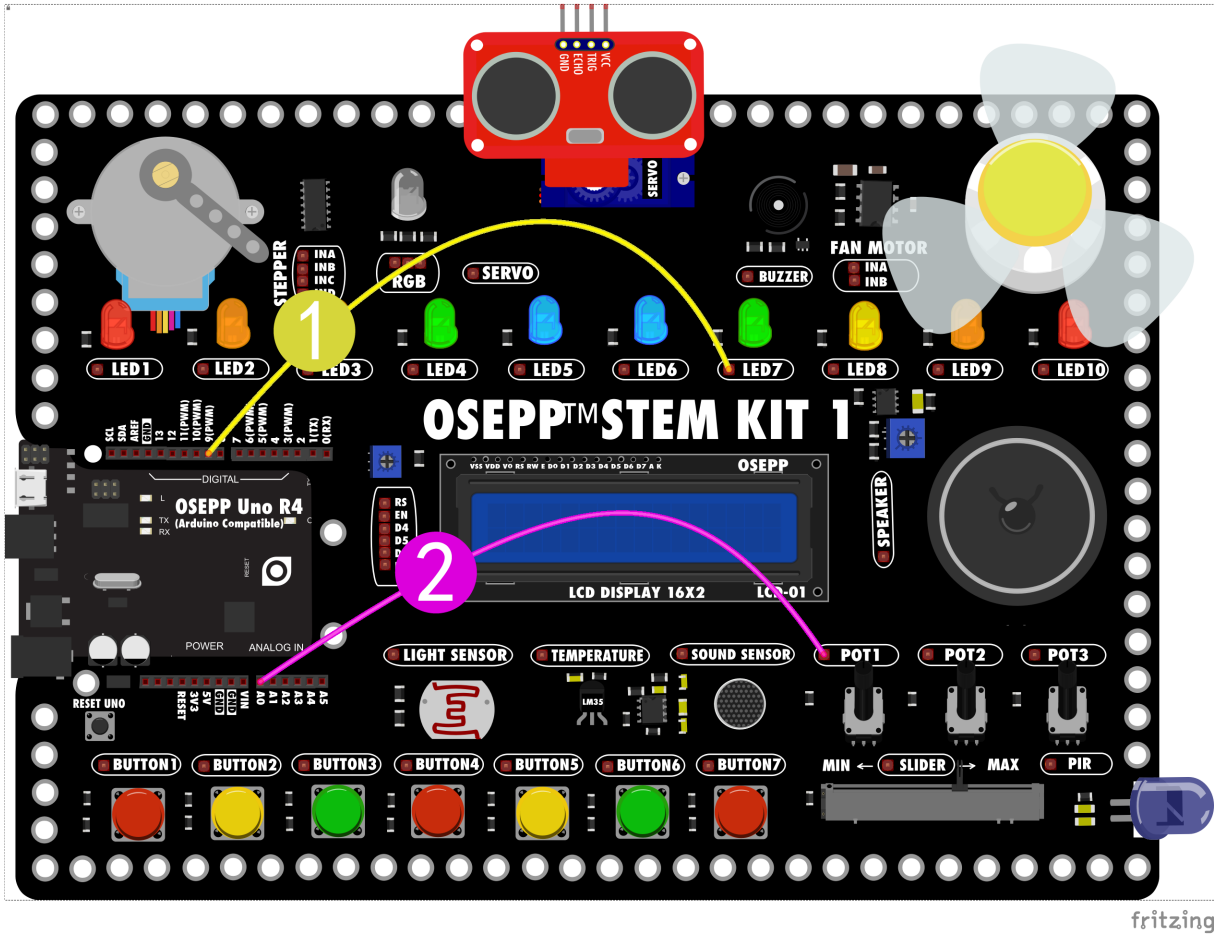
映射函数： map()
描述：将数据从一个范围映射到另一个范围。
函数原型：map(value, fromLow, fromHigh, toLow, toHigh)

积木图标：



部署

1. 把 LED7 接到 OSEPP UNO 的 9 号引脚。
2. 电位器 POT1 接到 OSEPP UNO 的 A0 号引脚。

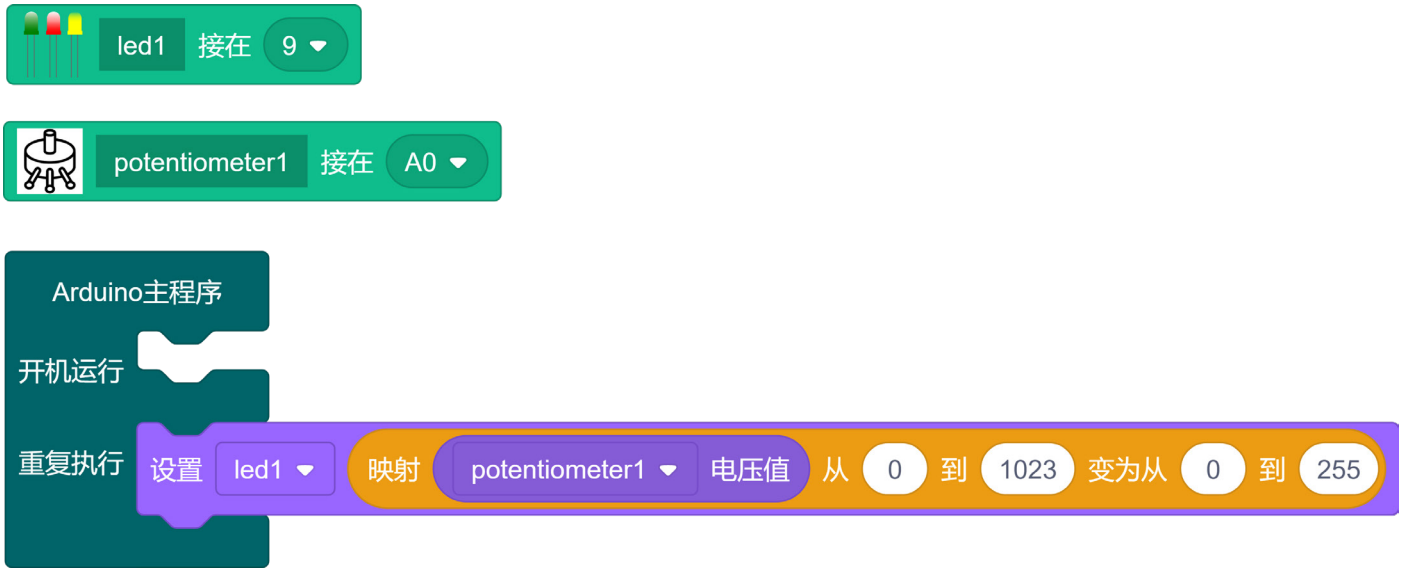


程序搭建

oseppBlock 积木知识

potentiometer1 接在 A0	电位器积木，定义名称和连接引脚。 电位器是模拟输入器件，只连接在 A0-A7 引脚。
potentiometer1 电压值	电位器读取数值积木，输出数值在 0~1023 之间。

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //potentiometer1
    pinMode(A0, INPUT);      // 定义 A0 引脚为输入模式
    //led1
    pinMode(9, OUTPUT);      // 定义 9 号引脚为输出模式
}

void loop()
{
    digitalWrite(9, map(analogRead(A0), 0, 1023, 0, 255));

                             // 把 A0 的值 0-1023 映射到 0-255
}
```

运行结果

转动电位器，LED 的亮度也会随之变化。电位器往左旋转，此时 LED 由亮到暗，往右旋转，LED 由暗到亮。

解析

模拟引脚

模拟引脚：A0-A7 ，Arduino 内置模数转换器 (ADC),A0-A7 引脚中的模拟量可以报告 0-1023 之间的值，该值映射到 0 伏至 VCC（电源正极电压），这里是 5V 的范围。
A0-A7 可以将采集到的 0-5V 表示为 0-1023 的数值，交由程序来处理。

函数 analogRead()

analogRead(pin) 用于读取模拟引脚 A0-A7 的模拟量的电压值，参数 pin 表示要获取模拟量电压值的引脚，该函数返回值为 0-1023 之间的一个整数。
0V 时值是 0，2.5V 时值是 512，5V 时值是 1023。

PWM 引脚

PWM 引脚：通常以 # 号和 * 号标注开头的数字引脚，脉冲宽度调制，它是利用微处理器的数字输出来控制模拟电路的一种技术。输出等级为 0~255。

Arduino UNO 有六个用于 PWM 的引脚 （数字引脚 3，5、6、9、10、11）。

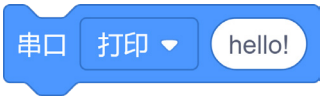
应用解析

PWM 的引脚输出等级为 0-255，A0 输入的数值是 0-1023。我们要把 0-1023 的数值映射到 0-255 来输出。

也可以通过改变**映射关系**来控制输入与输出的关系。比如我想电位器只拧到一半的时候 LED 达到最亮的值，那就把 0-1023 改成 0-512。如果只想 LED 达到一半亮度，就可以把后面 0-255 改为 0-128。

Arduino 串口打印

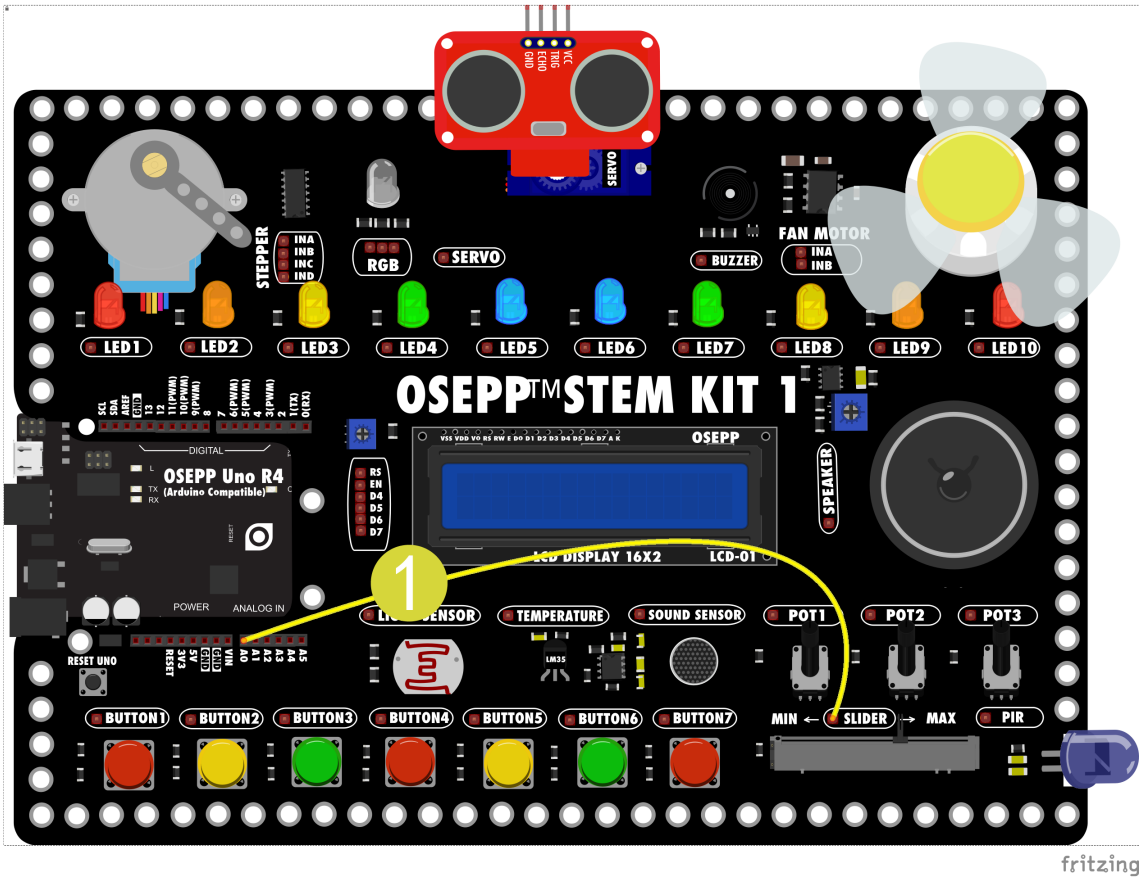
Arduino 上面有一个硬件叫做串口，它可以使用 **0** 号 (接收) 和 **1** 号引脚 (发送)，通过特定的方式，传递出信息。而电脑上一般使用的是 USB 接口，所以 **Arduino** 上还有一个芯片，专门把串口转换成 USB 接口，这样控制器就可以往串口上送出信息，再转成 USB 信号，然后电脑接收到信息，显示出来。

		
串口在使用前也需要先配置，主要是配置速度，称为波特率。配置指令是 `Serial.begin(speed)`，**speed** 是要设置的速度，通常设置为 **115200** 。	串口发出数据的指令是 `Serial.print(val)`，**val** 是要传送的信息，可以是文字或者数据。	`Serial.println(val)`, 在信息后添加换行符号，后面再发送的信息将显示在新的一行上。 可以通过选择框切换： 打印 / 打印并换行。

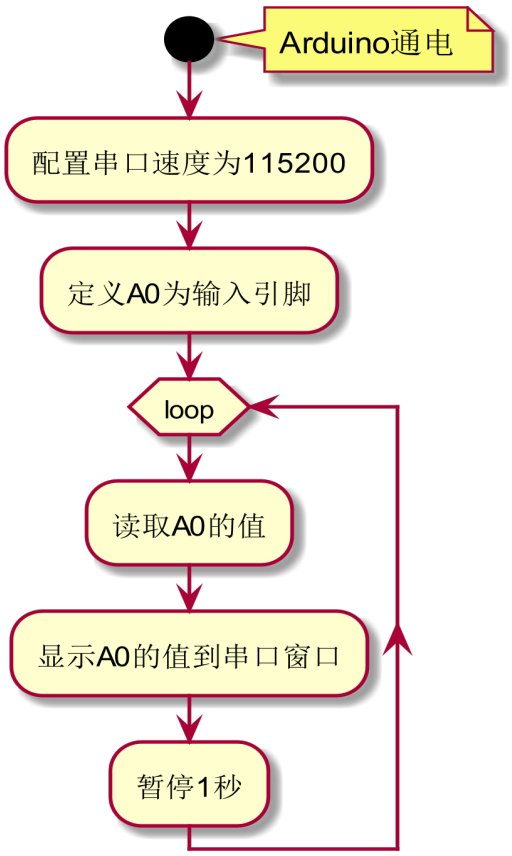
串口打印滑块电位器的值

部署

把滑块电位器 **Slider** 接到 **OSEPP UNO** 的 **A0** 引脚。滑块电位器也是电位器的一种。

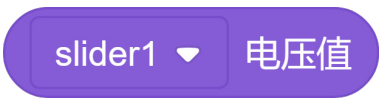


串口打印流程图



程序搭建

oseppBlock 积木认识

	滑块电位器积木，定义名称和连接引脚。 滑块电位器是模拟输入器件，只连接在 A0-A7 引脚。
	滑块电位器读取数值积木，输出数值在 0~1023 之间。

oseppBlock 程序

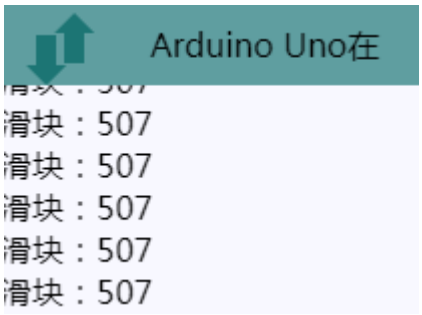
Arduino 程序

```
void setup()
{
    //slider1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    Serial.begin(115200); // 设置串口波特率
}

void loop()
{
    Serial.print(" 滑块: ");          // 打印括号内文字
    Serial.println(analogRead(A0)); // 打印 A0 的值并换行
    delay(1000);                    // 延时 1000 毫秒
}
```

运行结果

上传好程序后，点击下面的 图标。

当图标变成 ，就可以在下面窗口看到打印输出数据了。



解析

串口打印 (串行监视器)，能够从微控制器报告结果并显示出来。使用串行监视器，可以获取有关传感器状态的信息，并可以了解电路中的运行情况以及运行时的代码。

LCD 显示屏

LCD1602 液晶显示屏是广泛使用的一种字符型液晶显示模块。它是由字符型液晶显示屏，控制驱动主电路及其扩展驱动电路，以及少量电阻、电容元件和结构件等装配在 PCB 板上而组成。

字符型液晶显示模块是一种专门用于显示字母、数字和符号等的点阵式 LCD，常用 16×1，16×2，20×2 和 40×2 等的模块。

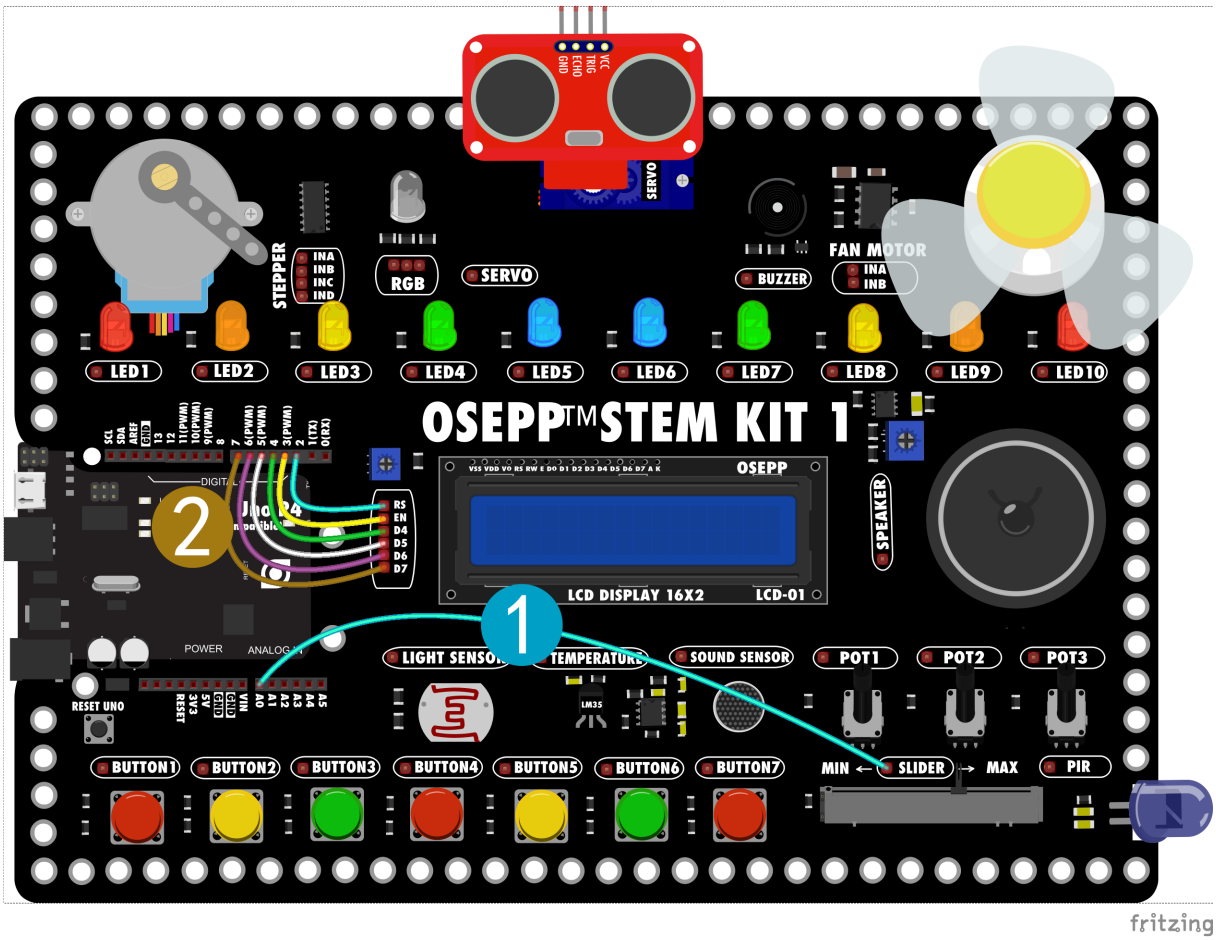
STEM KIT 学习板用的 **LCD1602** 液晶显示屏，可以显示 16x2 个字符。(注意：一个汉字将占用 2 个字符宽度，但是显示汉字需要字库，在没有字库支持的情况下只能显示英文。)

LCD 显示滑块电位器的值

如果我们需要知道电位器的值，可以用 LCD 显示屏来显示出来。只需简单的配置，就能让 LCD 显示字符。

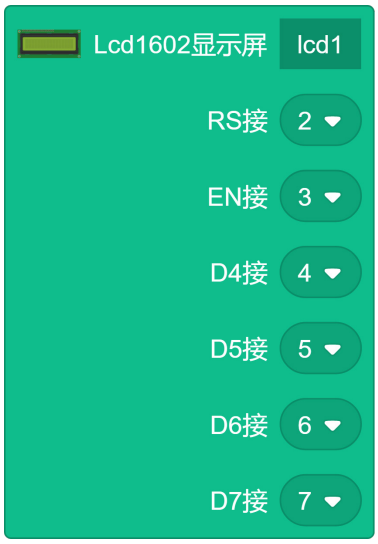
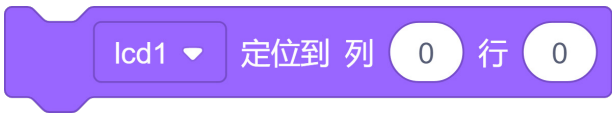
部署

1. 把滑块电位器 `Slider` 接到 **OSEPP UNO** 的 `A0` 引脚。
2.LCD 的连接端口 `RS-D7` 分别连接到 **OSEPP UNO** 的 `2~7` 号引脚。



程序搭建

oseppBlock 积木知识

	LCD 积木，定义显示屏的名称引脚号。对应的端口名称要与主板的端口名称对应，引脚号码可以更改，但连接时要对应名称连接。
	LCD 清屏积木，就是把屏幕所有的显示内容都清除。类似于我们要在黑板写字时，把黑板都擦干净。
	显示定位积木，就是指定接下来要显示字符的起始位置。LCD 有 16 列 2 行对应的是 0-15 列，0-1 行。 如果从第一行第一列显示就是 0,0 ，第二行第一列显示就是 0,1 。
	LCD 要显示的内容，这里显示的是字符 "**hello**!", 如果要显示滑块电位器的值，就把电位器的电压值图标拖入到框内。

oseppBlock 程序

Arduino 程序

```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 的引脚

void setup()
{
    lcd1.begin(16, 2); //LCD 初始化
    //slider1
    pinMode(A0, INPUT); // 定义滑块电位器连接的引脚为输入模式
}

void loop()
{
    lcd1.clear();                // 清屏
    lcd1.setCursor(0, 0);        // 设置光标位置在 0 行，0 列（左上角）
    lcd1.print(analogRead(A0)); // 显示滑块电位器的值
    delay(1000);                 // 延时 1000 毫秒
}
```

运行结果

LCD 的显示屏上面会显示滑块电位器的值，数值在 0-1023 之间。调节滑块电位器，这个数值也会跟着变化。我们在程序后面加了一个延时 1 秒，延时这 1 秒内程序会暂停，所以我们调节起来会感觉反应很慢。如果显示不清晰，可以用螺丝刀调节 LCD 接线端口上边的电位器，直到显示字符清晰。

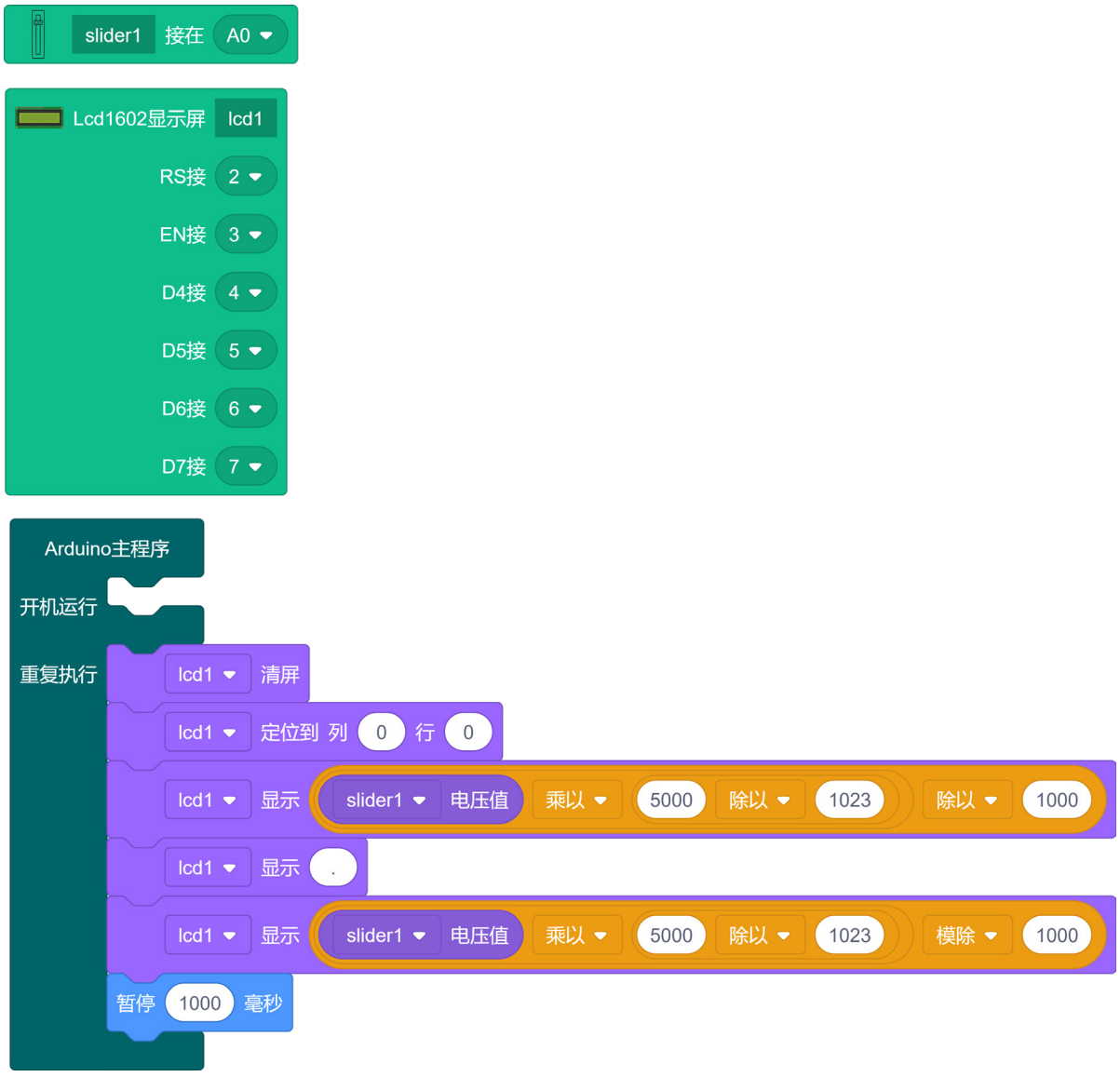
解析

此时 LCD 每秒就会刷新一次滑块的电压值。我们看到 LCD 显示的是 0-1023 的数字，就是 **Arduino** 把从 A0 读取到的电压值 0-5V 映射成 0-1023 来显示。如果要显示成 0-5V 的电压，需要通过程序计算，下面展示两种计算方法：

方法一：

用 1023 的值表示 5000mV，每个单位代表 5000/1023mV（即 4.89mV）analogRead(A0) 的值乘以每个单位（即 4.89mV）就是实际的毫伏值， 整除 1000 舍掉后面的小数点就是整伏数，后面用模除算出的余数就是 mV 部分。

oseppBlock 程序



Arduino 程序

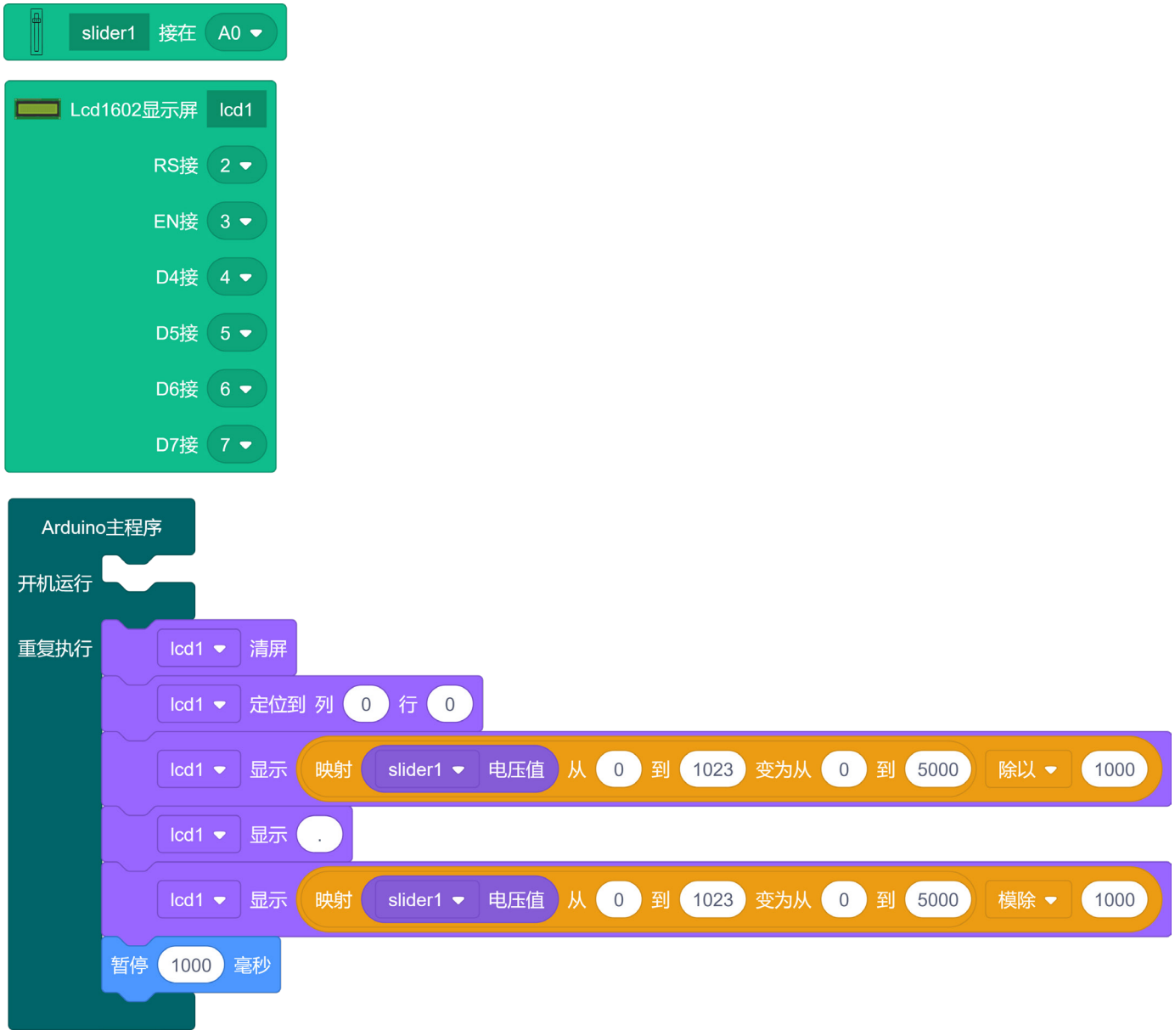
```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 的引脚

void setup()
{
    lcd1.begin(16, 2); //LCD 初始化
    //slider1
    pinMode(A0, INPUT); // 定义滑块电位器连接的引脚为输入模式
}

void loop()
{
    lcd1.clear();                                          // 清屏
    lcd1.setCursor(0, 0);                                  // 显示光标定位
    lcd1.print((analogRead(A0) * (5000 / 1023)) / 1000); // 计算整数部分
    lcd1.print(".");                                       // 添加一个小数点
    lcd1.print((analogRead(A0) * (5000 / 1023)) % 1000); // 用模除计算小数部分
    delay(1000);                                           // 延时 1000 毫秒
}
```

这时 LCD 显示的就是滑块的实际电压值。

方法二：

直接把 analogRead(A0) 的值映射到毫伏，整除 1000 舍掉后面的小数点就是整伏数，后面用模除算出的余数就是 mV 部分。

oseppBlock 程序



Arduino 程序

```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 的引脚

void setup()
{
    lcd1.begin(16, 2); //LCD 初始化
    //slider1
    pinMode(A0, INPUT); // 定义滑块电位器连接的引脚为输入模式
}

void loop()
{
    lcd1.clear();                                                    // 清屏
    lcd1.setCursor(0, 0);                                            // 显示光标定位
    lcd1.print(map(analogRead(A0), 0, 1023, 0, 5000) / 1000); // 计算整数部分
    lcd1.print(".");                                                 // 添加一个小数点
    lcd1.print(map(analogRead(A0), 0, 1023, 0, 5000) % 1000); // 用模除计算小数部分
    delay(1000);                                                     // 延时 1000 毫秒
}
```

当没有设定特殊变量时，**Arduino** 程序都是以整数进行运算的。为了计算后面小数部分我们用了**模除 (%)**，模除的结果的是一个数除以另一个数的余数。

Arduino 的数学运算

	两个数相加 $1+2=3$
	第一个数减去第二个数 $2-1=1$
	两个数相乘 $1*2=2$
	第一个数除以第二个数 $2/1=2$
	第一个数除以第二个数的余数 $5/3=1...2$结果为2

光线传感器

光线传感器是根据光电效应的原理起作用的。所谓光电效应，就是指某些特殊的物质在吸收了光线后能够将光能转换为电能的现象。

光电效应可以分为外光电效应和内光电效应两种。
外光电效应指的是在光线照射下，电子能够从物质的内部向外发射而产生电力作用，光电管、光电倍增管都是基于外光电效应制成的原件。相应地，内光电效应则是发生在物质的内部，当光线照射到物质上时，使其内部的电阻率发生改变，从而改变了电动势。光敏电阻、光电池等光电原件就是基于内光电效应制成的。

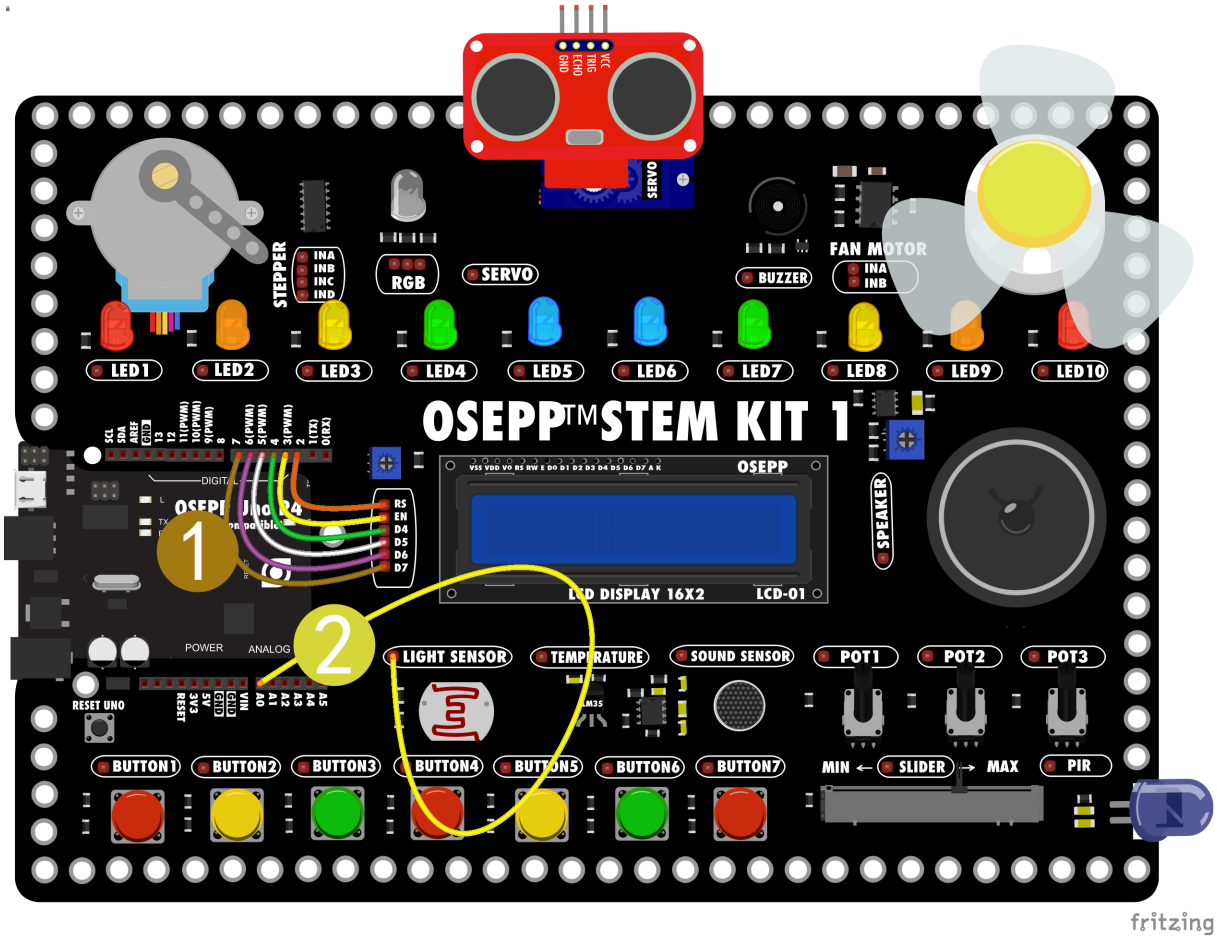
学习板上的光线传感器就是光敏电阻。

应用 1 LCD 显示光线传感器的值

光线照射到传感器上时，传感器的阻值发生了变化，加载到传感器上的电压也相应的产生变化，并反映在端子上。我们将使用程序读取这个变化的电压并显示在 LCD 上面。

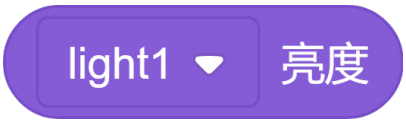
部署

1. LCD 的接口 RS-D7 分别连接到 OSEPP UNO 的 2~7 号引脚。
2. 光线传感器 Light Sensor 的端子接到 OSEPP UNO 的 A0 引脚。

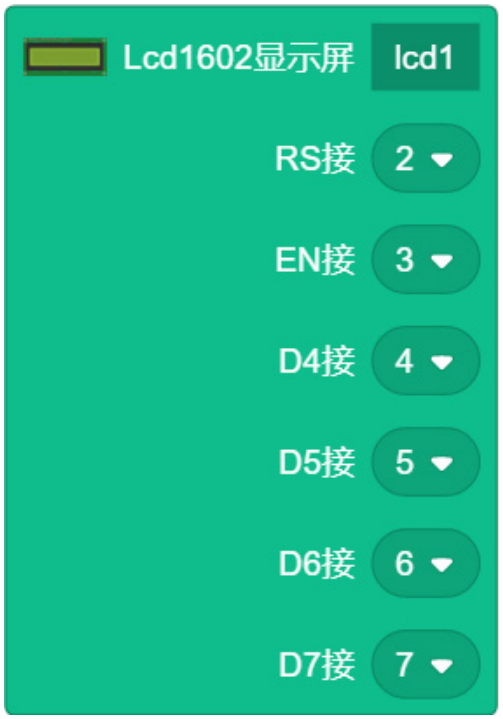


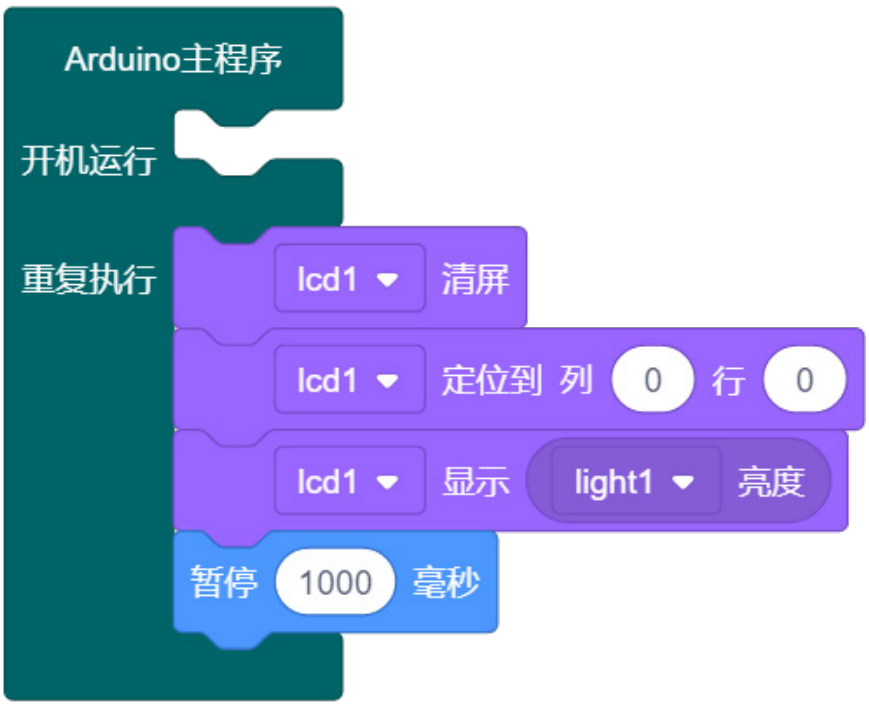
程序搭建

oseppBlock 积木知识

	光线传感器定义积木。设定传感器的名称和连接引脚，只能接在模拟输入端口 A0-A7。
	光线传感器亮度输出值积木。 输出值范围在 0-1023 之间。

oseppBlock 程序





Arduino 程序

```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚

void setup()
{
  lcd1.begin(16, 2); //LCD 初始化
  //light1
  pinMode(A0, INPUT); // 定义光线传感器引脚 A0 为输入模式
}

void loop()
{
  lcd1.clear();                // 清屏
  lcd1.setCursor(0, 0);        // 显示光标定位
  lcd1.print(analogRead(A0)); // 显示光线传感器的值
  delay(1000);                 // 延时 1000 毫秒
}
```

运行结果

此时遮挡光线传感器就会看到 LCD 显示的数值在变化。光线越弱数值越小，光线越强数值越大。

解析

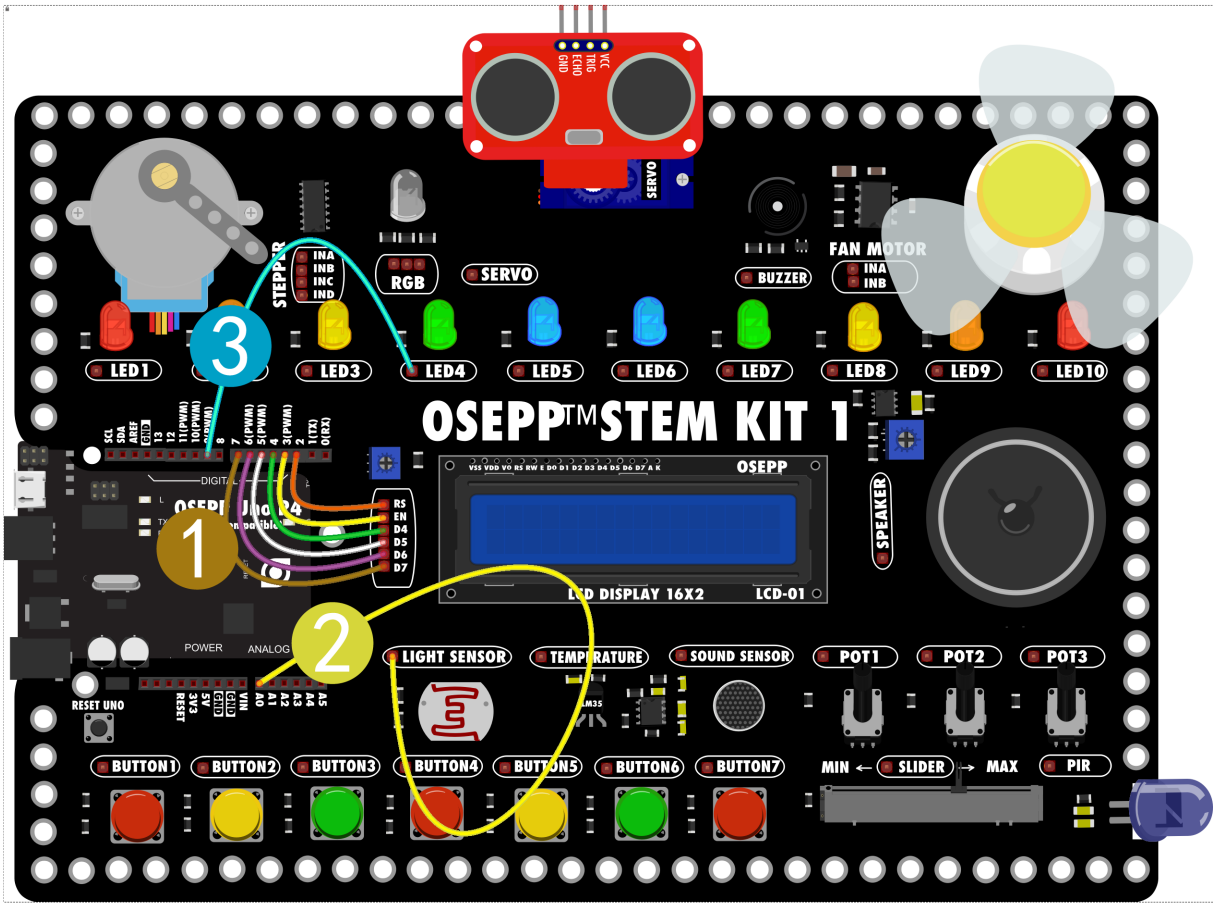
光敏电阻是一种电阻值随光照强度增加而下降的电阻，基于其内部光电效应，光照越强电阻值越小。

应用 2 光线传感器控制 LED 的亮度

光线传感器能感应光线的变化，并且把光线变化通过电压变化体现出来。那我们就可以利用这种变化来制作一个环境感应灯。如果光线越暗，LED 就越亮。光线充足，LED 就不会发光。

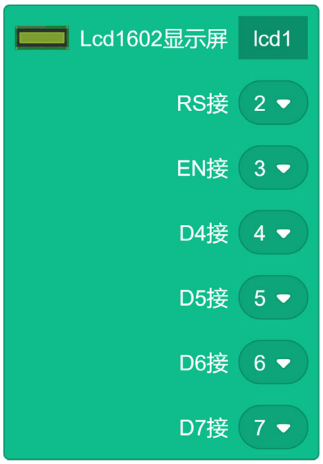
部署

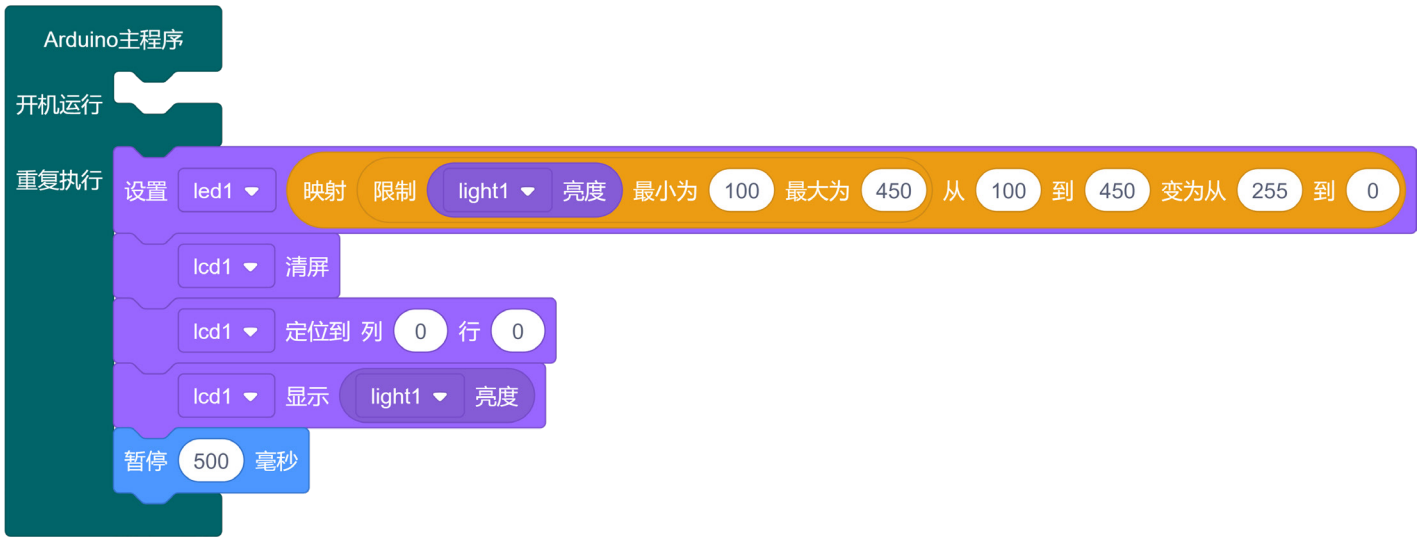
1.LCD 的接口 RS-D7 分别连接到 OSEPP UNO 的 2~7 号引脚。
2. 光线传感器 Light Sensor 的端子接到 OSEPP UNO 的 A0 引脚。
3.LED4 的端子连接到 OSEPP UNO 的 9 号引脚。



程序搭建

oseppBlock 程序





Arduino 程序

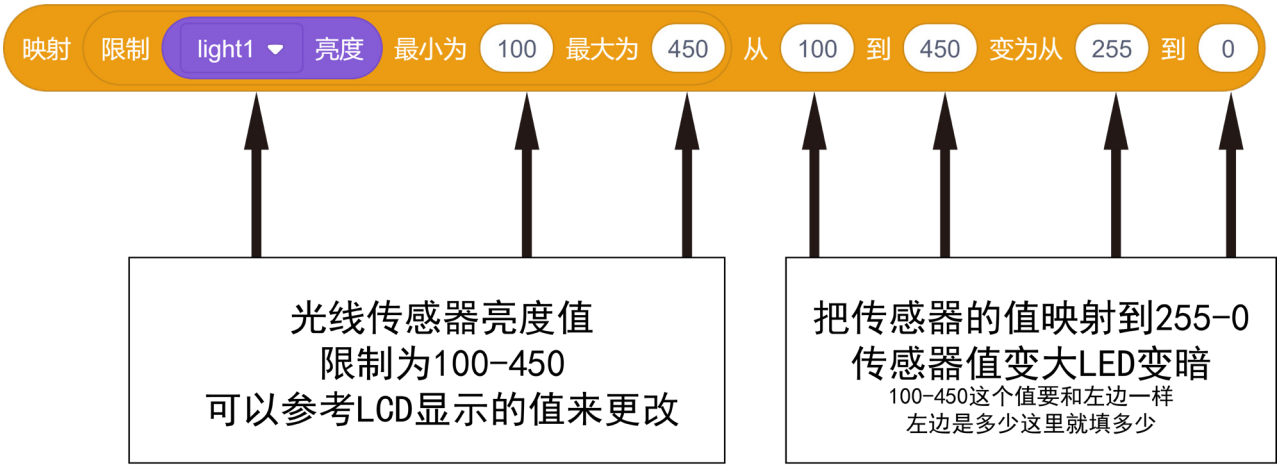
```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚

void setup()
{
  lcd1.begin(16, 2); //LCD 初始化
  //led1
  pinMode(9, OUTPUT); // 定义 LED 引脚
  //light1
  pinMode(A0, INPUT); // 定义光线传感器引脚
}

void loop()
{
  analogWrite(9, map(constrain(analogRead(A0), 100, 450), 100, 450, 255, 0));
  // 把光线传感器的值限制在一个范围，且把这个范围映射到 255-0 来驱动 LED
  lcd1.clear();                 // 清屏
  lcd1.setCursor(0, 0);         // 显示光标定位
  lcd1.print(analogRead(A0)); // 显示光线传感器的值
  delay(1000);                  // 延时 1000 毫秒
}
```

运行结果

传感器的光线被遮挡的时候，LED 就会变亮，移开遮挡物时 LED 就变暗了。这里映射的输出值填的是 255-0，由大到小。所以当光线越弱的时候 LED 显示就越亮，光线越强的时候，LED 显示就越暗。

解析

程序里面我们限制了光线传感器的值最小值是 100 最大值是 450，这是为了防止光线变化超出映射值范围时程序失效。当然你可以更改这个值，当光线最暗时，LCD 显示的是最小值，光线最亮时，LCD 显示的就是最大值，把这两个值填入积木里面。



限制一个值的范围积木，当这个值超出这个范围时，只返回最小值或最大值。

代码：

```
constrain(0,0,255);
```

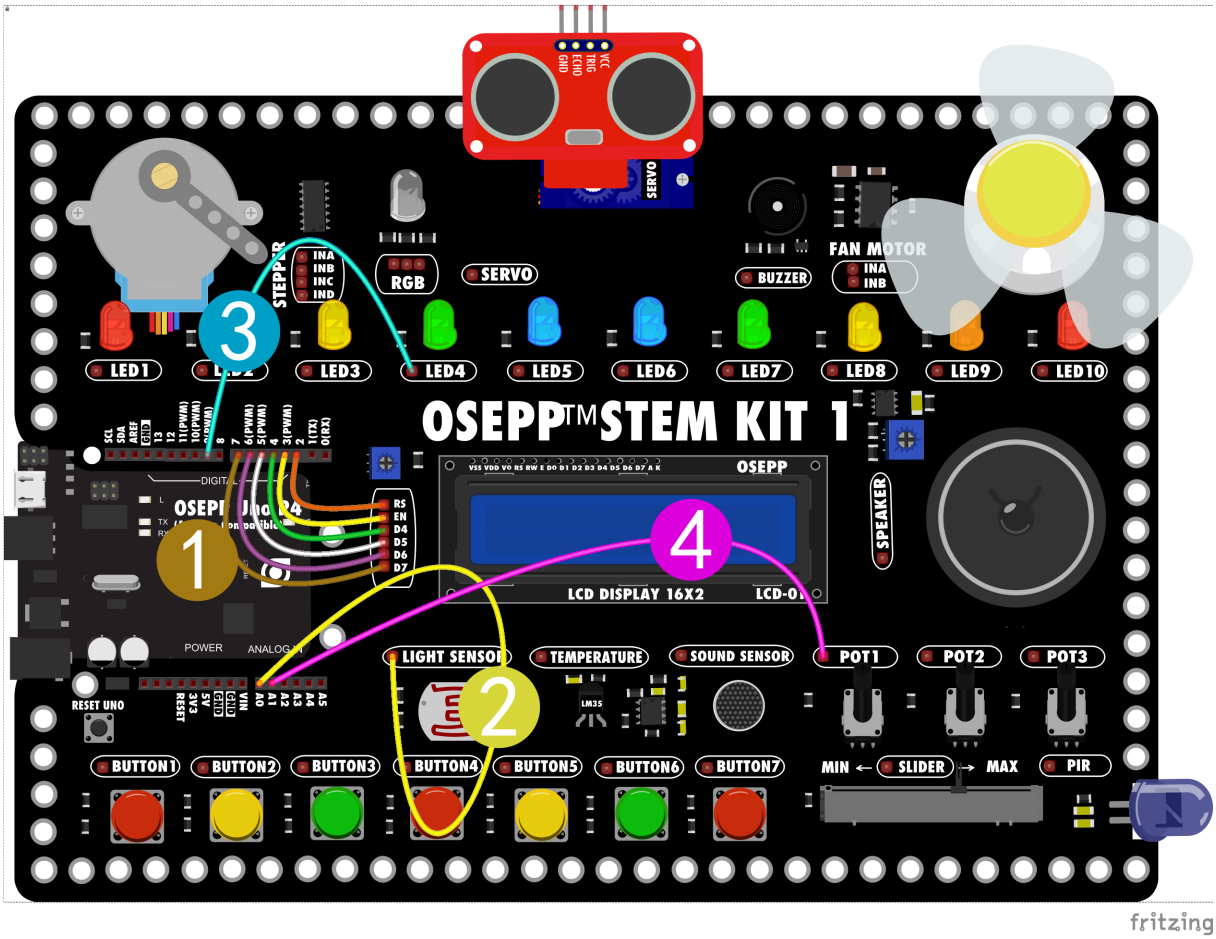
应用 3 制作一个光控灯

当晚上来临，光线变暗，这时我们需要开灯。能否做一个聪明一点的开关呢，无需我们手动开灯，天黑了自动会把灯打开。如果我们有一个光线传感器，我们就可以制作一个自动感应环境光线的开关灯装置。白天光线充足的时候不亮灯，晚上天黑了就会自动点亮。

部署

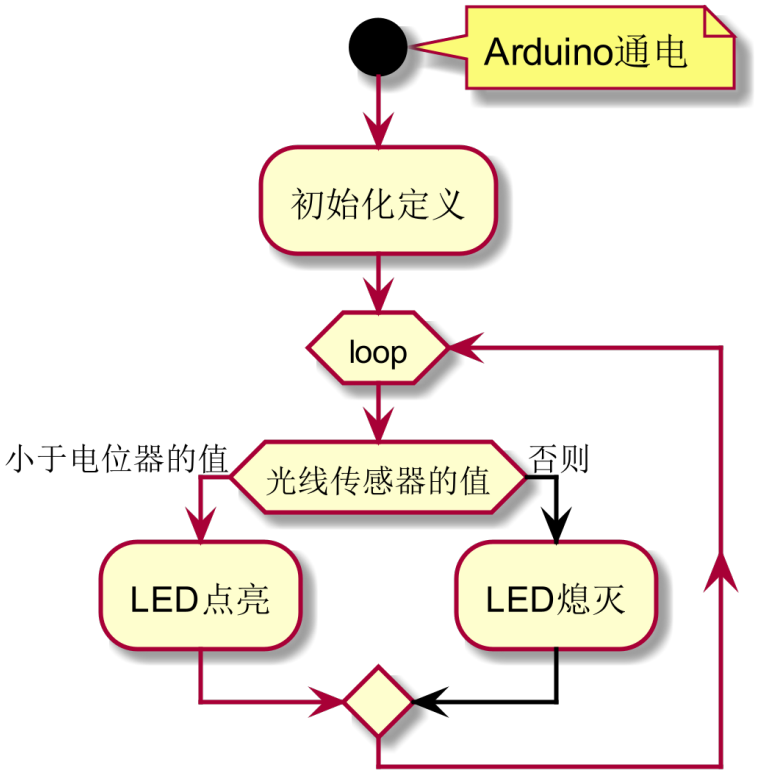
我们希望每个场景触发 LED 点亮的环境光亮度不一样，那就增加一个电位器来调节光线感应的阀值。可以调节电位器的值，让光线传感器的值来与之对比。当光线变化时，如果光线传感器的值小于电位器当前值，那 LED 就触发点亮程序。

1.LCD 的接口 RS-D7 分别连接到 OSEPP UNO 的 2~7 号引脚。
2. 光线传感器 Light Sensor 的端子接到 OSEPP UNO 的 A0 引脚。
3.LED4 的端子连接到 OSEPP UNO 的 9 号引脚。
4. 电位器 POT1 接到 OSEPP UNO 的 A1 引脚。



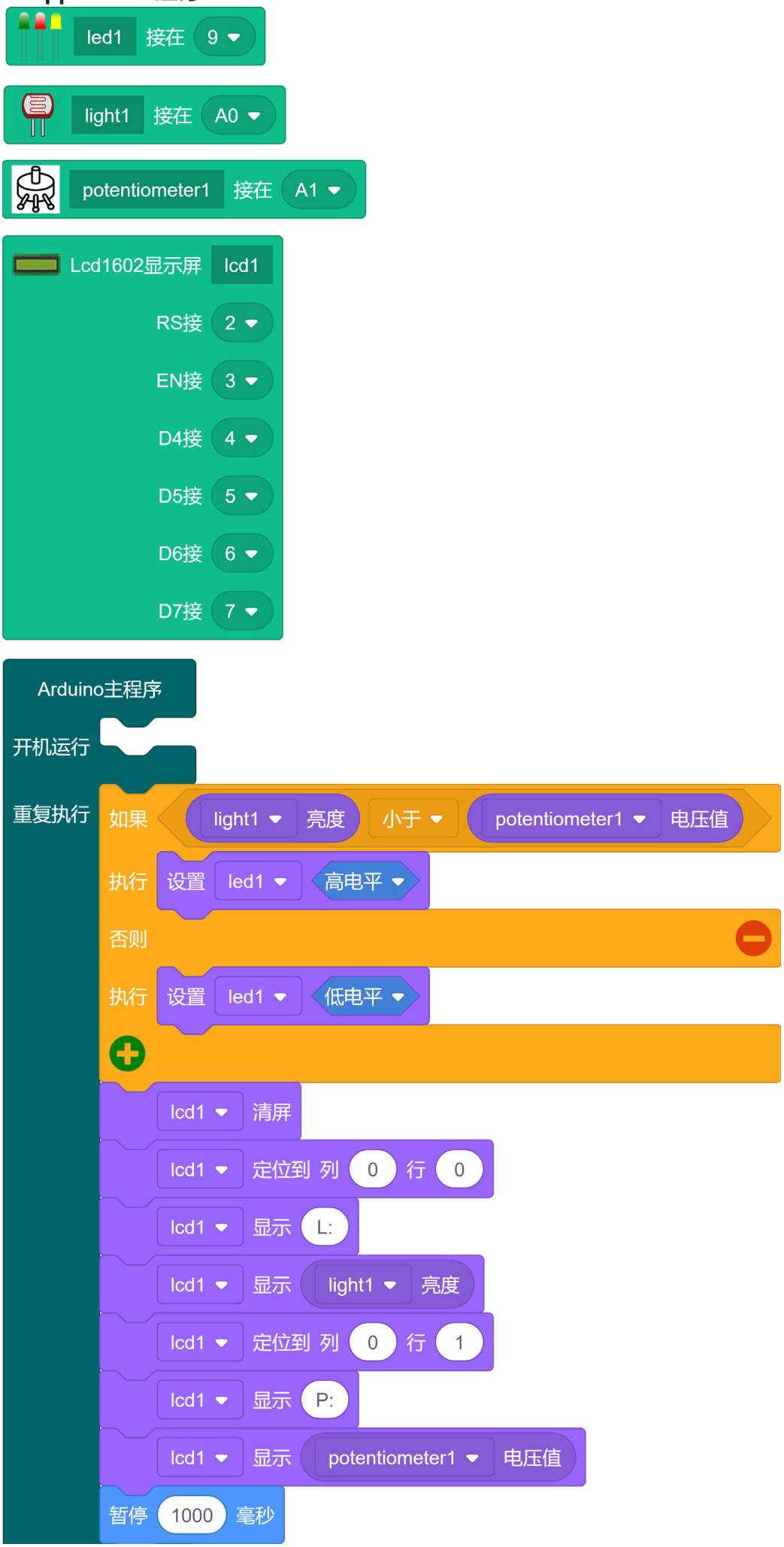
程序搭建

首先了解一下流程



如果光线传感器的值 (A0) 小于电位器的值 (A1)，9 号引脚输出高电平，点亮 LED。否则 9 号引脚输出低电平，LED 熄灭。

oseppBlock 程序



Arduino 程序

```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚

void setup()
{
  //led1
  pinMode(9, OUTPUT); // 定义 LED
  //light1
  pinMode(A0, INPUT); // 定义光线传感器
  //potentiometer1
  pinMode(A1, INPUT); // 定义电位器接到的引脚
  lcd1.begin(16, 2);  //LCD 初始化
}

void loop()
{
  if (analogRead(A0) < analogRead(A1)) // 比较两个值，如果光线传感器的值小于电位器的值
  {
    digitalWrite(9, HIGH); // 点亮 LED
  }
  else // 否则
  {
    digitalWrite(9, LOW); //LED 熄灭
  }
  lcd1.clear();                // 清屏
  lcd1.setCursor(0, 0);        // 显示光标定位
  lcd1.print("L:");            // 显示字符，字符里的：要用半角
  lcd1.print(analogRead(A0)); // 显示光线传感器的值
  lcd1.setCursor(0, 1);        // 显示光标定位
  lcd1.print("P:");            // 显示字符
  lcd1.print(analogRead(A1)); // 显示电位器的值
  delay(1000);                 // 延时 1000 毫秒
}
```

运行结果

LCD 第一行 L:xxx 显示的是光线传感器当前的值，第二行 P:xxx 显示的是电位器的值，也就是我们需要设置触发的阈值。当环境光变暗时，传感器的值就会变小，只要传感器的值小于设定值，LED 就会点亮。否则 LED 处于熄灭状态。调节电位器的值，就能设定 LED 在何种光线强度下触发。

解析

比较运算

比较是程序里面经常用到的一种数学运算。

运算符名称	运算符简写	描述	例子
等于 	==	检查两个操作数的值是否相等，如果相等，则条件为真（true）。	(2 == 3) 不为真
不等于 	!=	检查两个操作数的值是否相等，如果值不相等，则条件为真。	(2 != 3) 为真
小于 	<	检查左操作数的值是否小于右操作数的值，如果是，则条件为真。	(2 < 3) 为真
大于 	>	检查左操作数的值是否大于右操作数的值，如果是，则条件为真。	(2 > 3) 不为真
小于或等于 	<=	检查左操作数的值是否小于或等于右操作数的值，如果是，则条件为真。	(2 <= 3) 为真
大于或等于 	>=	检查左操作数的值是否大于或等于右操作数的值，如果是，则条件为真。	(2 >= 3) 不为真

条件语句

如果条件为真就执行动作 A，否则执行动作 B。

`if`(条件) `and` `==`, `!=`, `<`, `>` (比较运算符)
`if`，用于和比较运算符联合使用，测试某一条件是否成立。


```
if (i > 50) {
    // 执行动作
} else {
    // 执行 ...
}
```

该程序测试 `i` 是否大于 50。如果是， 程序执行特定的动作。换句话说，如果圆括号中的语句为真，大括号中的语句就会运行。否则，程序跳过该代码。
(`else` 意思是当上面条件不成立时，执行 .... 代码)

RGB 三原色 LED 彩灯

三原色光模式 (**RGB color model**)，又称 **RGB** 颜色模型或红绿蓝颜色模型，是一种加色模型，将红 (**Red**)、绿 (**Green**)、蓝 (**Blue**) 三原色的色光以不同的比例相加，以合成产生各种色彩光。

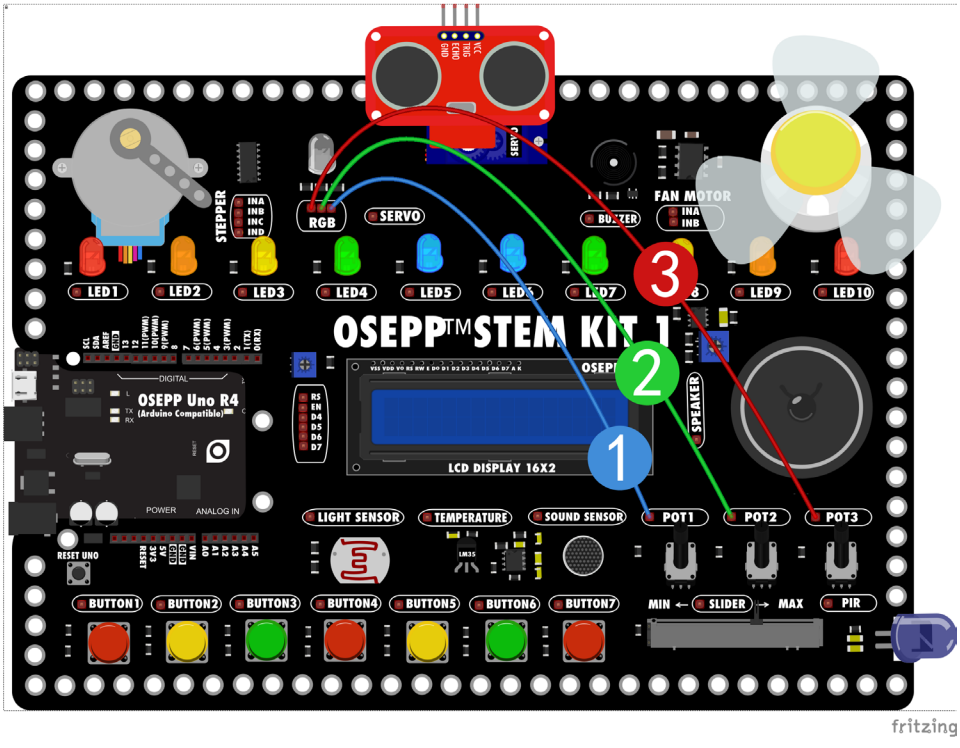
RGB 三原色 LED 彩灯，由红色、绿色和蓝色三个独立的灯珠构成，常见的有四个引脚，一个公共端和三个颜色控制端。三个颜色任意组合可以产生其他颜色，如红色和绿色同时亮，蓝色不亮则是黄色；绿色和蓝色同时亮，红色不亮则是青色；红色和蓝色同时亮，绿色不亮则是品红色；三色都亮则产生白色。

应用 1 电位器连接 RGB 发光二极管

RGB 发光二极管上面有 4 个引脚，学习板上面的是共阴极 **RGB** 发光二极管。类似于有红绿蓝三个 LED，把它们的阴极接在一起，只用一条导线引出来。那么只要给阳极的三个端子输入高电平，就会点亮 **RGB** 发光二极管。学习板上已经把阴极接到电源地 GND，我们只需要接阳极的三个端子就可以。

部署

1. 电位器 POT1 端子连接到 RGB 发光二极管的 B 端子。
2. 电位器 POT2 端子连接到 RGB 发光二极管的 G 端子。
3. 电位器 POT3 端子连接到 RGB 发光二极管的 R 端子。



运行结果

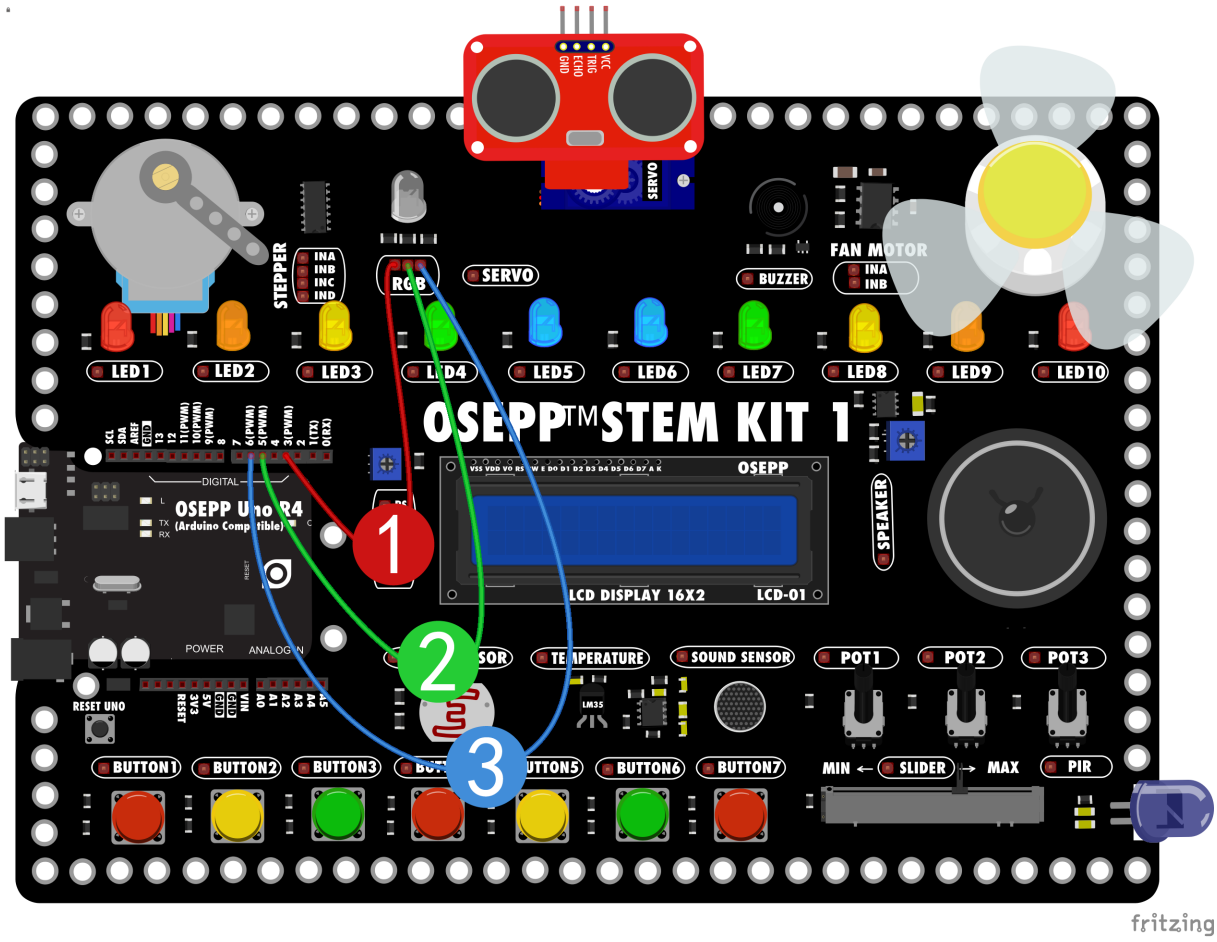
此时通过调节电位器就可以看到颜色的变化。通过控制各个 LED 的亮度，你可以混合出几乎任何你想要的颜色。

应用 2 Arduino 驱动 RGB 发光二极管

通过 R、G、B 三个引脚的 PWM 电压输入可以调节三种基色（红 / 蓝 / 绿）的强度从而实现全彩的混色效果。用 **Arduino** 程序控制可实现酷炫的灯光效果。

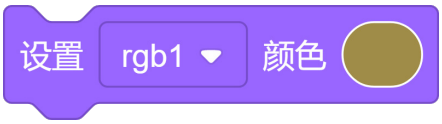
部署

1. RGB 发光二极管的 R 端子连接到 **OSEPP UNO** 的 3 号引脚。
2. RGB 发光二极管的 G 端子连接到 **OSEPP UNO** 的 5 号引脚
3. RGB 发光二极管的 B 端子连接到 **OSEPP UNO** 的 6 号引脚。

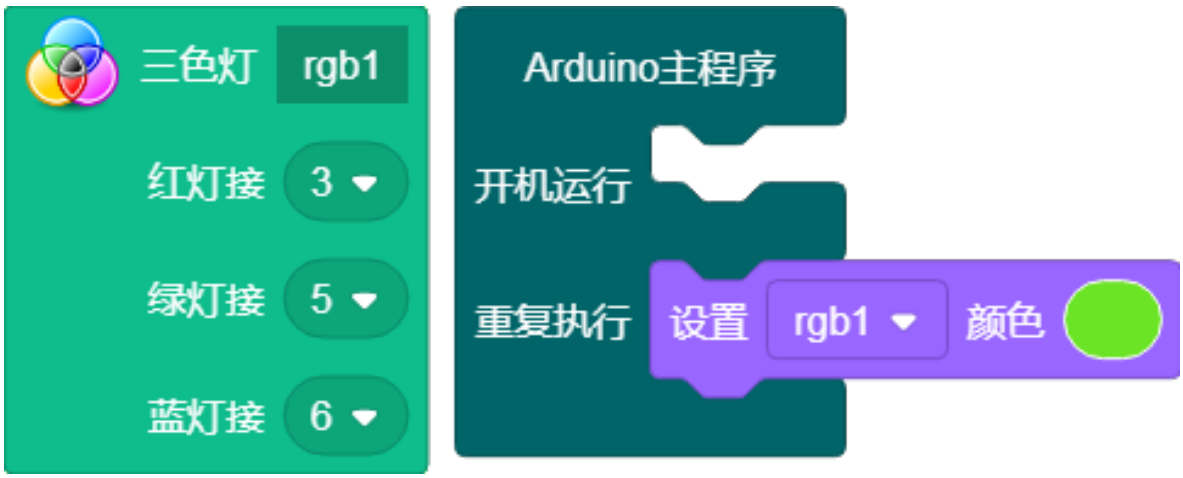


程序搭建

oseppBlock 积木知识

	RGB 发光二极管的定义积木。 定义名称和连接引脚。
	设置 RGB 发光二极管的颜色积木。通过滑块调节，可以设置颜色，饱和度和亮度。
	设置 RGB 发光二极管颜色积木。通过输入三种颜色的的数值来调节改变颜色，数值范围是 0-255。此积木要放入上面积木颜色框里面。

oseppBlock 程序



拖动滑块 RGB 发光二极管就会显示和积木里面一致的颜色。

下面有一个颜色显示数据表，可以在程序里面输入数字来控制颜色。利用 **Arduino** 来控制输出需要的颜色。

名称	颜色	色光		
		R	G	B
红色		255	0	0
黄色		255	255	0
绿色		0	255	0
青色		0	255	255
蓝色		0	0	255
品红色		255	0	255
栗色		128	0	0
橄榄色		128	128	0
深绿色		0	128	0
蓝绿色		0	128	128
深蓝色		0	0	128
紫色		128	0	128
白色		255	255	255
银色		192	192	192
灰色		128	128	128
黑色		0	0	0

把 RGB 颜色积木换成自定义数值的积木

oseppBlock 程序

Arduino 程序

```
void setup()
{
    //rgb1
    pinMode(3, OUTPUT); // 定义 3 号引脚为输出模式
    pinMode(5, OUTPUT); // 定义 5 号引脚为输出模式
    pinMode(6, OUTPUT); // 定义 6 号引脚为输出模式
}

void loop()
{
    analogWrite(3, 255); //3 号引脚输出红色全亮
    analogWrite(5, 0);   //5 号引脚输出绿色 0
    analogWrite(6, 0);   //6 号引脚输出蓝色 0
}
```

表格里面给了 RGB 三个通道的数值，我们只要照着填上去，就可以能看 RGB 发光二极管显示出表格里的颜色。

只要给 RGB 三个颜色输入不同的数值，RGB 发光二极管就能显示不同的颜色。那我们把这个数值改成随机数，RGB 发光二极管就能随机显示颜色。

oseppBlock 程序

Arduino 程序

```
int R = 0; // 定义整型变量 R
int G = 0; // 定义整型变量 G
int B = 0; // 定义整型变量 B

void setup()
{
    //rgb1
    pinMode(3, OUTPUT); // 定义 3 号引脚为输出模式
    pinMode(5, OUTPUT); // 定义 5 号引脚为输出模式
    pinMode(6, OUTPUT); // 定义 6 号引脚为输出模式
}

void loop()
{
    R = random(0, 256); //R 等于 0-256 之间的随机数
    G = random(0, 256); //G 等于 0-256 之间的随机数
    B = random(0, 256); //B 等于 0-256 之间的随机数
    analogWrite(3, R);  //3 号引脚输出 R 的随机数
    analogWrite(5, G);  //3 号引脚输出 G 的随机数
    analogWrite(6, B);  //3 号引脚输出 B 的随机数
    delay(500);         // 延时 500 毫秒
}
```

运行结果

把程序传上 **osepp UNO** 后，RGB 发光二极管就会随机变换颜色。每隔 0.5 秒换一次颜色，你可以更改这个延时数值，数值越小变化越快。

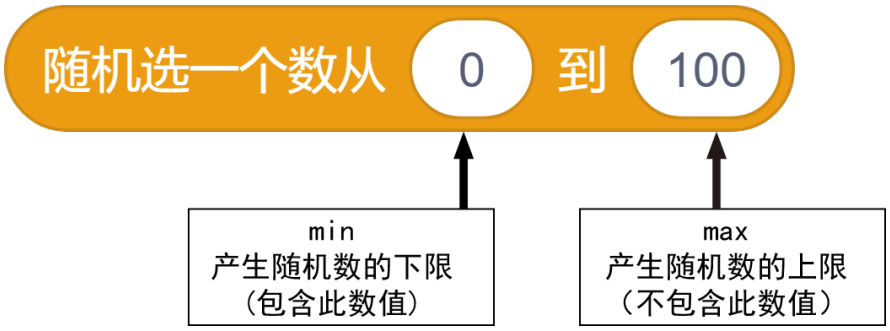
解析

整型变量： `int`

整型变量 (`int`) 是变量里的一种，它可以用来存储 `-32768`~`+32767` 之间的数字。在 **Arduino** 代码中，整型 (`int`) 是最常用的变量类型。变量就是告诉程序，这个值类型的和范围。

		
定义变量的类型名称和初始值积木。有整型变量，长整型变量，无符号整型变量，无符号长整型变量，字符变量，字节变量和浮点变量。	变量名称为 `i` 的值的积木。	改变变量值的积木，可以进行数学运算。

随机函数： `random()`



随机函数 `random(min, max)` ，就是生成一个随机的数值。
参数
`min`: 产生随机数的下限 (包含此数值)
`max`: 产生随机数的上限（不包含此数值）
返回值
在最小值 `min` 和最大值 `max-1` 之间的随机数值

扬声器

扬声器又称喇叭、音响。是一种转换电子信号成为声音的换能器、电子组件，可以由一个或多个组成音响组。电视机，录音机里面的声音都是扬声器发出来的。

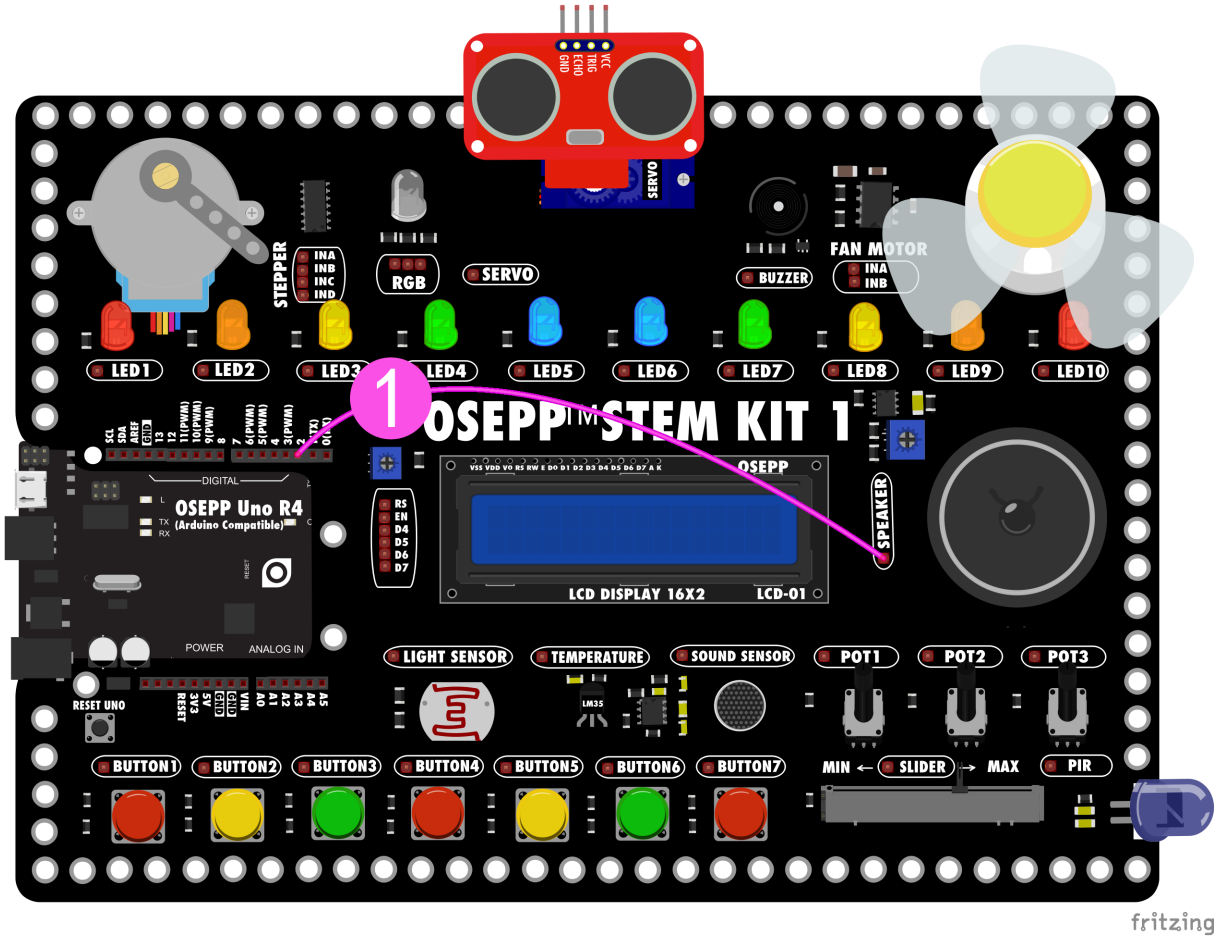
扬声器是由电磁铁、线圈、喇叭振膜组成。扬声器把电流频率转化为声音。物理学原理，当电流通过线圈产生电磁场，磁场的方向为右手法则。假设，扬声器播放 C 调，其频率为 256 赫兹，即每秒振动 256 次，扬声器输出 256 赫兹的交流电，每秒 256 次电流改变，发出 C 调频率。当电线圈与扬声器薄膜一起振动，推动周围的空气振动，扬声器由此产生声音。
人耳可以听到的声波的频率一般在 20 赫兹至 20000 赫兹之间，所以一般的扬声器都会把工作频率设定在这个范围内。

应用 1 Arduino 让扬声器发出声音

如何让扬声器发出一个声音。扬声器的工作频率在 **20-20KHz** 之间，那只要让 **Arduino** 输出这个频段内的频率就能让扬声器发出声音了。

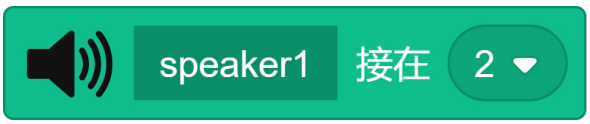
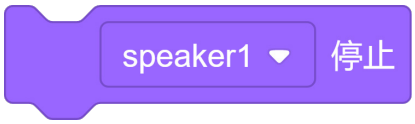
部署

把扬声器 SpeaKer 的端子接到 **OSEPP UNO** 的 2 号引脚。



程序搭建

oseppBlock 积木知识

	扬声器定义积木。 定义名称和端口号。
	扬声器驱动积木。 扬声器播放 c3 音调的频率， 对应代码 `tone(2, 131);`
	扬声器驱动积木。 扬声器停止播放声音， 对应代码 `noTone(2);`

oseppBlock 程序

Arduino 程序

```
void setup()
{
    //speaker1
    pinMode(2, OUTPUT); // 定义 2 号引脚为输出模式
}

void loop()
{
    tone(2, 131); //2 号引脚输出 131 频率的波形
    delay(500);   // 延时 500 毫秒
    noTone(2);    // 停止声音
    delay(3000);  // 延时 3000 毫秒
}
```

运行结果

上传代码后，扬声器就能发出 c3 音调声音 0.5 秒，然后暂停 3 秒。

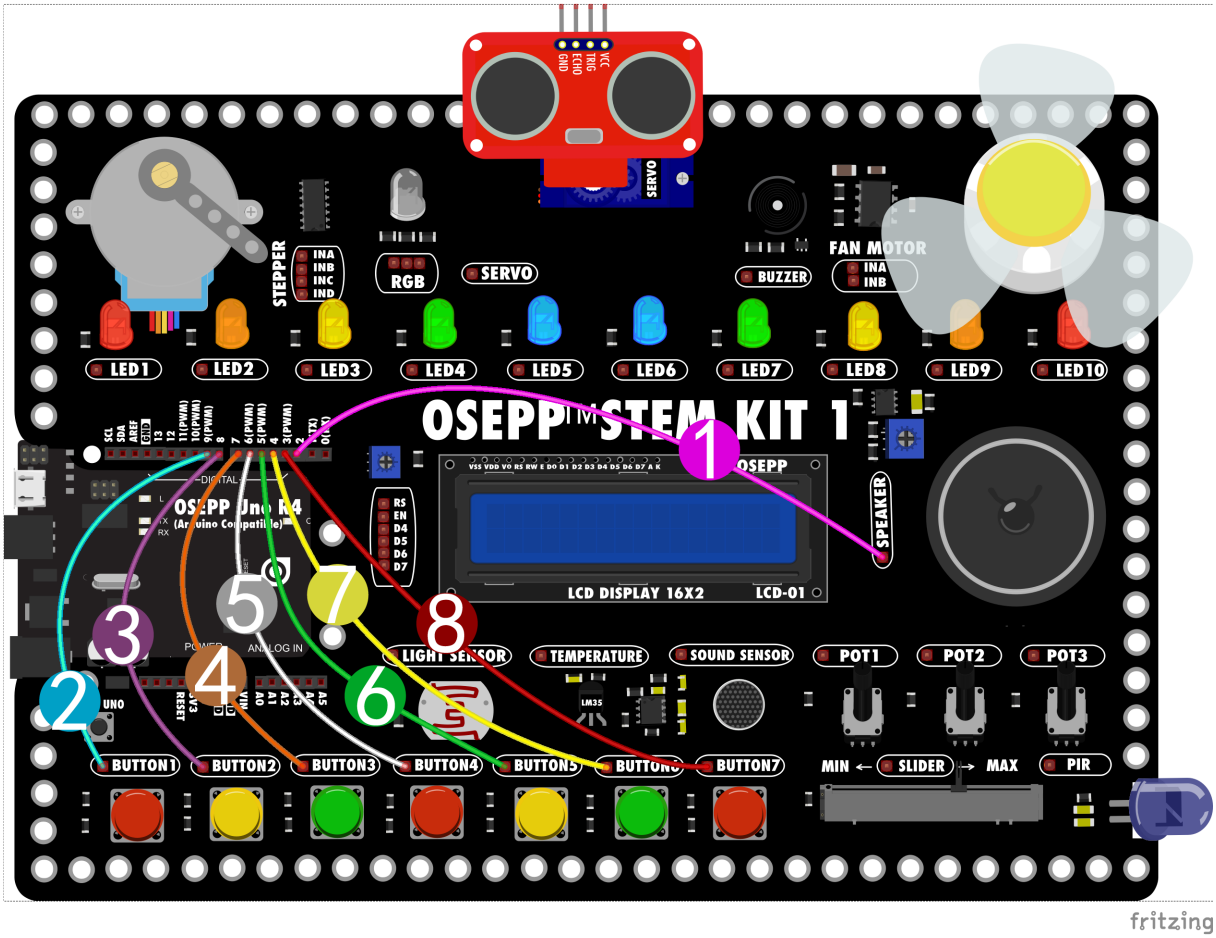
应用 2 Arduino 电子琴

上面的实验让扬声器只发出一个声音显得太无趣，接下来我们要让扬声器发出不同的声调。并且使用一个开关控制一个声调，就像电子琴一样。

部署

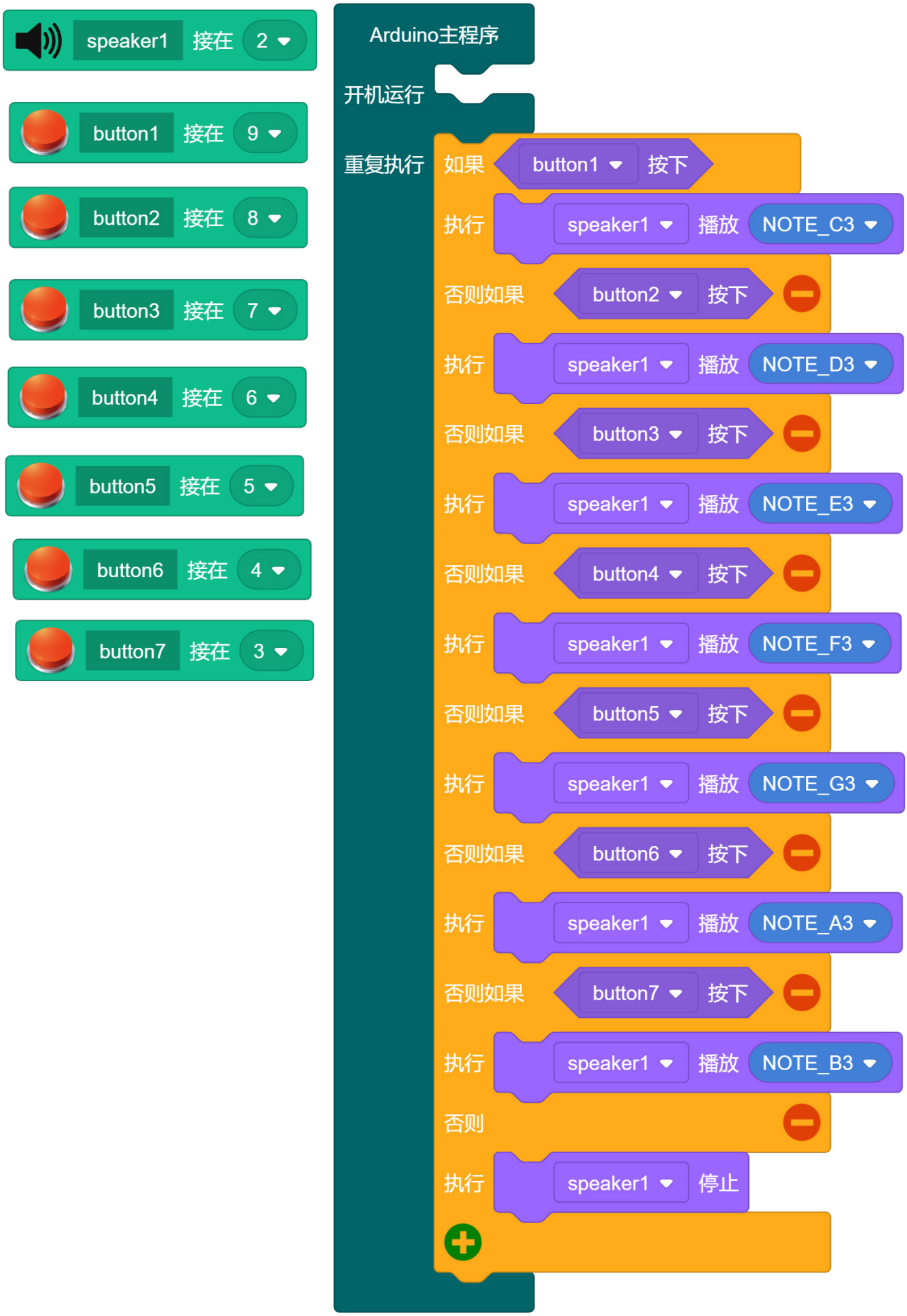
按照下表连接好导线

序号	元件端子	osepp UNO 引脚号
1	Speaker	2
2	Button1	9
3	Button2	8
4	Button3	7
5	Button4	6
6	Button5	5
7	Button6	4
8	Button7	3



程序搭建

oseppBlock 程序



Arduino 程序

```
void setup()
{
  //speaker1
  pinMode(2, OUTPUT); // 定义 2 号引脚为输出模式
  //button1
  pinMode(9, INPUT); // 定义 9 号引脚为输入模式
  //button2
  pinMode(8, INPUT); // 定义 8 号引脚为输入模式
  //button3
  pinMode(7, INPUT); // 定义 7 号引脚为输入模式
  //button4
  pinMode(6, INPUT); // 定义 6 号引脚为输入模式
  //button5
  pinMode(5, INPUT); // 定义 5 号引脚为输入模式
  //button6
  pinMode(4, INPUT); // 定义 4 号引脚为输入模式
  //button7
  pinMode(3, INPUT); // 定义 3 号引脚为输入模式
}

void loop()
{
  if (digitalRead(9) == LOW) // 如果 9 号引脚为低电平时
  {
    tone(2, 131); //2 号引脚输出 131 频率的声音
  }
  else if (digitalRead(8) == LOW) // 如果 8 号引脚为低电平时
  {
    tone(2, 147); //2 号引脚输出 147 频率的声音
  }
  else if (digitalRead(7) == LOW) // 如果 7 号引脚为低电平时
  {
    tone(2, 165); //2 号引脚输出 165 频率的声音
  }
  else if (digitalRead(6) == LOW) // 如果 6 号引脚为低电平时
  {
    tone(2, 175); //2 号引脚输出 175 频率的声音
  }
  else if (digitalRead(5) == LOW) // 如果 5 号引脚为低电平时
  {
    tone(2, 196); //2 号引脚输出 196 频率的声音
  }
  else if (digitalRead(4) == LOW) // 如果 4 号引脚为低电平时
  {
    tone(2, 220); //2 号引脚输出 220 频率的声音
  }
```

```
    else if (digitalRead(3) == LOW) // 如果 3 号引脚为低电平时
    {
      tone(2, 247); //2 号引脚输出 247 频率的声音
    }
    else // 否则
    {
      noTone(2); // 停止声音
    }
  }
```

运行结果

学习板上只有 7 个开关，我们就设定前面 7 个音调。从开关 1 至 7 就是哆唻咪哔嗦啦西。找一首简单的曲子，试试能不能弹出来 1-2-3-1，1-2-3-1，3-4-5，3-4-5...

一首乐曲有若干音符组成，一个音符对应一个频率。tone() 函数可以产生固定频率的 PWM 信号来驱动扬声器发声。发声时间长度和声调都可以通过参数控制。

定义发声时间长度有两种方法，第一种是通过 tone() 函数的参数来定义发声时长，另一种是使用 no Tone() 函数来停止发声。

如果您在使用 tone() 函数时没有定义发声时间长度，那么除非您通过 noTone() 函数来停止声音，否则 Arduino 将会一直通过 tone() 函数产生声音信号。

解析

Arduino 一次只能产生一个声音。假如 Arduino 的某一个引脚正在通过 tone() 函数产生发声信号，那么此时让 Arduino 使用另外一个引脚通过 tone() 函数发声是不行的。

函数 tone()：	

代码：
 	
`tone(pin, frequency)`
`tone(pin, frequency, duration)`
	参数：
	
	pin: 发声引脚（该引脚需要连接扬声器）
	frequency: 发声频率（单位：赫兹 Hz） – 无符号整数型
	duration: 发声时长（单位：毫秒，此参数为可选参数）
	– 无符号长整型
	该函数无返回值

蜂鸣器

蜂鸣器（buzzer）是一种电子发声元器件。是一体化结构的电子讯响器，广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机、定时器等电子产品中作发声器件。

蜂鸣器分为有源蜂鸣器和无源蜂鸣器两种：

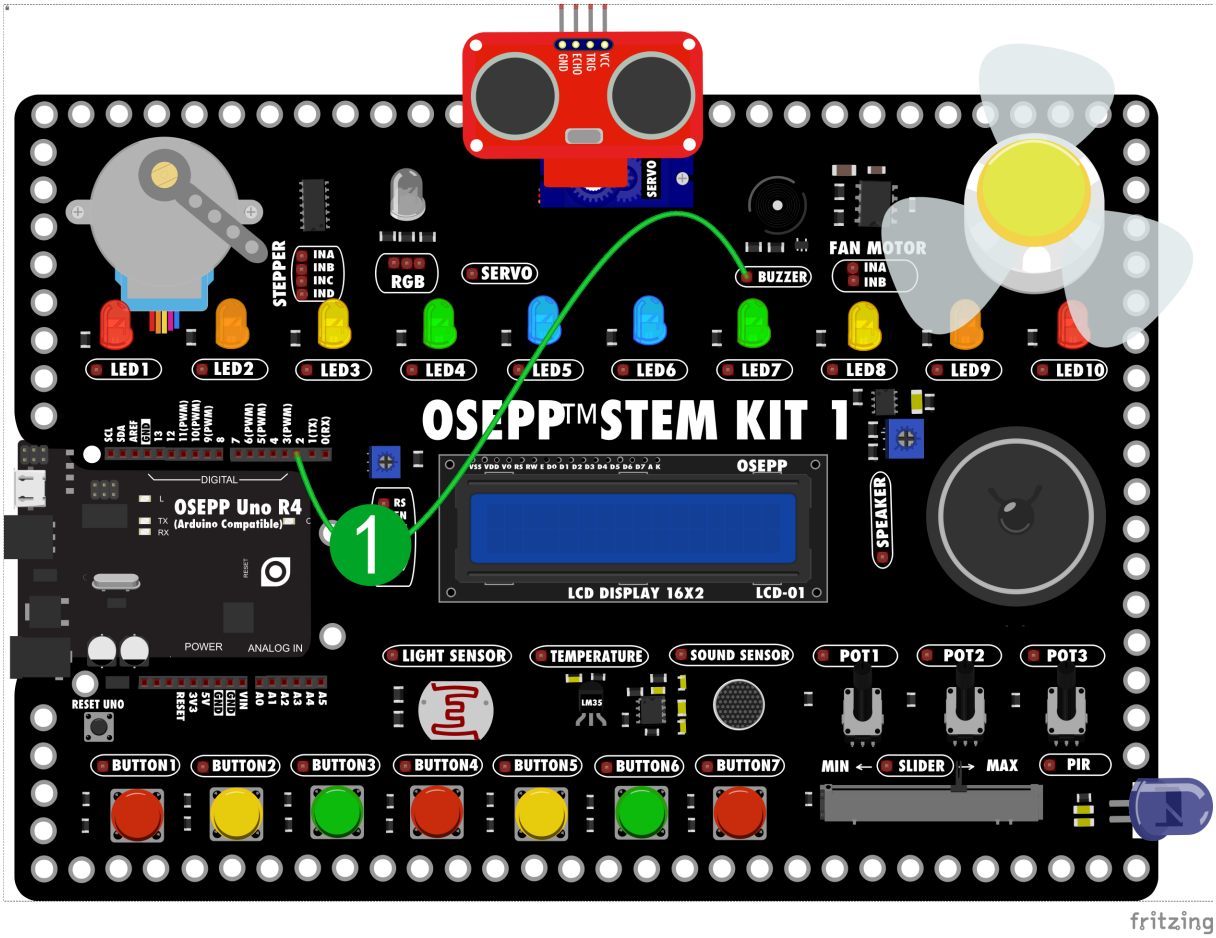
有源蜂鸣器内部带震荡源，所以只要一通电就会发声。
无源蜂鸣器内部不带震荡源，无法用直流信号驱动发出声音，必须用 2K-5K 的波形脉冲信号去驱动它。

让蜂鸣器发出声音

STEM KIT 学习板上使用的是有源蜂鸣器。有源蜂鸣器一通电就会发声，那我们把它接入 Arduino，只要程序里设定高电平输出就会发声了。

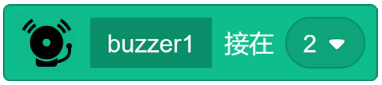
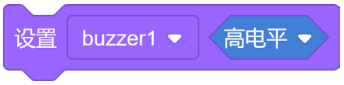
部署

把扬声器 SpeaKer 的端子接到 OSEPP UNO 的 2 号引脚。

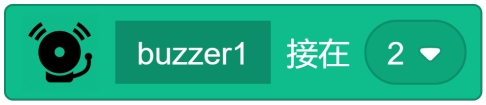


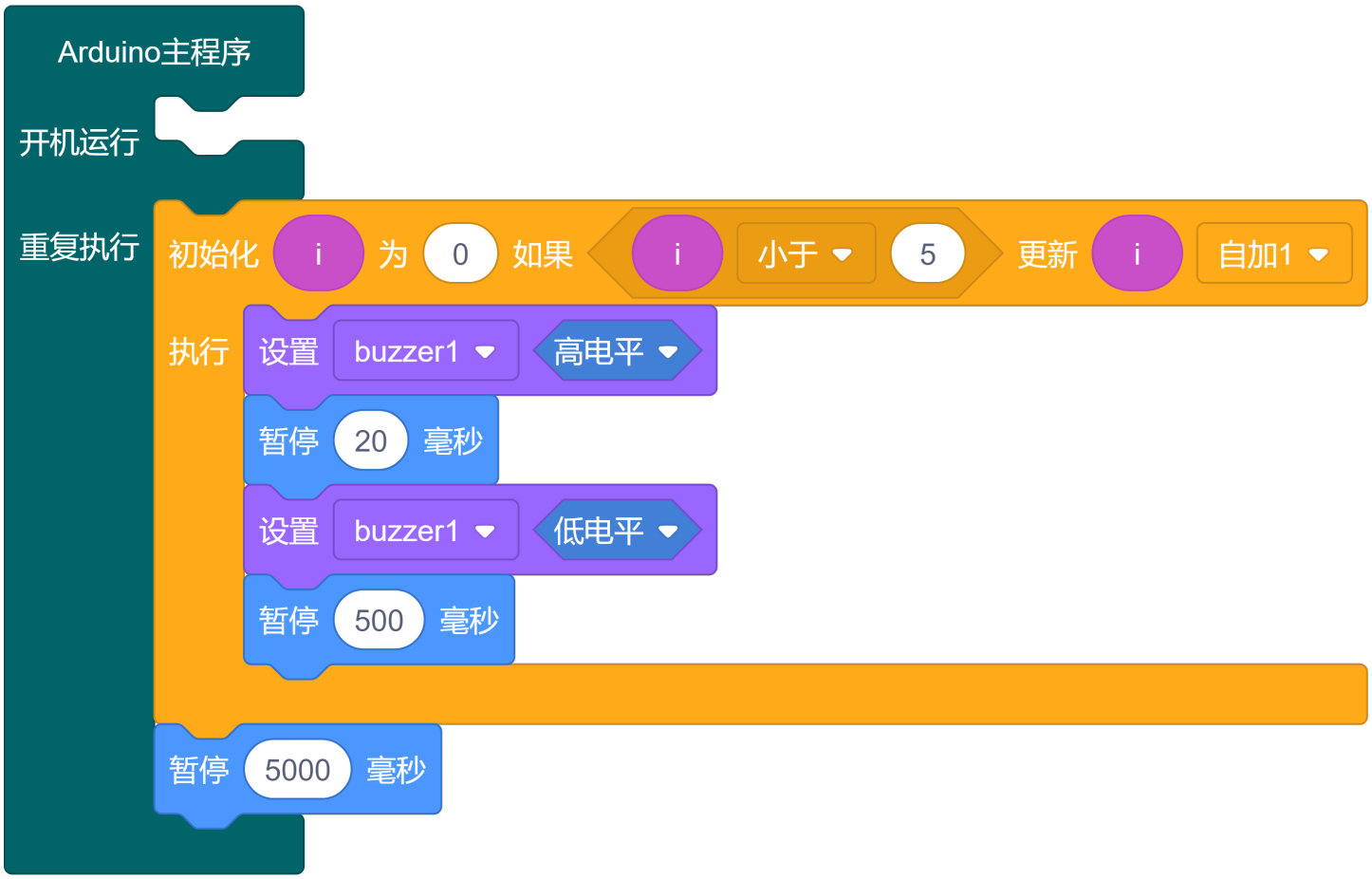
程序搭建

oseppBlock 积木知识

	蜂鸣器模块积木。蜂鸣器是输出器件，数字和模拟引脚都可以选择。
	驱动蜂鸣器积木。可以选择高电平或者低电平，高电平时蜂鸣器发出声音，低电平静音。
	循环计数器积木，从给定条件中得到循环次数。

oseppBlock 程序





Arduino 程序

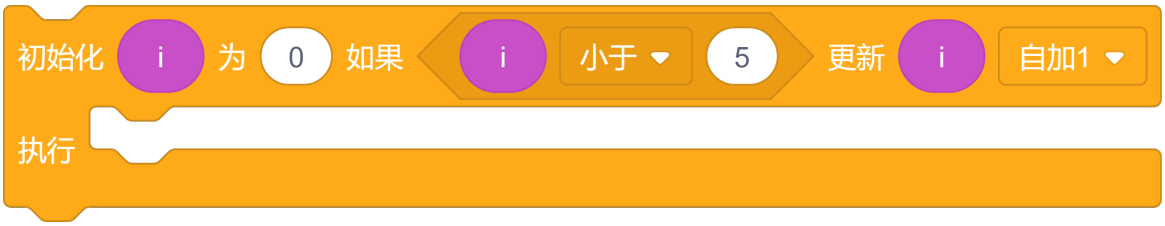
```
int i = 0; // 定义一个整型变量

void setup()
{
  //buzzer1
  pinMode(2, OUTPUT); // 定义蜂鸣器引脚
}

void loop()
{
  for (i = 0; i < 5; i++) // 循环 5 次
  {
    digitalWrite(2, HIGH); // 蜂鸣器发声
    delay(20);             // 延时 20 毫秒
    digitalWrite(2, LOW);  // 蜂鸣器静音
    delay(500);            // 延时 500 毫秒
  }
  delay(5000); // 延时 5 秒
}
```

运行结果

蜂鸣器每 0.5 秒响一次，连续 5 次，之后暂停 5 秒再次进入循环。

解析

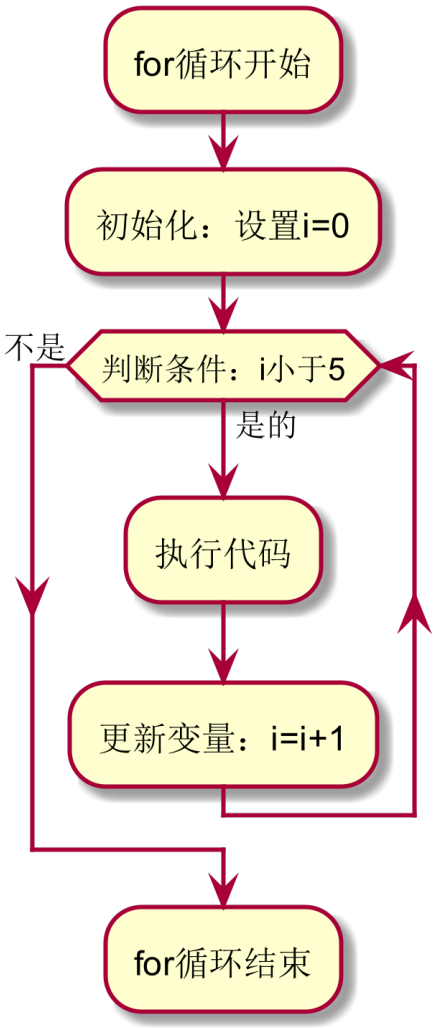
使用计数器重复执行语句



for 循环由三部分组成：初始化，条件测试和迭代（即在每一个循环过程的最后执行的语句），每一部分都用分号隔开。

代码：`for (i = 0; i < 5; i++)`

代码解答：`int i = 0;` 初始化变量 `i` 为 `0`，`i < 5;` 测试变量，看看它是否小于 `5`，`i++` 使 `i` 递增。



PIR(人体热释电红外传感器)

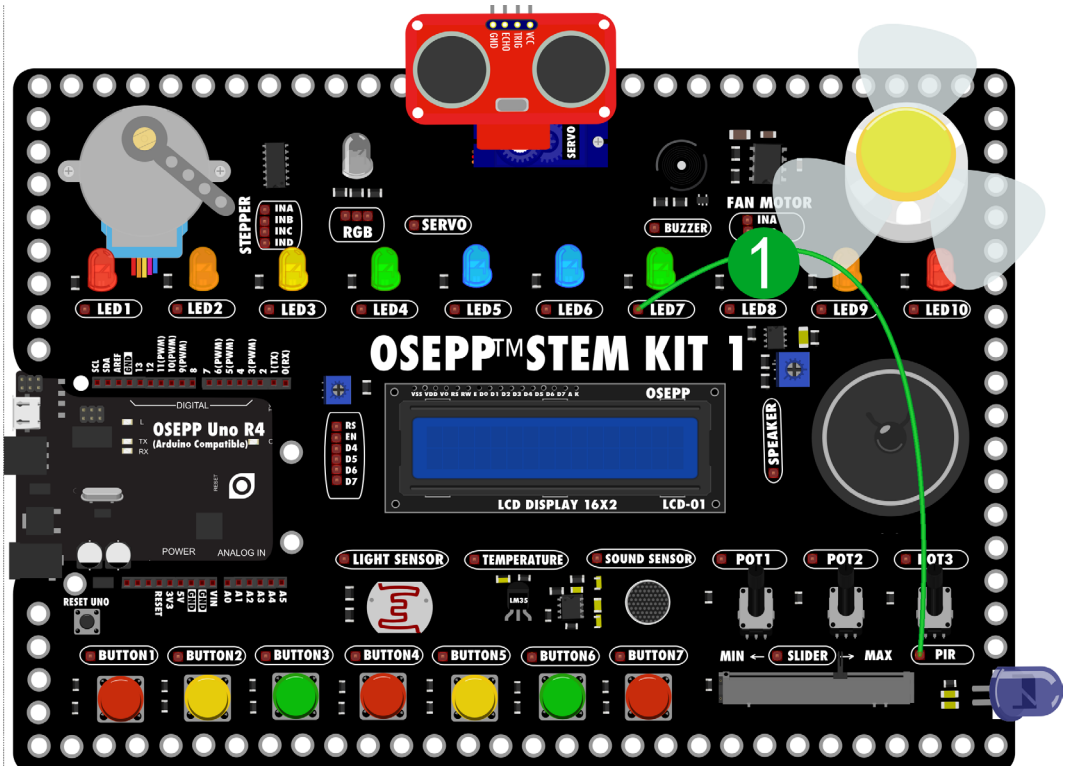
发热的物体都会产生红外线，不同的温度其辐射的红外线波长不同。因为人体的体温恒定在 37 度，所以会固定辐射 10um 的远红外线，PIR 就是鉴于这样的特征来进行人体侦测的。
原理是这样的，当人体发出的 10um 红外线通过 PIR 前段的菲尼尔透镜聚焦到 PIR 传感器的表面，PIR 本身有一个滤光片，只让 8~14um 的远红外线进入，PIR 内部的传感器在接收到 10nm 的红外线时，会在热释电晶体上产生电荷移动，从而再驱动内部的 MOS 产生正弦波形式的信号，此信号经过 PIR 后端的运放放大后，可以通过比较器将超过或低于某个电压值的信号取出，这个信号就是 PIR 输出的脉冲信号了，接 MCU 或其他线路就可以用来实现各种形式的控制或报警。
STEM KIT 上有一个 PIR 传感器，

它在检测到人体时，会在端子上产生高电平。1.5 秒后恢复为低电平，此后的 1 秒钟为静默时间，期间即使有人在活动也不会再次触发。

通过将它连接到 LED 上，我们可以很方便地观察到它具体是如何工作的

应用 1 PIR 连接 LED

把 PIR 的端子接到 LED7 的端子上面，接通电源。PIR 进入自检状态，大约持续 3-5 秒钟。自检完成后进入检测状态。当 PIR 检测到人体发出的红外辐射，会在端子上面输出高电平并持续 1 秒钟，随后进入 1 秒钟的静默期。静默期间 PIR 不工作。端子输出低电平，随后再次进入检测状态。



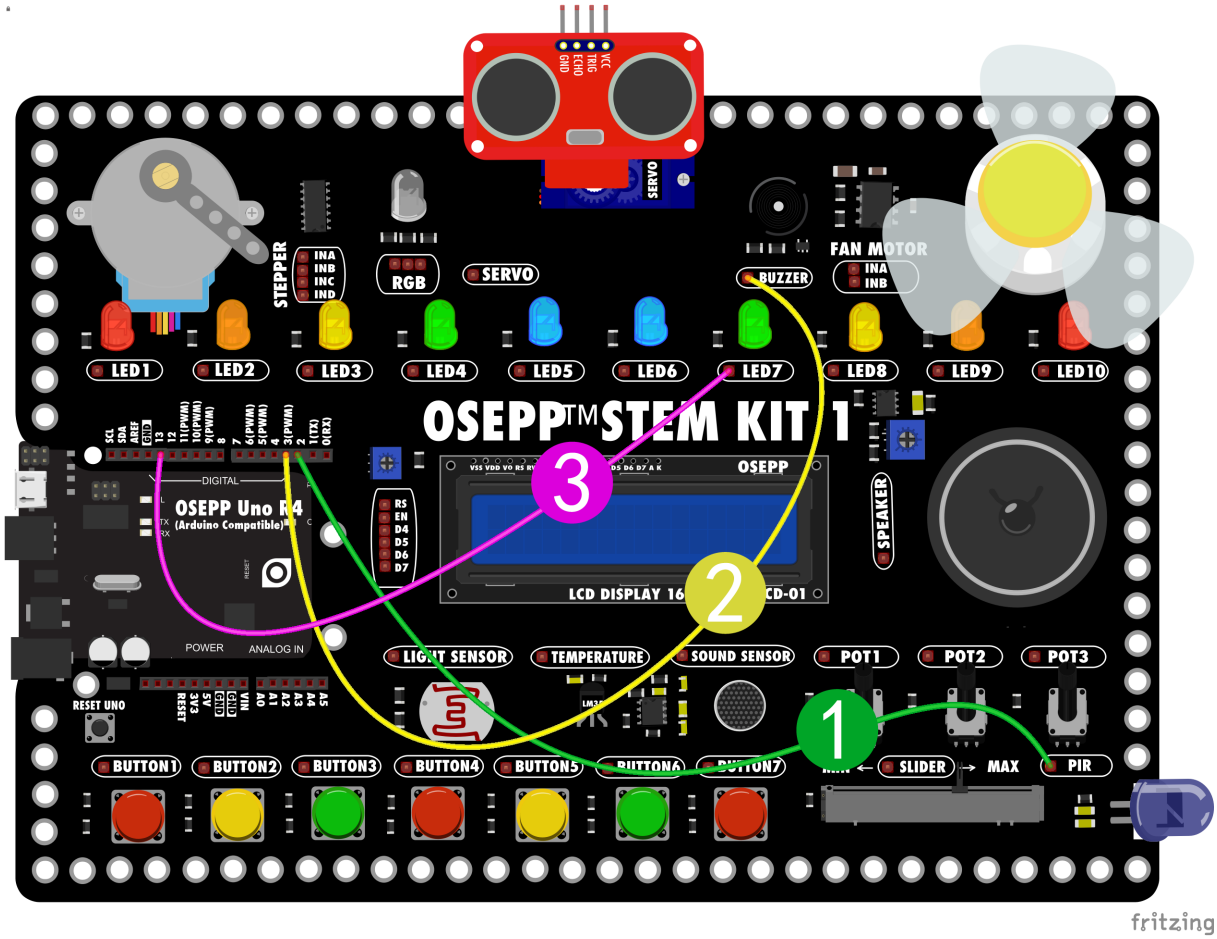
应用 2 欢迎门铃

我们走进一些门店的时候，是不是会听到欢迎光临的声音呢？这个就是利用 PIR 传感器感应到我们人体，然后通过程序处理，发出欢迎光临的声音。我们也可以实验用 PIR 做一个门铃。

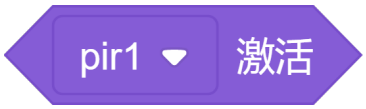
部署

1. PIR 接到 OSEPP UNO 的 2 号引脚。
2. 蜂鸣器 Buzzer 接到 OSEPP UNO 的 3 号引脚。
3. LED7 接到 OSEPP UNO 的 13 号引脚。

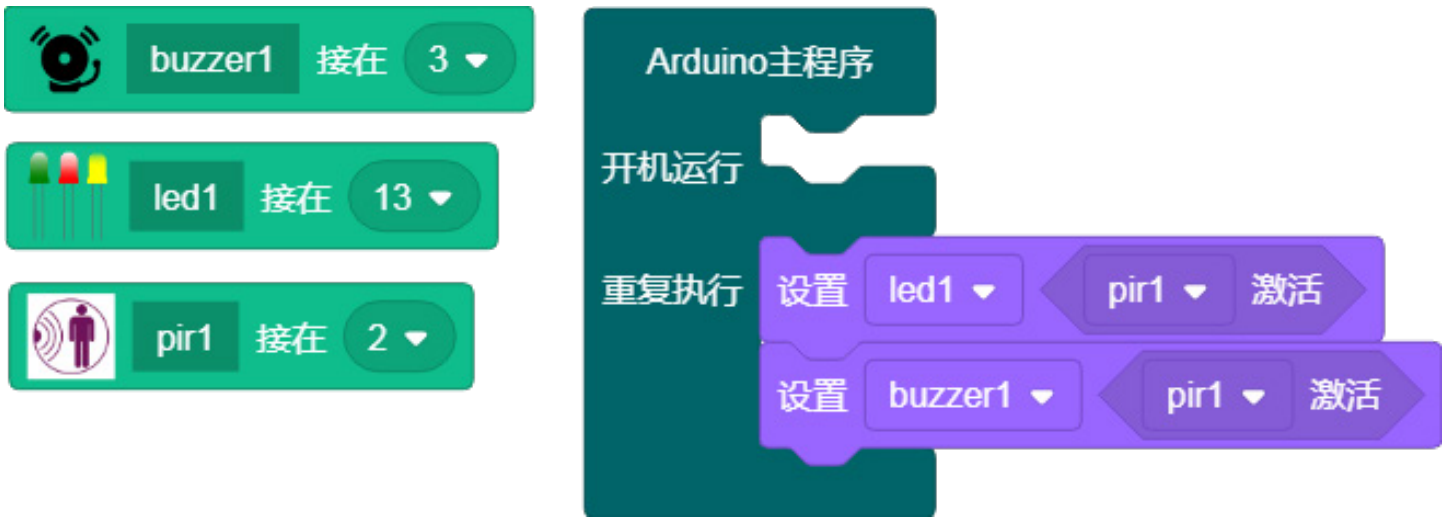
程序搭建



oseppBlock 积木知识

	PIR(人体热释电红外传感器) 模块积木。只有高电平和低电平两种状态，数字和模拟端口都可以选择连接。
	PIR(人体热释电红外传感器) 模块返回值。当 PIR 被出发时返回高电平，否则低电平。

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //led1
    pinMode(13, OUTPUT); // 定义 13 号引脚为输出模式
    //pir1
    pinMode(2, INPUT); // 定义 2 号引脚为输入模式
    //buzzer1
    pinMode(3, OUTPUT); // 定义 3 号引脚为输出模式
}

void loop()
{
    digitalWrite(13, digitalRead(2)); //2 号引脚高电平时 13 号输出高电平
    digitalWrite(3, digitalRead(2));  //2 号引脚高电平时 3 号输出高电平
}
```

运行结果

当 **PIR** 感应到人体的红外热源的时候，就会通过内部电路触发高电平，从而点亮 LED 并且让蜂鸣器发声。

学习板上面的 **PIR** 模块里面内置了延时电路，所以这里没有加 `delay()` 代码也会有延时效果。

麦克风

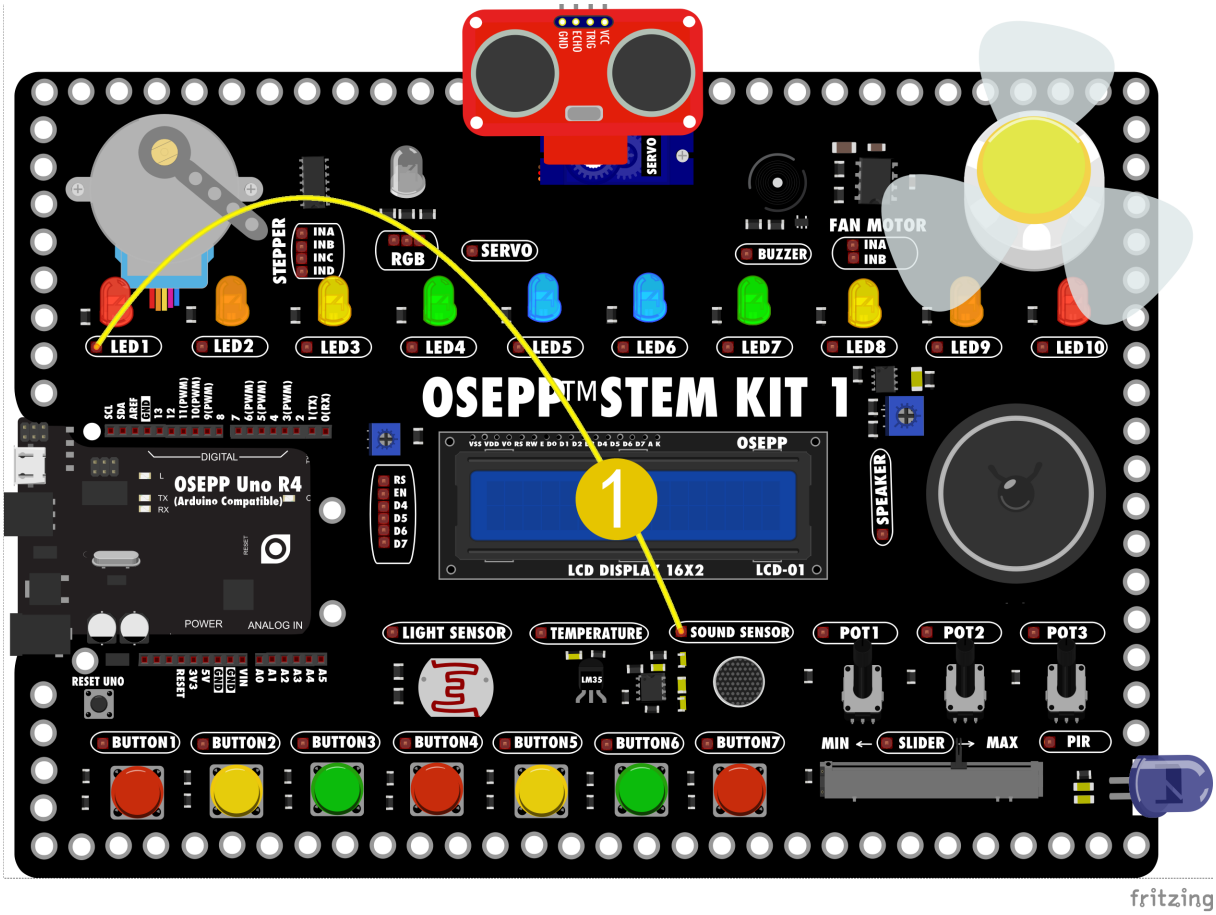
麦克风又是一种声音采集器，是将周围采集到的声音转换为一种电信号的元器件。

应用 1 LED 直连麦克风

麦克风把外界采集到的声音信号转换成电信号。我们直接把麦克风连接到 LED，有声音时麦克风就能触发 LED 发光。

部署

把 LED1 的端子连上麦克风 Sound Sensor 的端子。



运行结果

此时对着麦克风说话就会看到 LED 闪烁。

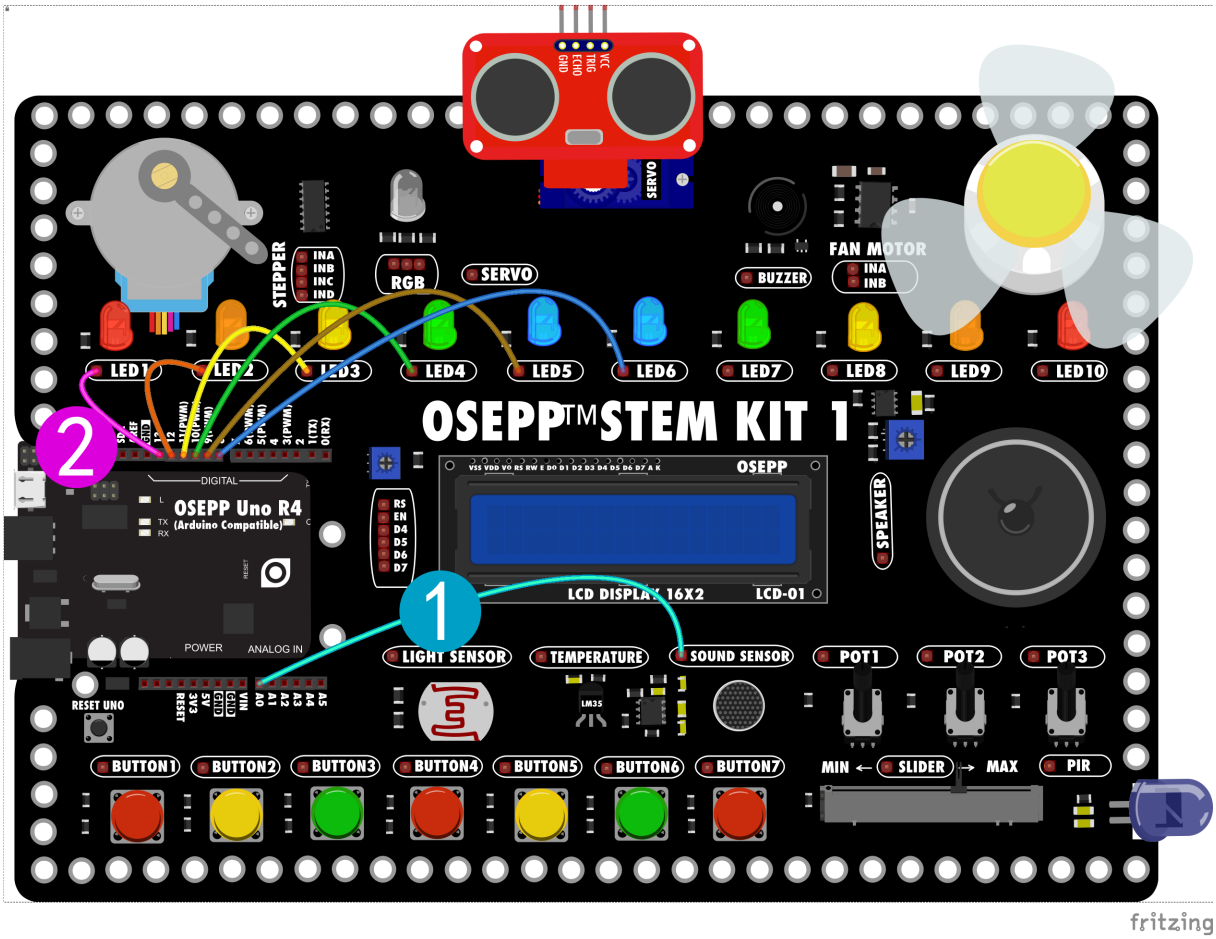
解析

麦克风是一种电声转换器件，它是把外界采集到的声音转换成电信号。麦克风接收到外界声音，产生出来的电信号用 **Arduino** 的 A0 引脚转换出来在程序里就是 0-1023 中的变化的值。

应用 2 麦克风分贝计

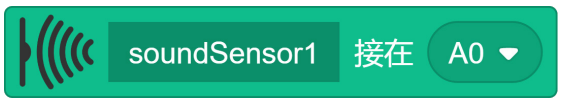
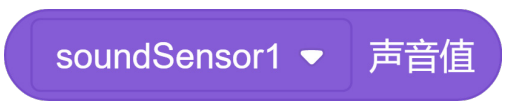
部署

1. 麦克风 Sound Sensor 连到 OSEPP UNO 的 A0 引脚。
2. 把 LED1-led6 分别连到 OSEPP UNO 的 13-8 引脚。

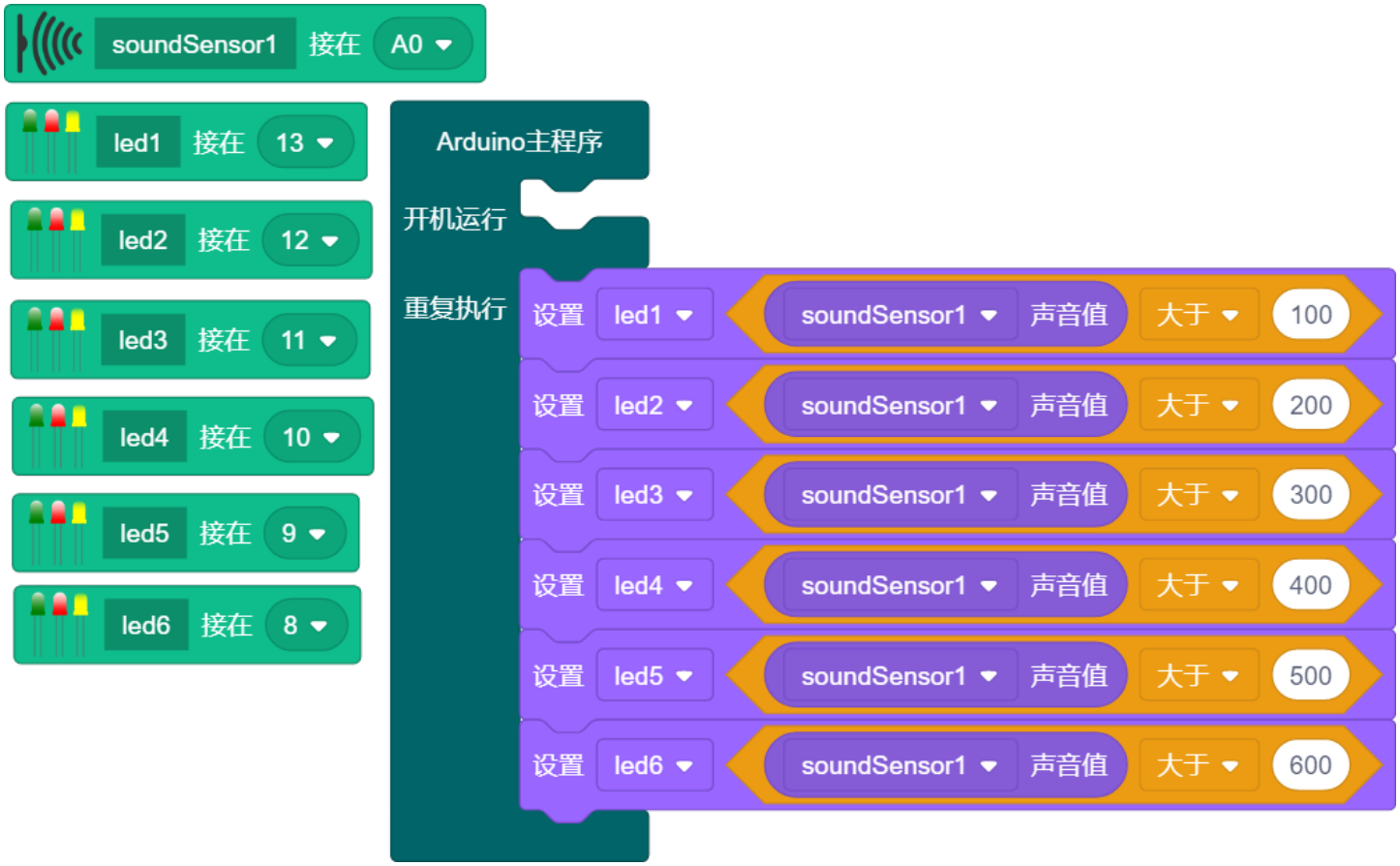


程序搭建

oseppBlock 积木知识

	麦克风积木。麦克风输出是模拟信号，直能接在 A0-A7 引脚。
	麦克风输出声音值。数值在 0-1023 之间。

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //soundSensor1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    //led1
    pinMode(13, OUTPUT); // 定义 13 号引脚为输出模式
    //led2
    pinMode(12, OUTPUT); // 定义 12 号引脚为输出模式
    //led3
    pinMode(11, OUTPUT); // 定义 11 号引脚为输出模式
    //led4
    pinMode(10, OUTPUT); // 定义 10 号引脚为输出模式
    //led5
    pinMode(9, OUTPUT); // 定义 9 号引脚为输出模式
    //led6
    pinMode(8, OUTPUT); // 定义 8 号引脚为输出模式
}

void loop()
{
    digitalWrite(13, analogRead(A0) > 100); //A0 大于 100 时 13 号引脚输出高电平
    digitalWrite(12, analogRead(A0) > 200); //A0 大于 200 时 12 号引脚输出高电平
    digitalWrite(11, analogRead(A0) > 300); //A0 大于 300 时 11 号引脚输出高电平
    digitalWrite(10, analogRead(A0) > 400); //A0 大于 400 时 10 号引脚输出高电平
    digitalWrite(9, analogRead(A0) > 500);  //A0 大于 500 时 9 号引脚输出高电平
    digitalWrite(8, analogRead(A0) > 600);  //A0 大于 600 时 8 号引脚输出高电平
}
```

运行结果

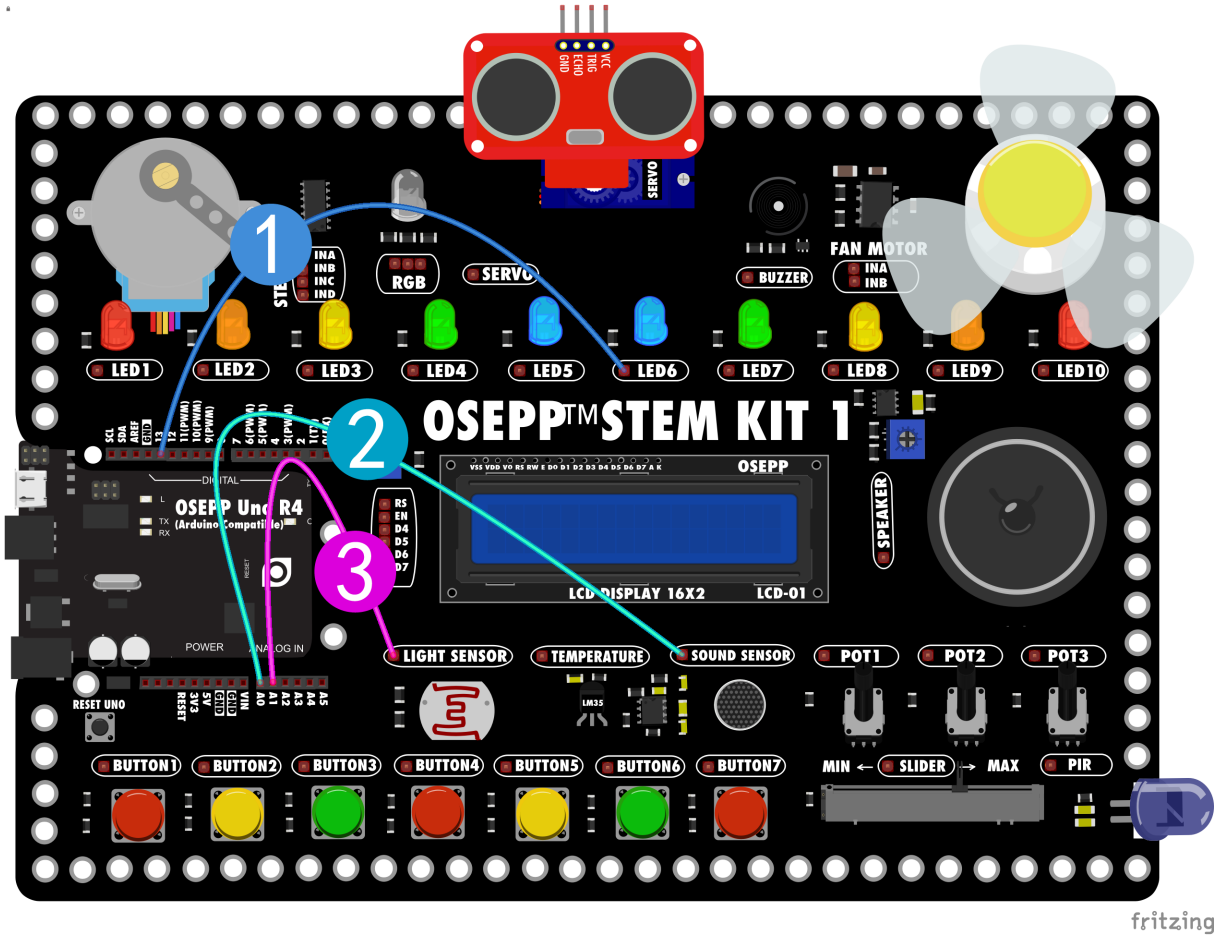
对着麦克风说话时，就会看到 LED 亮起来了，声音越大 LED 就亮得越多。上面代码里面用了比较运算，大于某个值就会输出高电平。

应用 3 声光感应灯

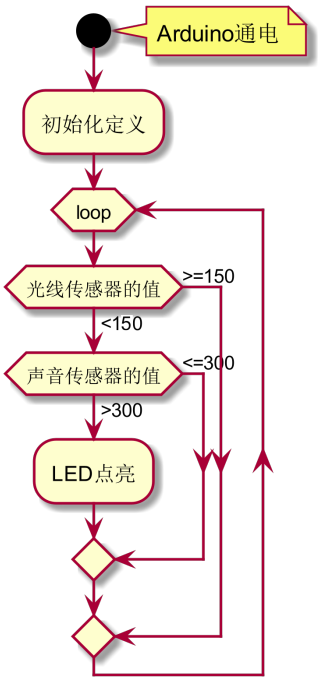
当我们走在楼道里，一拍手，楼道里面的灯就自动亮起来了，楼道里面用的声控灯就是用麦克风来控制的。我们还可以结合之前的光线感应器做一个灯，只有光线不足的时候，然后有声音的时候 LED 才会亮起，并且延时 5 秒后熄灭。

部署

1. LED6 接到 OSEPP UNO 的 13 号引脚。
2. 麦克风接到 OSEPP UNO 的 A0 引脚。
3. 光线感应器接到 OSEPP UNO 的 A1 引脚。



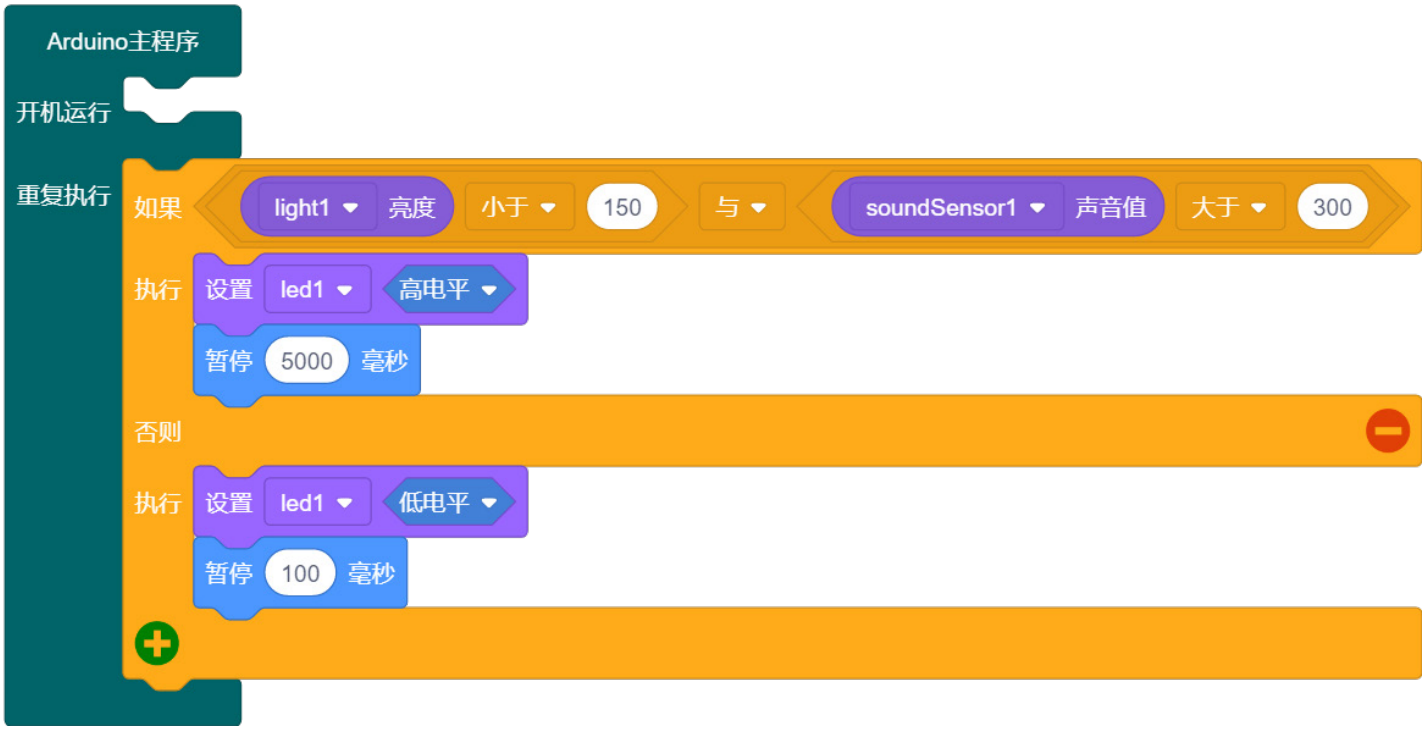
流程图



程序的值可以根据实际情况设定，光线传感器的值越低，就需要更暗的环境，声音传感器的值越高，就需要更大的声音来触发，你也可以适当的修改来看一下实际效果。

程序搭建

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //soundSensor1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    //led1
    pinMode(13, OUTPUT); // 定义 13 号引脚为输出模式
    //light1
    pinMode(A1, INPUT); // 定义 A0 为输入模式
}

void loop()
{
     if (analogRead(A1) < 150 && analogRead(A0) > 300) // 如
果 A1 小于 150 与 A0 大于 300，要同时满足这两个条件
    {
        digitalWrite(13, HIGH); //13 号引脚输出高电平
        delay(5000);             // 延时 5000 毫秒
    }
    else
    {
        digitalWrite(13, LOW); //13 号引脚输出低电平
        delay(100);             // 延时 100 毫秒
    }
}
```

运行结果

当光线变暗，光线传感器的值小于 150，并且声音传感器的值要大于 300，此时就会执行下面的程序。点亮 LED 并且延时 5 秒后熄灭，否则 LED 保持低电平状态。

解析

逻辑运算符

<table>
<tr><td colspan="2">代码里面 用到了一个新的运算符号 与 (&&) ，常用的运算符还有 或 (||) 和非 (!) 。</td></tr>
<tr><td> 
</td><td>代码里面用 && 表示 ，就像串联关系。如果第一个条件与第二个条件皆为 " 真 ", 传回 " 真 "(true) 值。也就是条件都同时满足时，才执行。

代码:

if(a&&b)</td></tr>
<tr><td> 
</td><td>只有当 a 和 b 同时为真时才能为真，其余为假。
代码里面用 || 表示 ，就像并联关系。如果第一个条件和第二个条件，其中只要有一个为 " 真 "，传回 " 真 "(true) 值。也就是满足给定条件其中的一个，就执行。

代码:

if(a||b)</td></tr>
<tr><td> 
</td><td>只有当 a 和 b 同时为假时才能为假，其余为真。
代码里面用 ! 表示 ，如果条件为 " 假 "，传回 " 真 "(true) 值。也就是不符合给定条件，就执行。

代码:

if(!a)

!a，若 a 为真，则 !a 为假；若 a 为假，则 !a 为真。</td></tr>
</table>

温度传感器

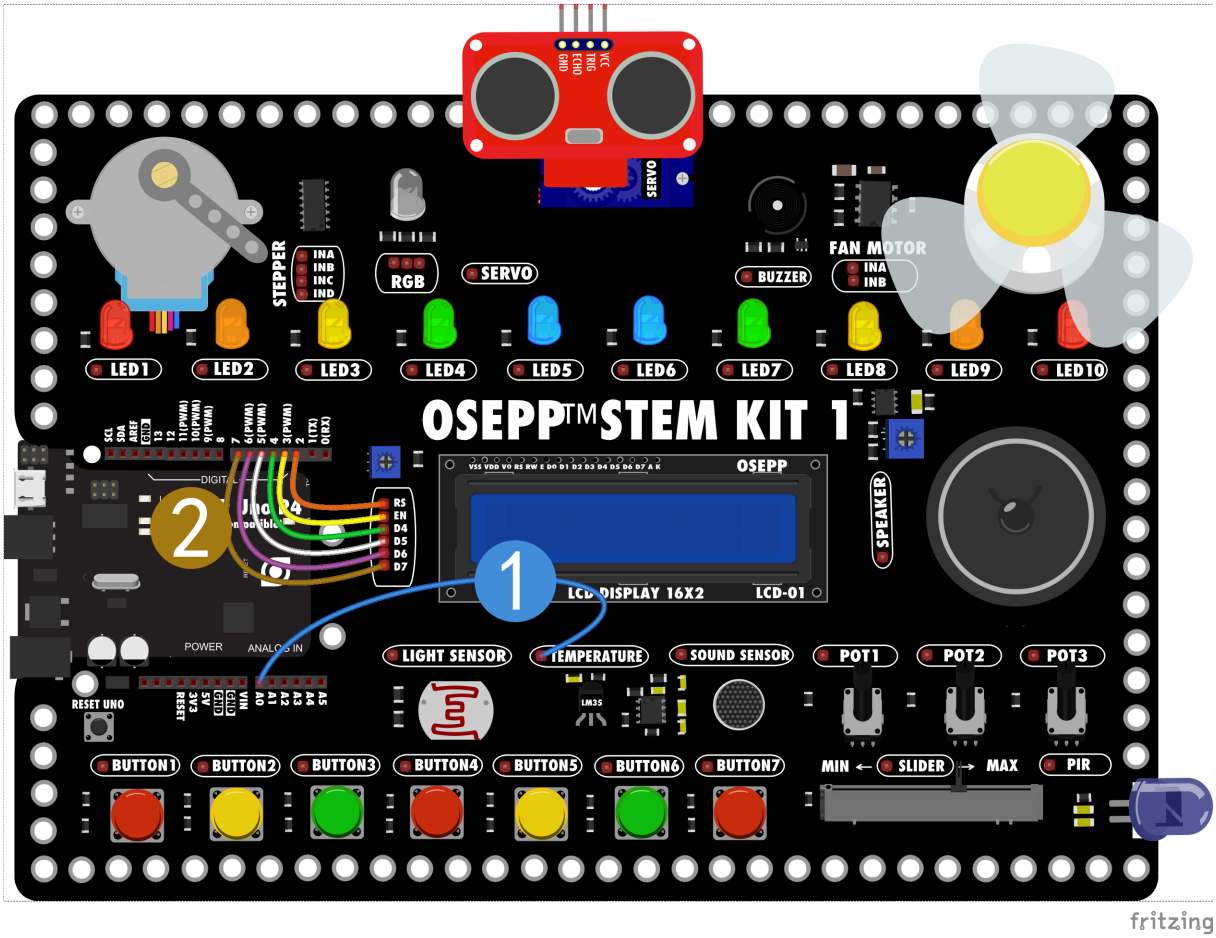
温度传感器一般是指将温度转化为电子数据的元器件。不少材料、元件的特性都随温度的变化而变化，所以能作为温度传感器的材料相当多。温度传感器随温度而引起物理参数变化的有：膨胀、电阻、电容、和电动势、磁性能、频率、光学特性及热噪声等等。

LCD 显示温度传感器的值

学习板上面用的是 **LM35 温度传感器**。如果给温度传感器加上电压，当外界温度变化时，**LM35 温度传感器**就会输出一个变化的电压。学习板内部已经接好电源，我们只需要把传感器的引脚接到 **OSEPP UNO** 的引脚就可以了。

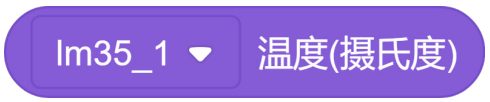
部署

1. 温度传感器 Temperature 的端子接到 **OSEPP UNO** 的 A0 引脚。
2. LCD 的接口 RS-D7 分别连接到 **OSEPP UNO** 的 2~7 号引脚。



程序搭建

oseppBlock 积木知识

	温度传感器模块积木。传感器输出是模拟信号，直能接在 A0-A7 引脚。
	温度传感器返回值积木，单位是摄氏度。

oseppBlock 程序

Arduino 程序

```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚

void setup()
{
    //lm35_1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    lcd1.begin(16, 2);  //LCD 初始化
}

void loop()
{
    lcd1.clear();                                  // 清屏
    lcd1.setCursor(0, 0);                          // 显示光标定位
    lcd1.print(analogRead(A0) * 0.48828125); // 计算温度并显示
    delay(1000);                                   // 延时 1000 毫秒
}
```

运行结果

这时就能看到 LCD 上显示当前的环境温度，单位是摄氏度℃。用手压住温度传感器时，LCD 显示的温度传感器的温度值就会有变化。

电动马达

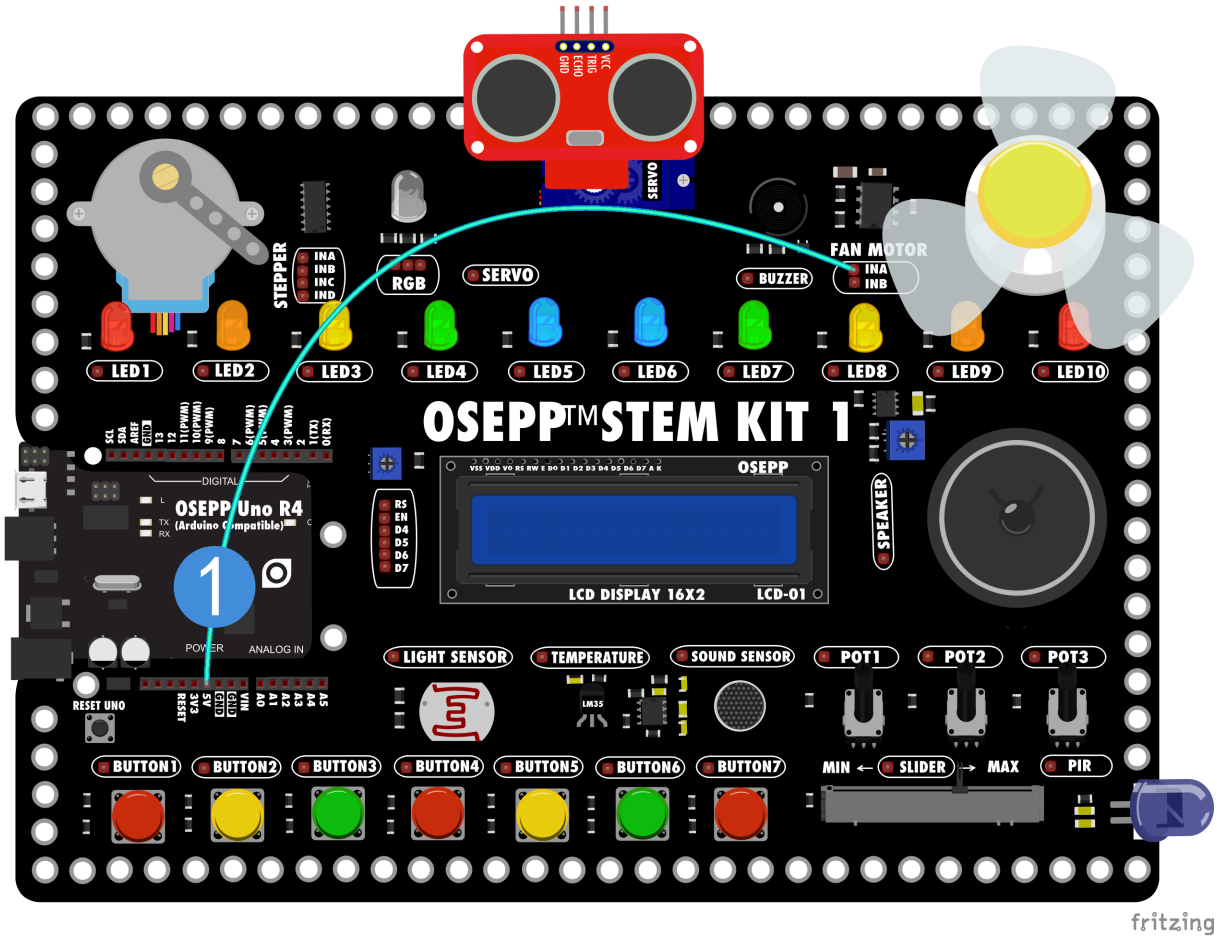
电动马达 (Electric motor)，又称为马达或电动机，是一种将电能转化成机械能，并可再使用机械能产生动能，用来驱动其他装置的电气设备。

马达的旋转原理的依据为佛来明左手定则或是右手开掌定则，当一导线置放于磁场内，若导线通上电流，则导线会切割磁场线使导线产生移动。电流进入线圈产生磁场，利用电流的磁效应，使电磁铁在固定的磁铁内连续转动的装置，可以将电能转换成动能。与永久磁铁或由另一组线圈所产生的磁场互相作用产生动力。

直流马达的原理是定子不动，转子依相互作用所产生作用力的方向运动。

应用 1 马达小风扇

马达是一种把电能转化成动能的元件，我们只要给其通上电，马达就能转起来。
把 OSEPP UNO 的 5V 电压引脚连到电动马达 (Fan Motor) 的 INA 端子。

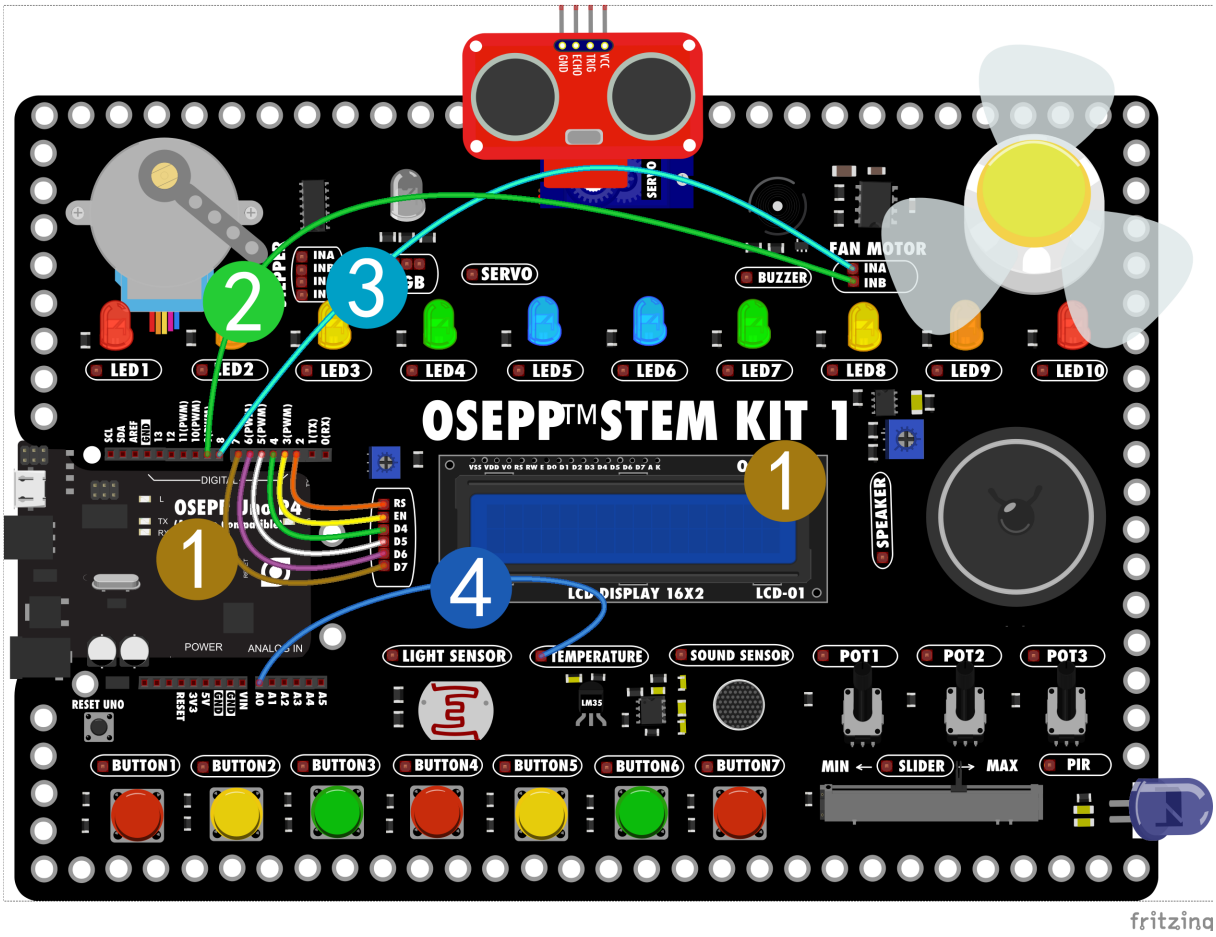


这时就可以看到马达在转动了，
如果连到 INB 的话，马达就会和之前的旋转方向相反。

部署

结合之前学的温度传感器，来做一个温度感应的小风扇。当温度达到设定的阀值时，小风扇就会转动起来。温度低于某一个温度值，风扇就会停止。

1.LCD 的接口 RS-D7 分别连接到 OSEPP UNO 的 2~7 号引脚。
2. 电动马达 (Fan Motor) 的 INA 端子接到 OSEPP UNO 的 9 号引脚。
3. 电动马达 (Fan Motor) 的 INB 端子接到 OSEPP UNO 的 8 号引脚。
4. 温度传感器接到 OSEPP UNO 的 A0 引脚。

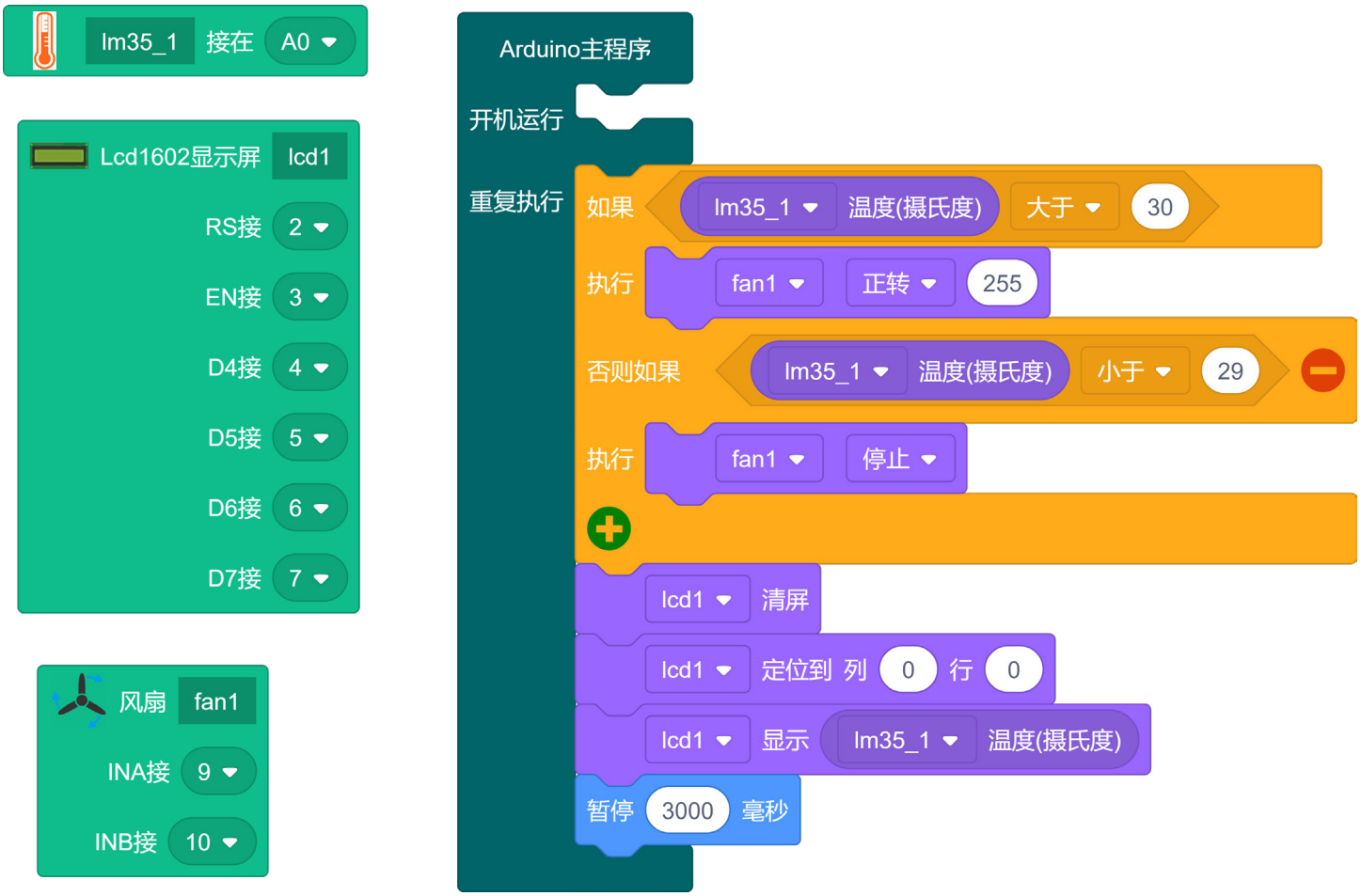


程序搭建

oseppBlock 积木知识

	电动马达设定积木。 INA 是顺时针驱动引脚，INB 是逆时针驱动引脚。
	电动马达驱动积木。可以选择正转和反转，正转是顺时针，反转是逆时针。设定值的范围是 0-255。

oseppBlock 程序



Arduino 程序

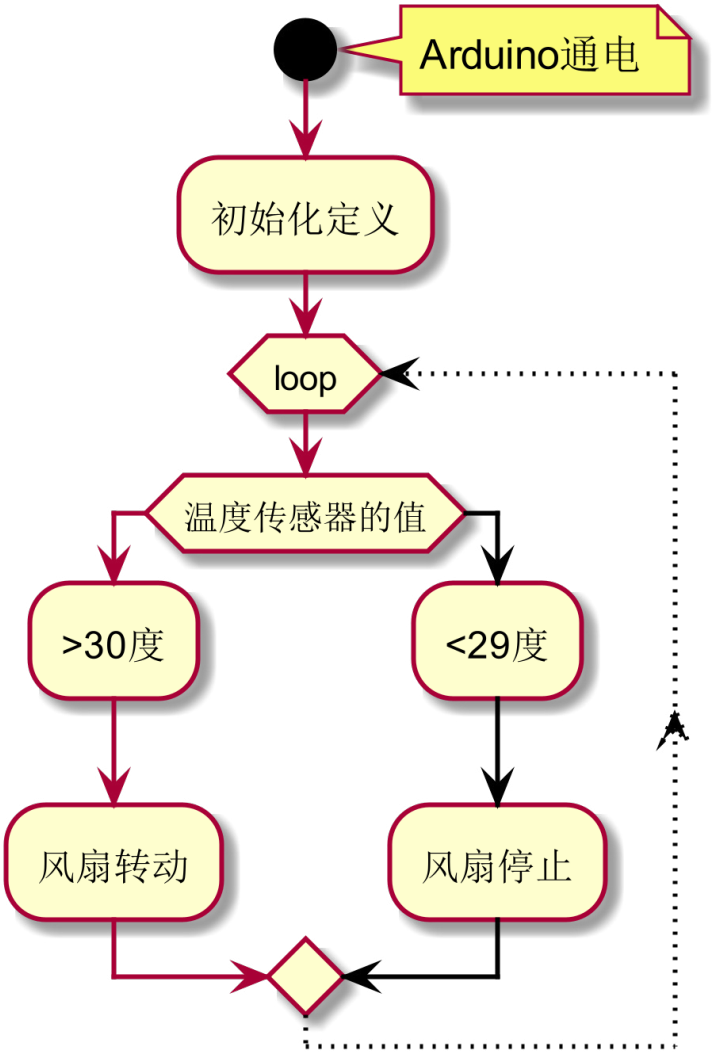
```
#include <LiquidCrystal.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚

void setup()
{
    lcd1.begin(16, 2);
    //lm35_1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    //fan1
    pinMode(9, OUTPUT);  // 定义 9 号引脚为输出模式
    pinMode(10, OUTPUT); // 定义 10 号引脚为输出模式
}

void loop()
{
    if (analogRead(A0) * 0.48828125 > 30) // 如果温度值大于 30 度
    {
        analogWrite(9, 255); //3 号引脚输出 pwm 为 255
        analogWrite(10, 0);  //5 号引脚输出 pwm 为 0
    }
    else if (analogRead(A0) * 0.48828125 < 29) // 如果温度值小于 29 度
    {
        analogWrite(9, 0);  //2 号引脚输出 pwm 为 0
        analogWrite(10, 0); //5 号引脚输出 pwm 为 0
    }
    lcd1.clear();                                  // 清屏
    lcd1.setCursor(0, 0);                          // 显示光标定位
    lcd1.print(analogRead(A0) * 0.48828125); // 显示温度
    delay(3000);                                   // 延时 1000 毫秒
}
```

运行结果

如果你当前实验环境温度高于 30°C时，小风扇就会转动起来。如果低于 29°C就停止转动。你可以根据显示屏上面显示你当前的环境温度来修改这两个值，让风扇在这个值的变化中间启停。

解析

流程图



如果温度传感器感应到的温度大于 30°C，风扇就会转动。代码里面判断 > 30，大于 30°C。当温度刚好是 30°C时，风扇是不会转动的，要 31°C才可以。

如果要求达到 30°C就转动的话，代码要用 if (analogRead(A0) * 0.48828125 >= 30)。就是把大于符号改成大于等于符号。小于 29°C同理。

应用 2 用 PWM 来控制电动马达

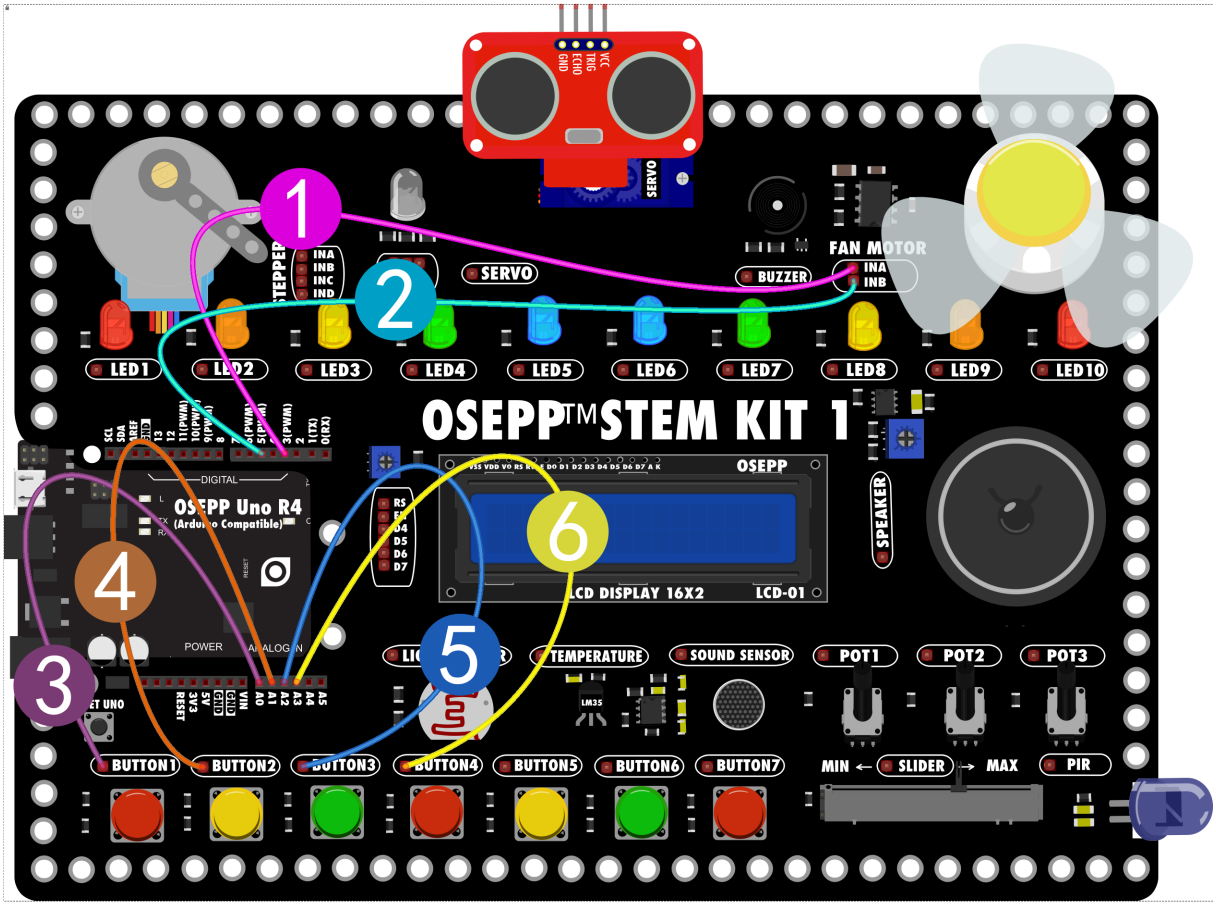
当我们给马达通电时马达就会转动，断电就会停止。如果我们不停的缩短这个通电和断电的时间间隔，是不是间隔越短马达就会转得越快，间隔越长就会转得越慢呢？
PWM 就是利用这个原理来实现马达调速的。

脉冲宽度调制（**英语：Pulse Width Modulation，**缩写：PWM），简称脉宽调制，是将模拟信号变换为脉冲的一种技术，一般变换后脉冲的周期固定，但脉冲的工作周期会依模拟信号的大小而改变。

部署

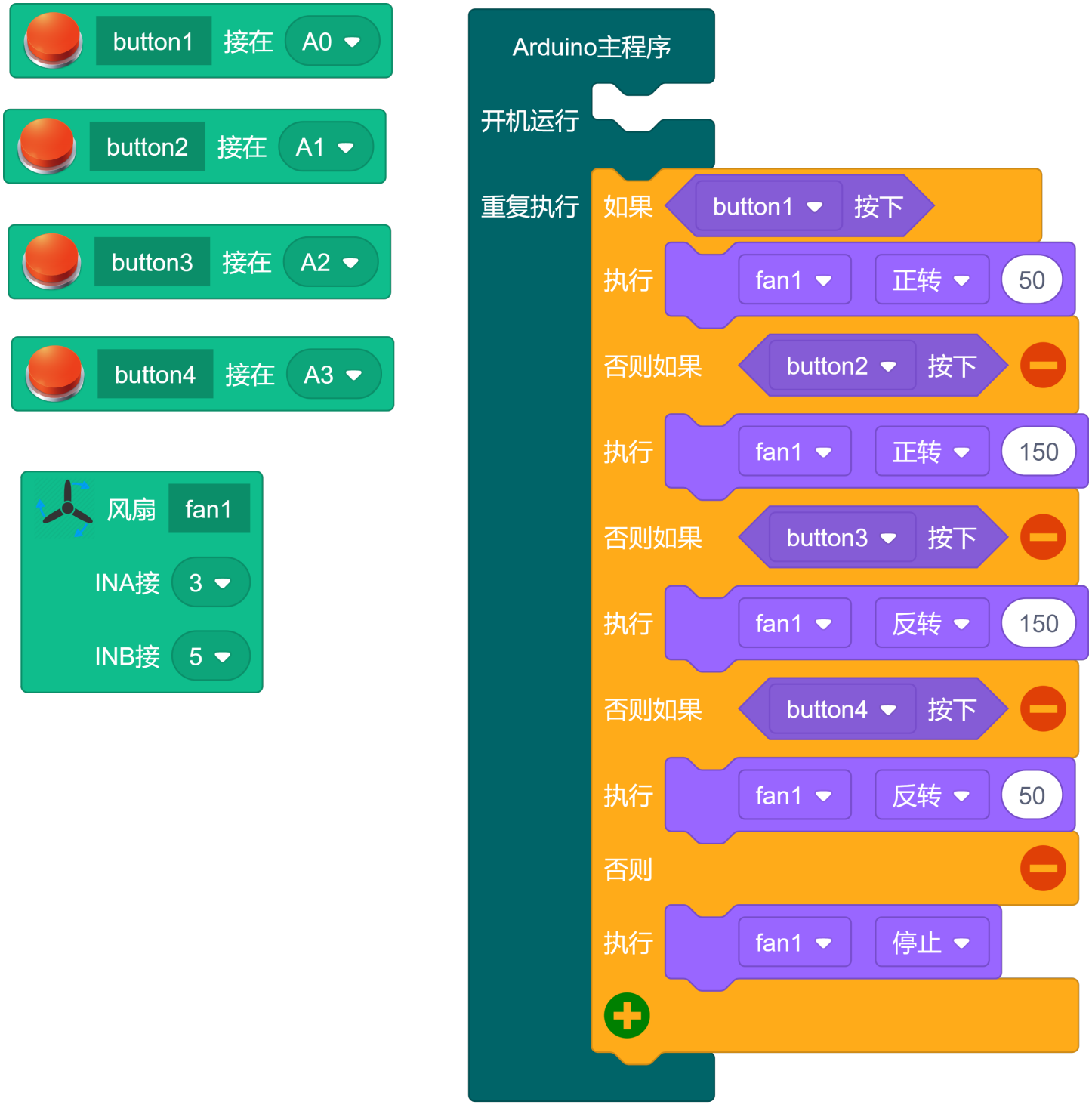
按照下表连接好导线

序号	元件端子	osepp UNO 引脚号
1	(Fan Motor) 的 INA	3
2	(Fan Motor) 的 INB	5
3	开关 Button1	A0
4	开关 Button2	A1
5	开关 Button3	A2
6	开关 Button4	A3



程序搭建

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //button1
    pinMode(A0, INPUT); // 定义开关 1
    //button2
    pinMode(A1, INPUT); // 定义开关 2
    //button3
    pinMode(A2, INPUT); // 定义开关 3
    //button4
    pinMode(A3, INPUT); // 定义开关 4
    //fan1
    pinMode(3, OUTPUT); // 定义 3 号引脚为输出模式
    pinMode(5, OUTPUT); // 定义 5 号引脚为输出模式
}

void loop()
{
    if (digitalRead(A0) == LOW) // 如果开关 1 按下
    {
        analogWrite(3, 50); //3 号引脚输出 pwm 为 50
        analogWrite(5, 0);  //5 号引脚输出 pwm 为 0
    }
    else if (digitalRead(A1) == LOW) // 如果开关 2 按下
    {
        analogWrite(3, 150); //3 号引脚输出 pwm 为 150
        analogWrite(5, 0);   //5 号引脚输出 pwm 为 0
    }
    else if (digitalRead(A2) == LOW) // 如果开关 3 按下
    {
        analogWrite(3, 0);   //3 号引脚输出 pwm 为 0
        analogWrite(5, 150); //5 号引脚输出 pwm 为 150
    }
    else if (digitalRead(A3) == LOW) // 如果开关 4 按下
    {
        analogWrite(3, 0);  //3 号引脚输出 pwm 为 0
        analogWrite(5, 50); //5 号引脚输出 pwm 为 50
    }
    else
    {
        analogWrite(3, 0); // 马达停止转动
        analogWrite(5, 0);
    }
}
```

运行结果

当开关 Button1 按下，风扇低速正转，开关 Button2 按下高速正转，开关 Button3 按下，风扇低速反转，开关 Button4 按下高速反转。利用 PWM 输出模式可以控制速度，但方向是利用不同的引脚输出信号来实现的。

解析

在模拟电路中，模拟信号的值可以连续进行变化，在时间和值的幅度上都几乎没有限制，基本上可以取任何实数值，输入与输出也呈线性变化。所以在模拟电路中，电压和电流可直接用来进行控制对象，例如家用电器设备中的音量开关控制、采用卤素 LED 灯具的亮度控制等等。但模拟电路有诸多的问题：例如控制信号容易随时间漂移，难以调节；功耗大；易受噪声和环境干扰等等。

与模拟电路不同，数字电路是在预先确定的范围内取值，在任何时刻，其输出只可能为 ON 和 OFF 两种状态，所以电压或电流会以**通 / 断**方式的重复脉冲序列加载到模拟负载。

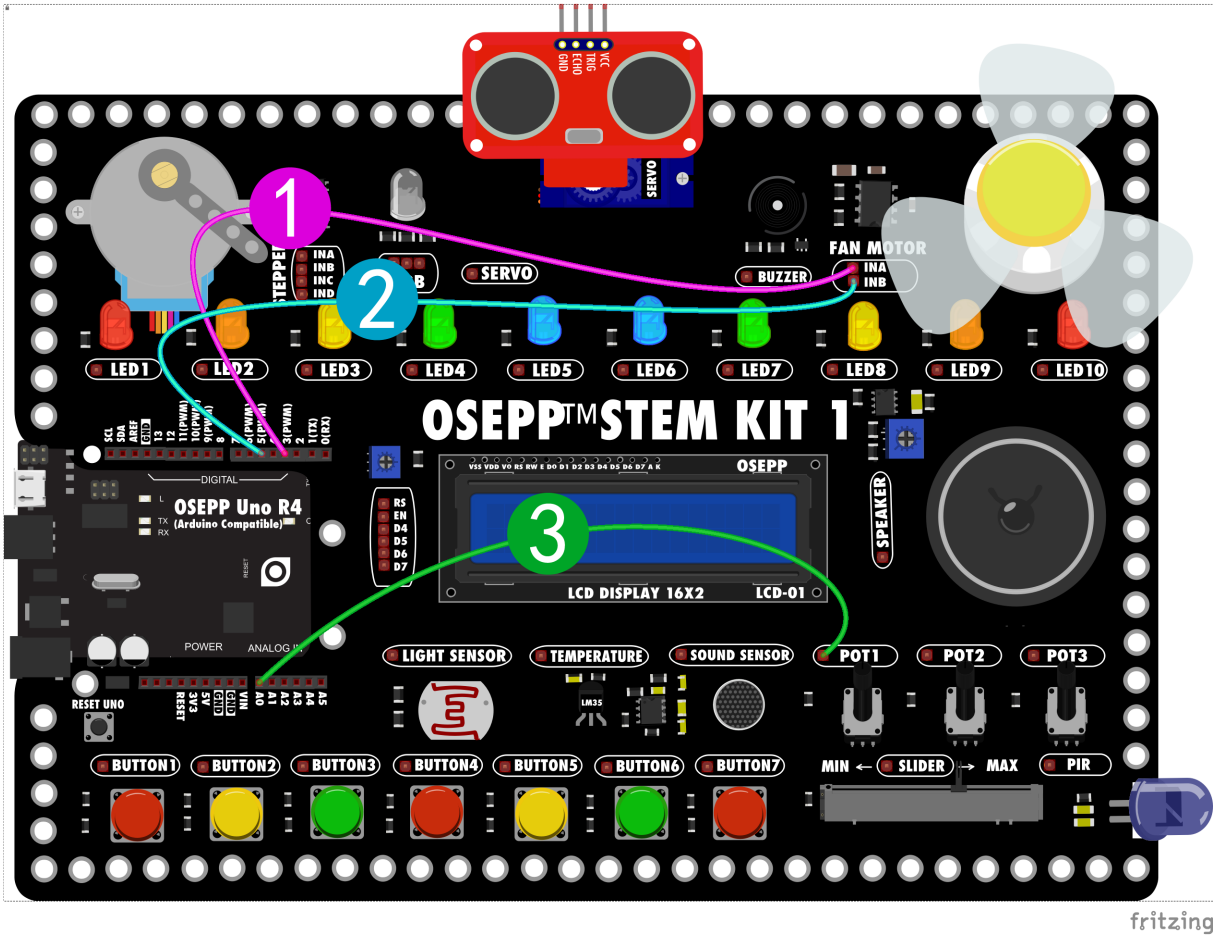
PWM 技术是一种对模拟信号电平的数字编码方法，通过使用高分辨率计数器 (调制频率) 调制方波的占空比，从而实现对一个模拟信号的电平进行编码。其最大的优点是从处理器到被控对象之间的所有信号都是数字形式的，无需再进行数模转换过程；而且对噪声的抗干扰能力也大大增强 (噪声只有在强到足以将逻辑值改变时，才可能对数字信号产生实质的影响)，这也是 **PWM** 在通讯等信号传输行业得到大量应用的主要原因。目前在很多微型控制器 **(MCU)** 内部都包含有 **PWM** 控制器模块。

应用 3 无级调速小风扇

用电位器来调节马达的转速。无极变速，实际是把风扇的速度分成 255 个等级，类似 255 个档位，这样我们就感觉不到档位的存在了。

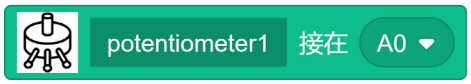
部署

1. 电动马达 (Fan Motor) 的 INA 端子接到 OSEPP UNO 的 3 号引脚。
2. 电动马达 (Fan Motor) 的 INB 端子接到 OSEPP UNO 的 5 号引脚。
3. 电位器 POT1 接到 A0 引脚。



程序搭建

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //potentiometer1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    //fan1
    pinMode(3, OUTPUT); // 定义 3 号引脚为输出模式
    pinMode(5, OUTPUT); // 定义 5 号引脚为输出模式
}

void loop()
{
    analogWrite(3, map(analogRead(A0), 0, 1023, 0, 255));
                                                //3 号引脚输出 pwm 为 A0 的映射值
    analogWrite(5, 0);                                  //5 号引脚输出 pwm 为 0
    delay(1);                                           // 延时 1 毫秒
}
```

运行结果

转动电位器，风扇的转速也会产生变化。电位器从最左边顺时针旋转，风扇的速度也会慢慢的增加。风扇速度的增减有一个很平滑的过度。

解析

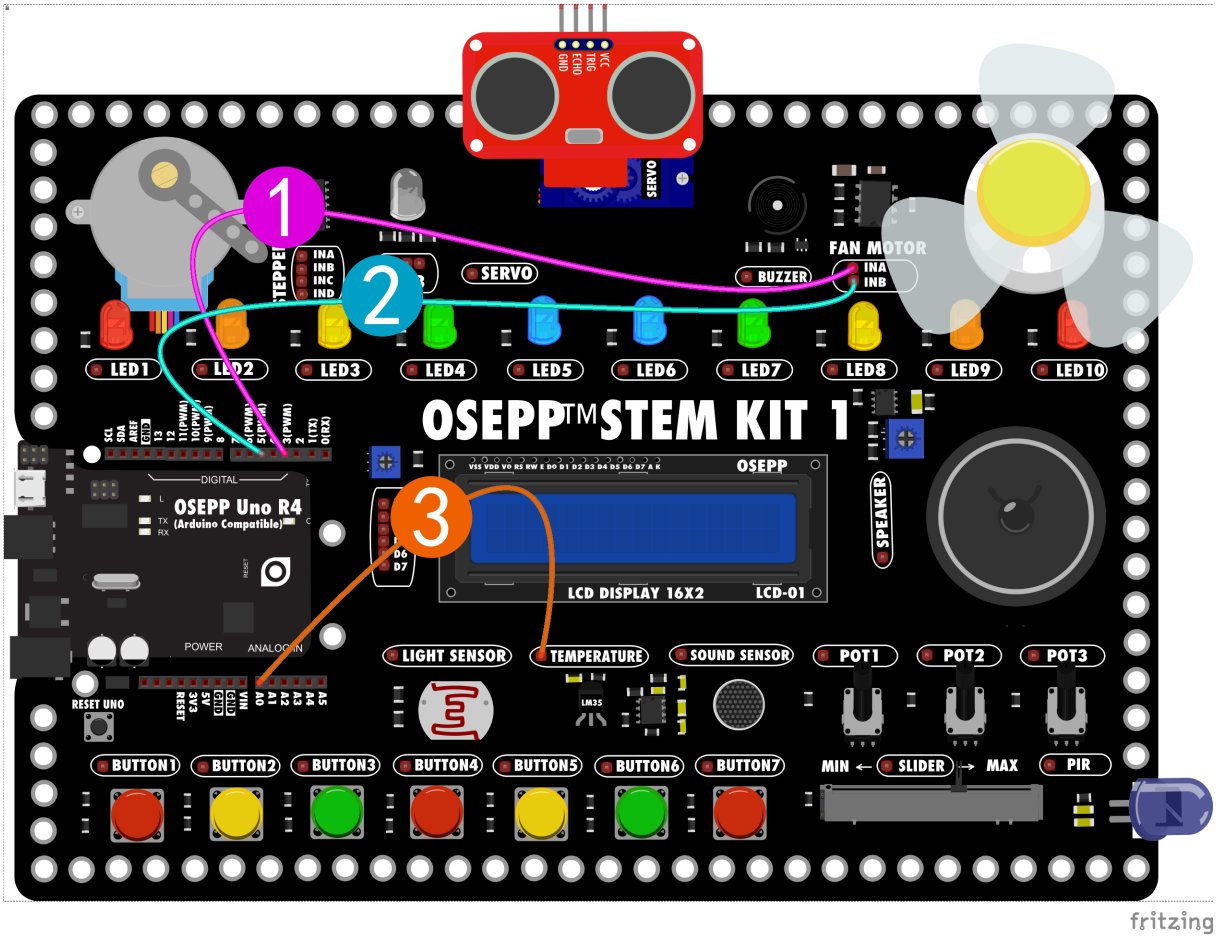
程序里 **Arduino** 首先会读取电位器的值，最小值是 **0**，最大是 **1023**。然后把这个值映射到 **0-255** 输出 **PWM** 信号驱动风扇。

应用 4 温控风扇

结合温度传感器就可以利用温度来控制风扇电机的开关。当温度低时，风扇转速就慢，温度高时，风扇转速就快。当低于某个温度时风扇就停转。这样就做成了一个自动启停的温控调速风扇。

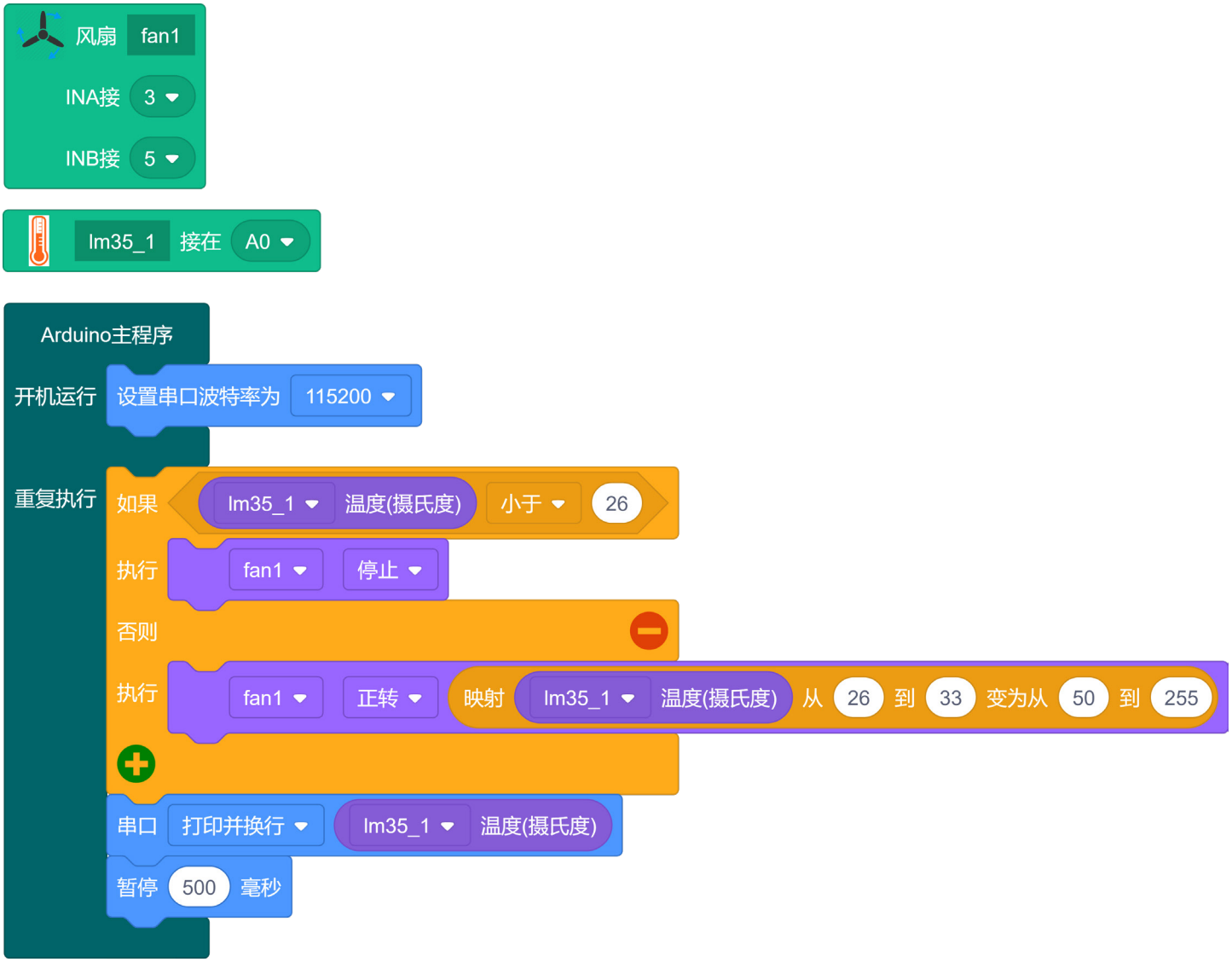
部署

1. 电动马达 (Fan Motor) 的 INA 端子接到 OSEPP UNO 的 3 号引脚。
2. 电动马达 (Fan Motor) 的 INB 端子接到 OSEPP UNO 的 5 号引脚。
3. 温度传感器 Temperature 连接到 A0 引脚。



程序搭建

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //lm35_1
    pinMode(A0, INPUT); // 定义 A0 为输入模式
    //fan1
    pinMode(3, OUTPUT); // 定义 3 号引脚为输出模式
    pinMode(5, OUTPUT); // 定义 5 号引脚为输出模式
    Serial.begin(115200); // 设置串口波特率
}

void loop()
{
    if (analogRead(A0) * 0.48828125 < 26) // 如果温度小于 26 度
    {
        analogWrite(3, 0); // 风扇停止转动
        analogWrite(5, 0);
    }
    else
    {
        analogWrite(3, map(analogRead(A0) * 0.48828125, 26, 33, 50, 255));
        // 温度在 26 自 33 之间映射至风扇转速在 20-100%
        analogWrite(5, 0);
    }
    Serial.println((analogRead(A0) * 0.48828125)); // 串口显示温度
    delay(500);                                    // 延时 500 毫秒
}
```

运行结果

实验里设定了一个温度值 26，当温度传感器的温度超过这个值时，风扇就会转动起来。当温度为 26℃时转速大约是 20%，温度到达 33℃时，转速达到 100%。这里设置的 26 这个数值要根据你当时的环境温度变化值来设定。
程序里面加了串口打印，可以在串口看到温度显示，设定温度的值时请参考串口显示这个值的变化。

步进电机

步进电机 ------**stepping motor**

步进电机又称脉冲电机，它是一种感应电机，涉及到机械、电机、电子及计算机等许多专业知识。步进电机作为执行元件，是机电一体化的关键产品之一，广泛应用在各种自动化控制系统中。

什么是步进电机?

步进电机是一种将电脉冲转化为角位移的执行机构。当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度（及步进角）。您可以通过控制脉冲个来控制角位移量，从而达到准确定位的目的。同时您可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

主要特征

步进马达只需要通过脉波信号的操作，即可简单实现高精度的定位，并使工作物在目标位置高精度地停止。步进马达是以基本步级角的角度为单位来进行定位。

实验用的 **28BYJ-48** 步进电机参数:

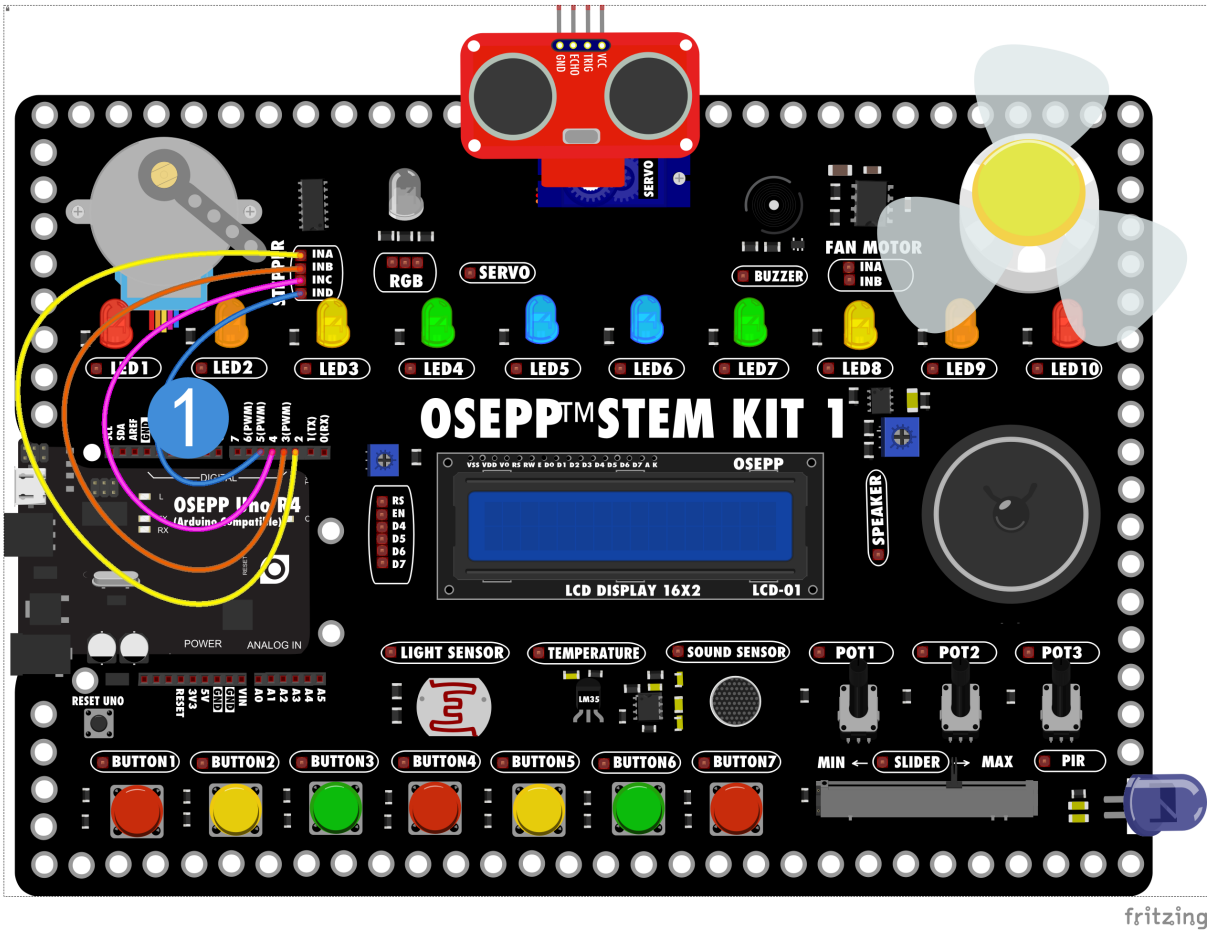
4 步控制信号序列：**11.25 度 / 步，32 步旋转一周。**
4 步模式下旋转一周将用：**32（步 / 周）X64（齿轮比） = 2048 步。**

应用 1 Arduino 驱动步进电机

实验用的步进马达一圈是 2048 步，步进电机转一圈需要给其 2048 个脉冲信号。那用 2048 除以 60 就可以做成一个类似秒针的效果了。

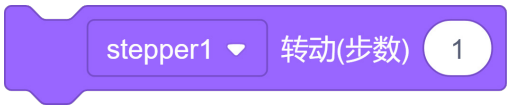
部署

OSEPP UNO 的 2-5 号引脚连到步进电机 IN1-IN4。



程序搭建

oseppBlock 积木知识

	步进电机模块积木。定义步进电机的名称，引脚和一些参数。一般不需要更改。
	步进电机驱动积木。设定步进电机的转动步数，可以是任意整数。
	步进电机转动速度积木。如需改变，请设定值在 20 以内，超出这个范围步进电机无法启动。

oseppBlock 程序

Arduino 程序

```
#include <Stepper.h>

Stepper stepper1(2048, 2, 4, 3, 5); // 定义步进电机的步数和引脚

void setup()
{
    stepper1.setSpeed(10); // 定义每分钟转速
}

void loop()
{
    stepper1.step(-2048 / 60); // 每次执行步数
    delay(1000);               // 延时 1000 毫秒
}
```

运行结果

代码上传后，步进电机就像时钟的秒针一样转动。

解析

由于 `60` 不能被 `2048` 整除，所以每走几圈就会有一秒的误差。

$2048/60 = 34...8$，每走 4 圈就差不多慢了一秒。

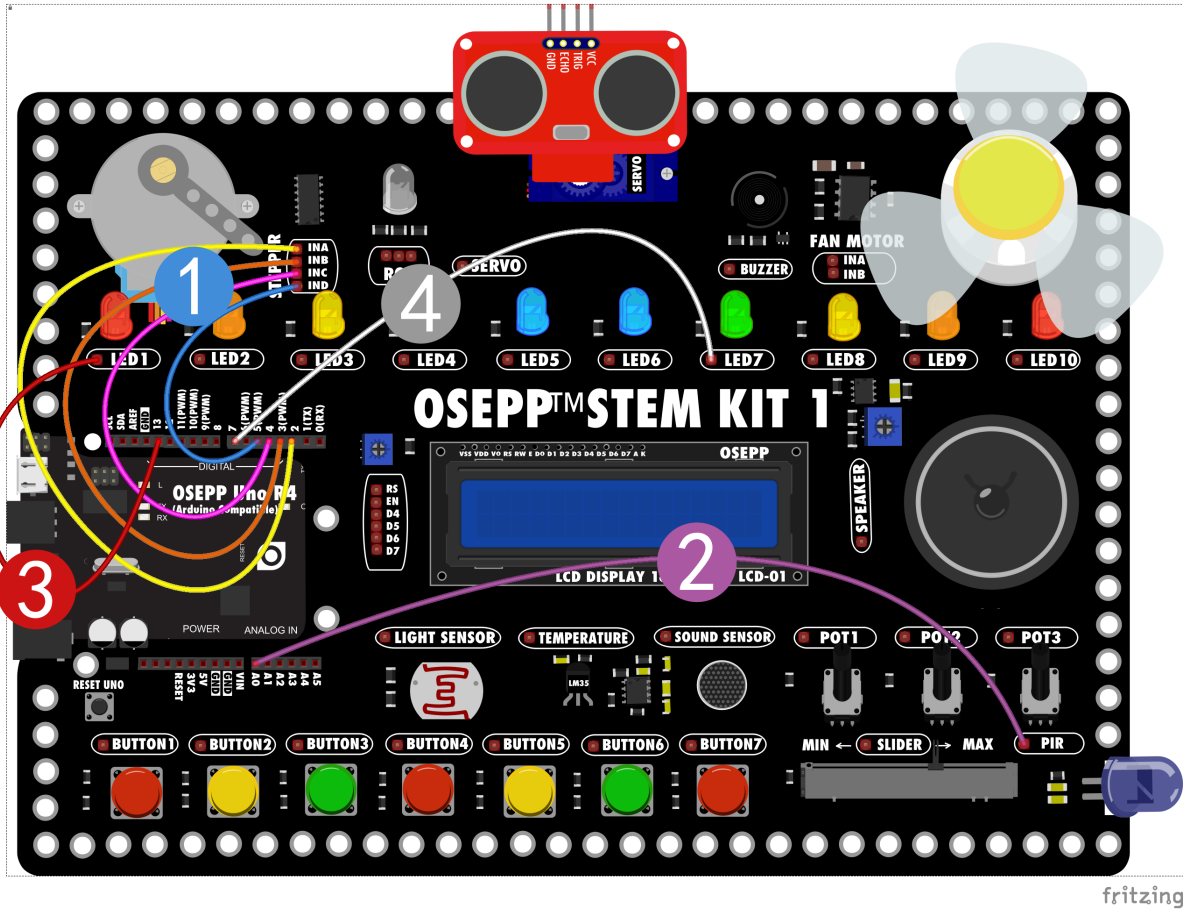
应用 2 自动感应门

当步进电机和 PIR 结合会用到什么场景呢?

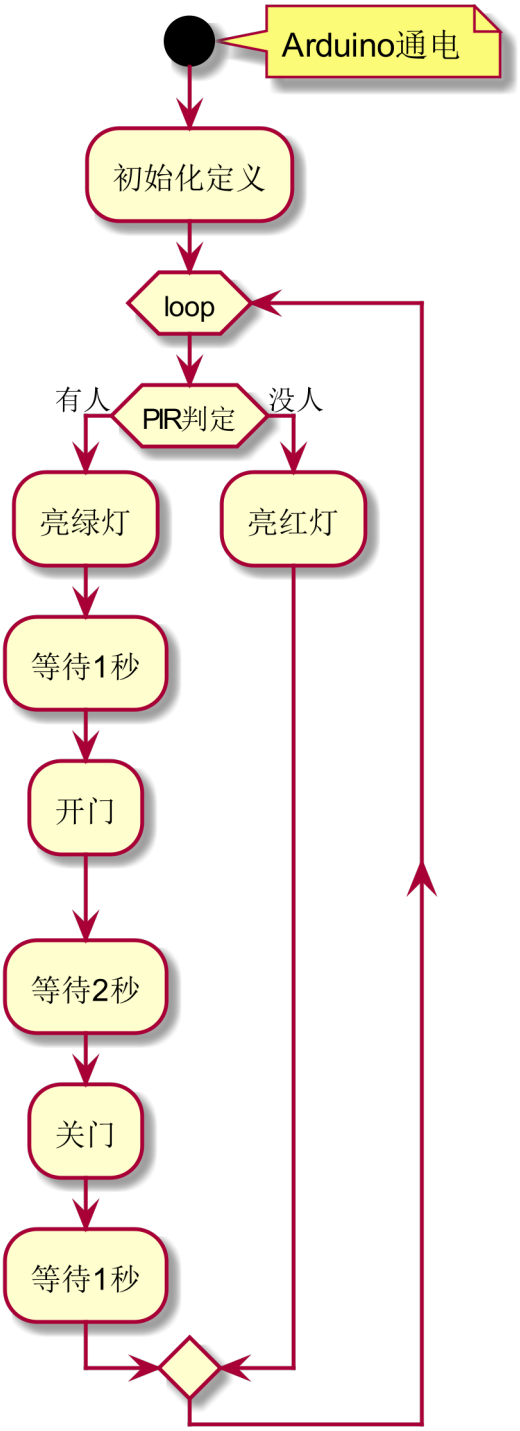
自动开门，如果有人走到门边，PIR 感应到有人靠近，然后把门打开。等人通过后门会自动关闭。
门打开的时候显示绿灯，关闭的时候显示红灯。

部署

1. OSEPP UNO 的 2-5 号引脚连到步进电机 INA-IND。
2. 把 PIR 接到 OSEPP UNO 的 A0 引脚。
3. LED1 接到 OSEPP UNO 的 13 号引脚。
4. LED7 接到 OSEPP UNO 的 7 号引脚。

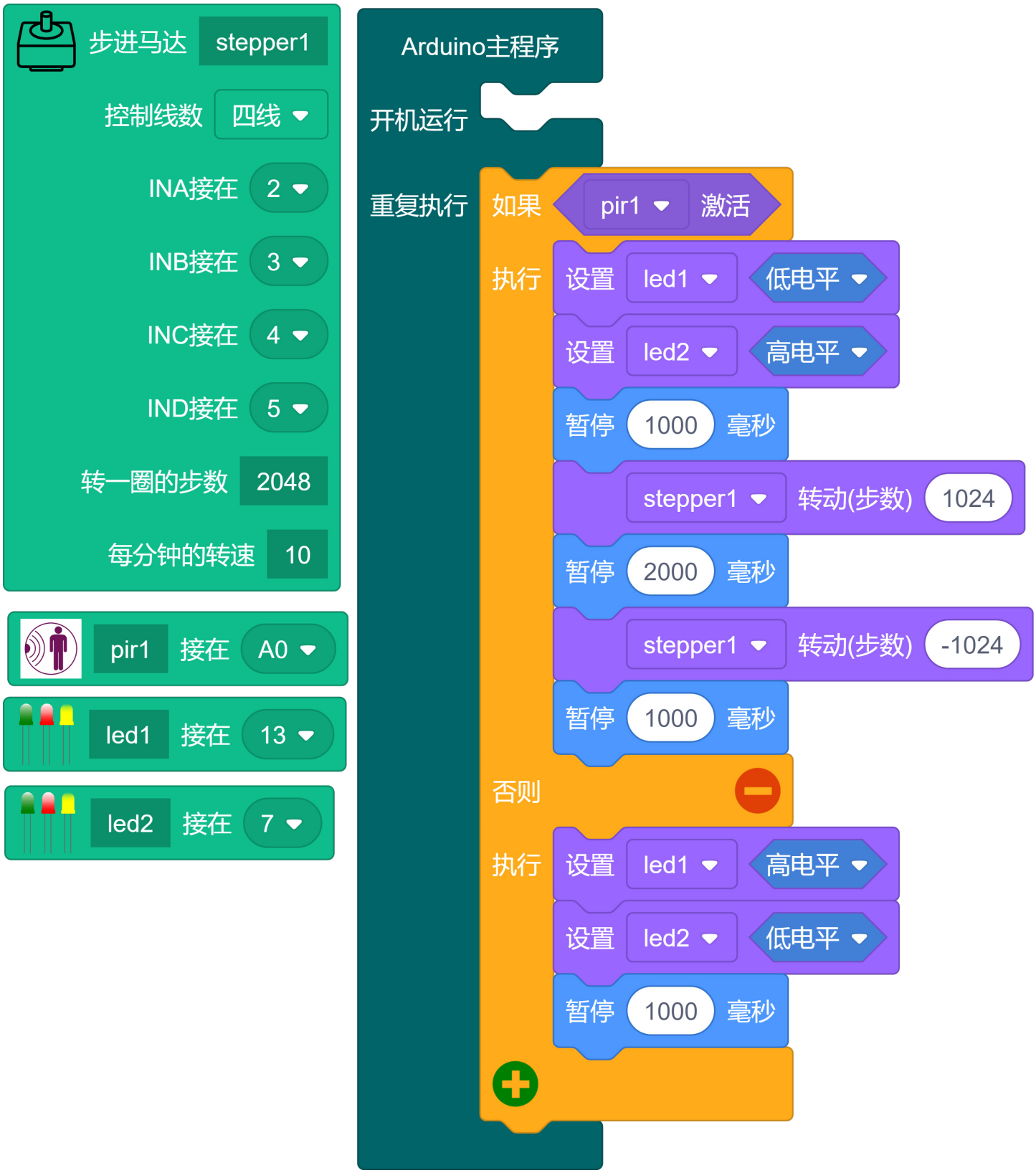


流程图



程序搭建

oseppBlock 程序



Arduino 程序

```
#include <Stepper.h>

Stepper stepper1(2048, 2, 4, 3, 5); // 定义步进电机的步数和引脚

void setup()
{
    stepper1.setSpeed(10); // 定义每分钟转速
    //pir1
    pinMode(A0, INPUT); // 定义 PIR 引脚
    //led1
    pinMode(13, OUTPUT); // 定义 LED1 引脚
    //led2
    pinMode(7, OUTPUT); // 定义 LED2 引脚
}

void loop()
{
    if (digitalRead(A0)) // 读取 PIR 的值
    {
        digitalWrite(13, LOW); // 红色熄灭
        digitalWrite(7, HIGH); // 绿色点亮
        delay(1000);           // 延时 1000 毫秒
        stepper1.step(1024);   // 步进电机正向转动 1024 步
        delay(2000);           // 延时 2000 毫秒
        stepper1.step(-1024);  // 步进电机反向转动 1024 步
        delay(1000);           // 延时 1000 毫秒
    }
    else
    {
        digitalWrite(13, HIGH); // 红色点亮
        digitalWrite(7, LOW);   // 绿色熄灭
        delay(1000);            // 延时 1000 毫秒
    }
}
```

运行结果

当 PIR 检测到人体红外热源时，触发高电平。绿色 LED 点亮，步进马达正向转动半圈，等待 2 秒后反转半圈回来。然后红色 LED 点亮，绿色 LED 熄灭。

舵机

舵机是一种位置（角度）伺服的驱动器，适用于那些需要角度不断变化并可以保持的控制系统。在高档遥控玩具，如飞机、潜艇模型，遥控机器人中已经得到了普遍应用。

舵机主要适用于那些需要角度不断变化并可以保持的控制系统，比如人形机器人的手臂和腿，车模和航模的方向控制。舵机的控制信号实际上是一个脉冲宽度调制信号（PWM 信号），该信号可由 FP-GA 器件、模拟电路或单片机产生。

舵机简单的说就是集成了直流电机、电机控制器和减速器等，并封装在一个便于安装的外壳里的伺服单元。舵机可以根据你的指令旋转到 0 至 180 度之间的任意角度然后精准的停下来。能够利用简单的输入信号比较精确的转动给定角度的电机系统。舵机安装了一个电位器（或其它角度传感器）检测输出轴转动角度，控制板根据电位器的信息能比较精确的控制和保持输出轴的角度。

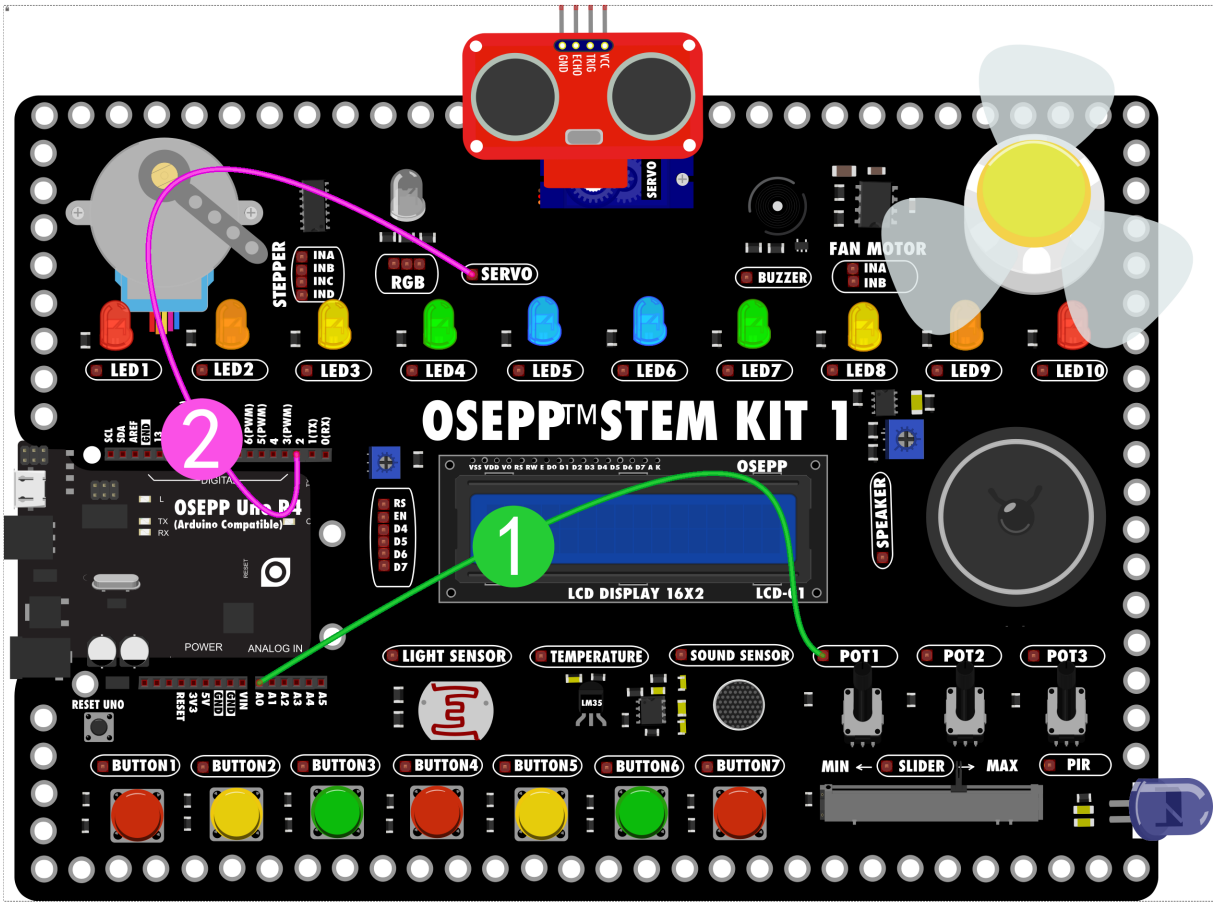
电位器控制舵机

本实验使用电位器控制舵机的角度，为此需要同时连接舵机和电位器。

部署

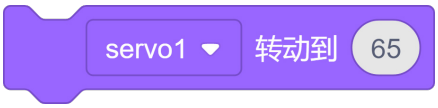
1. 电位器连接到 OSEPP UNO 的 A0 引脚。
2. 舵机连接到 OSEPP UNO 的 3 号引脚。

连线图

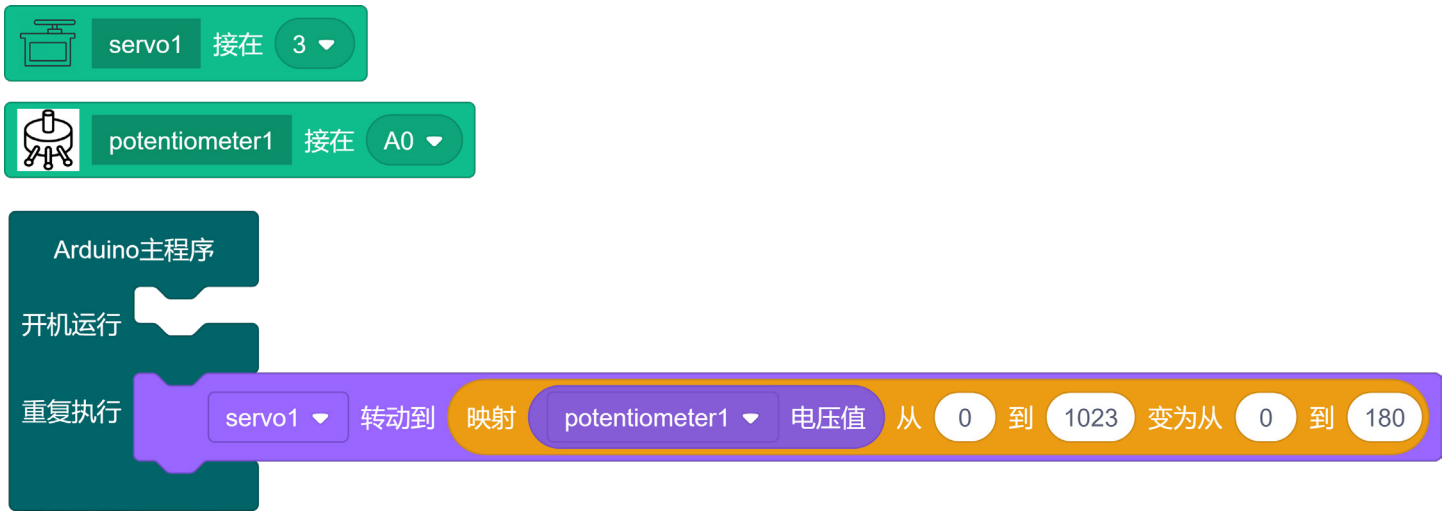


程序搭建

oseppBlock 积木知识

	舵机模块积木。定义名称和引脚
	舵机执行指令模块，数值只能在 0-180 之间。

oseppBlock 代码



Arduino 代码

```
#include <Servo.h>

Servo servo1;

void setup()
{
    servo1.attach(3); // 定义舵机引脚
    //potentiometer1
    pinMode(A0, INPUT); // 定义电位器引脚
}

void loop()
{
    servo1.write(map(analogRead(A0), 0, 1023, 0, 180)); // 把电位器的值映射到舵机输出
}
```

运行结果

当电位器转动时，舵机也会随之转动。电位器旋转到左边最小时舵机处于 0 度，电位器转到右边最大值时舵机处于 180 度位置。

解析

舵机只能在 0-180 度工作，所以要把电位器的输入范围 0-1023 映射成 0-180。

超声波模块

HC-SR04 超声波测距模块可提供 **2cm-400cm** 的非接触式距离感测功能，测距精度可达到 **3mm**，模块包括超声波发射器、接收器与控制电路。像智能小车的测距以及转向，或是一些项目中，常常会用到。智能小车测距可以及时发现前方的障碍物，使智能小车可以及时转向，避开障碍物。

超声波测距的原理是利用超声波在空气中的传播速度为已知，测量声波在发射后遇到障碍物反射回来的时间，根据发射和接收的时间差计算出发射点到障碍物的实际距离。

首先，超声波发射器向某一方向发射超声波，在发射时刻的同时开始计时，超声波在空气中传播，途中碰到障碍物就立即返回来，超声波接收器收到反射波就立即停止计时。超声波在空气中的传播速度为 **C=340m/s**，根据计时器记录的时间 **T**，就可以计算出发射点距障碍物的距离 **L**，即：

$$L = C \times \frac{T}{2}$$

这就是所谓的时间差测距法。

人们可以听到的声音的频率为 **20Hz~20KHz**，也就是可听声波。超出此频率范围的声音，**20Hz** 以下的声音称为低频声波，**20KHz** 以上的声音称为超声波，一般说话的频率范围是 **100Hz-8KHz**。超声波方向性好，穿透能力强，易于获得较集中的声能，在水中传播距离远，超声波因其频率下限大约等于人的听觉上限而得名。

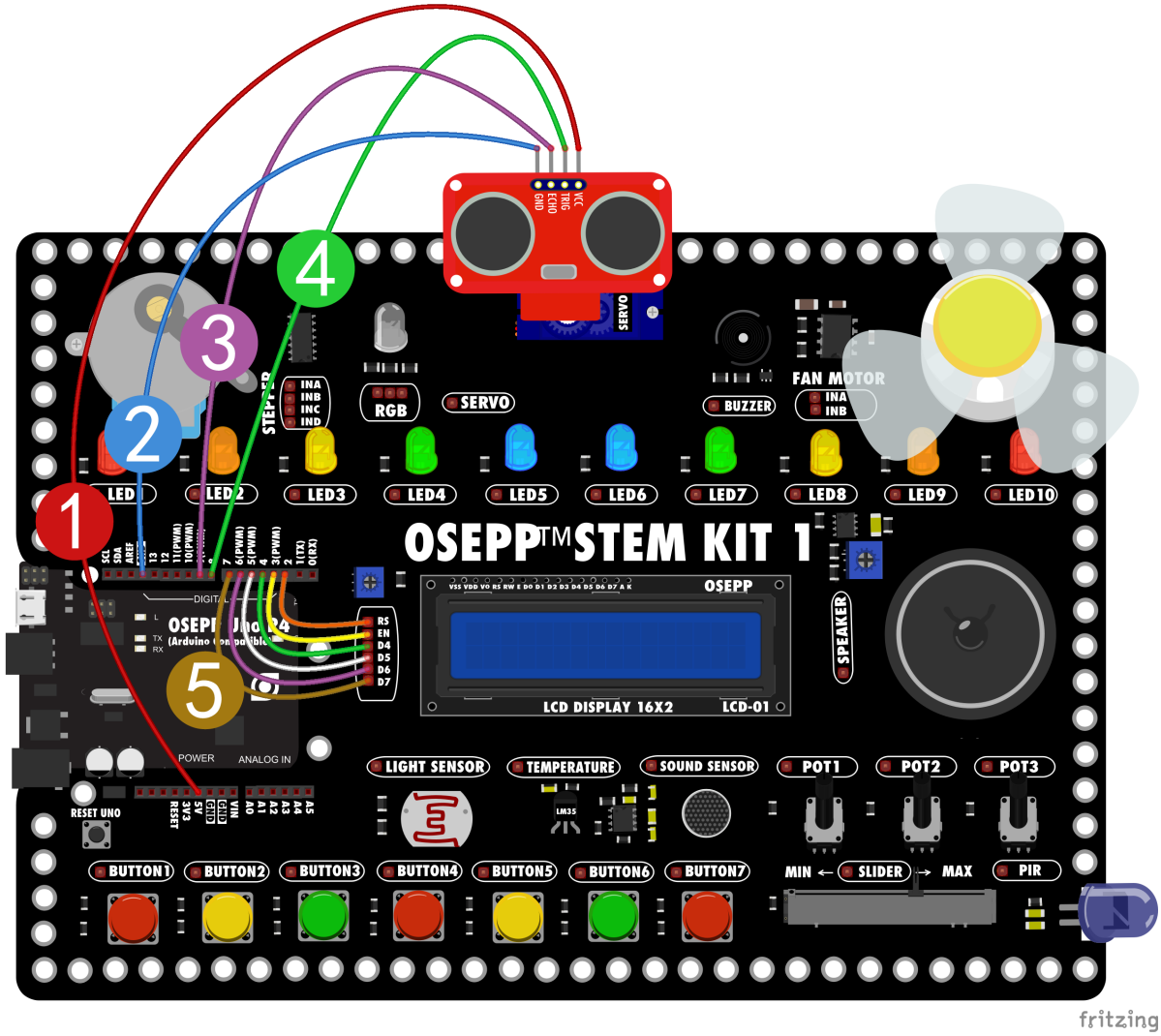
超声测距应用

部署

使用 **Arduino** 控制超声波传感器测量前方障碍物的距离 , 并显示到 LCD 显示屏上。为此，你需要同时连接 LCD 和超声波传感器到 **osepp UNO** 上：

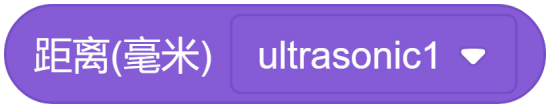
1. 超声波模块 **VCC** 接 **OSEPP UNO** 的 **5V**。
2. 超声波模块 **GND** 接 **OSEPP UNO** 的 **GND**。
3. 超声波模块 **ECHO** 接到 **OSEPP UNO** 的 **9** 号引脚。
4. 超声波模块 **TRIG** 接到 **OSEPP UNO** 的 **8** 号引脚。
5. LCD 的接口 **RS-D7** 分别连接到 **OSEPP UNO** 的 **2~7** 号引脚。

连线图

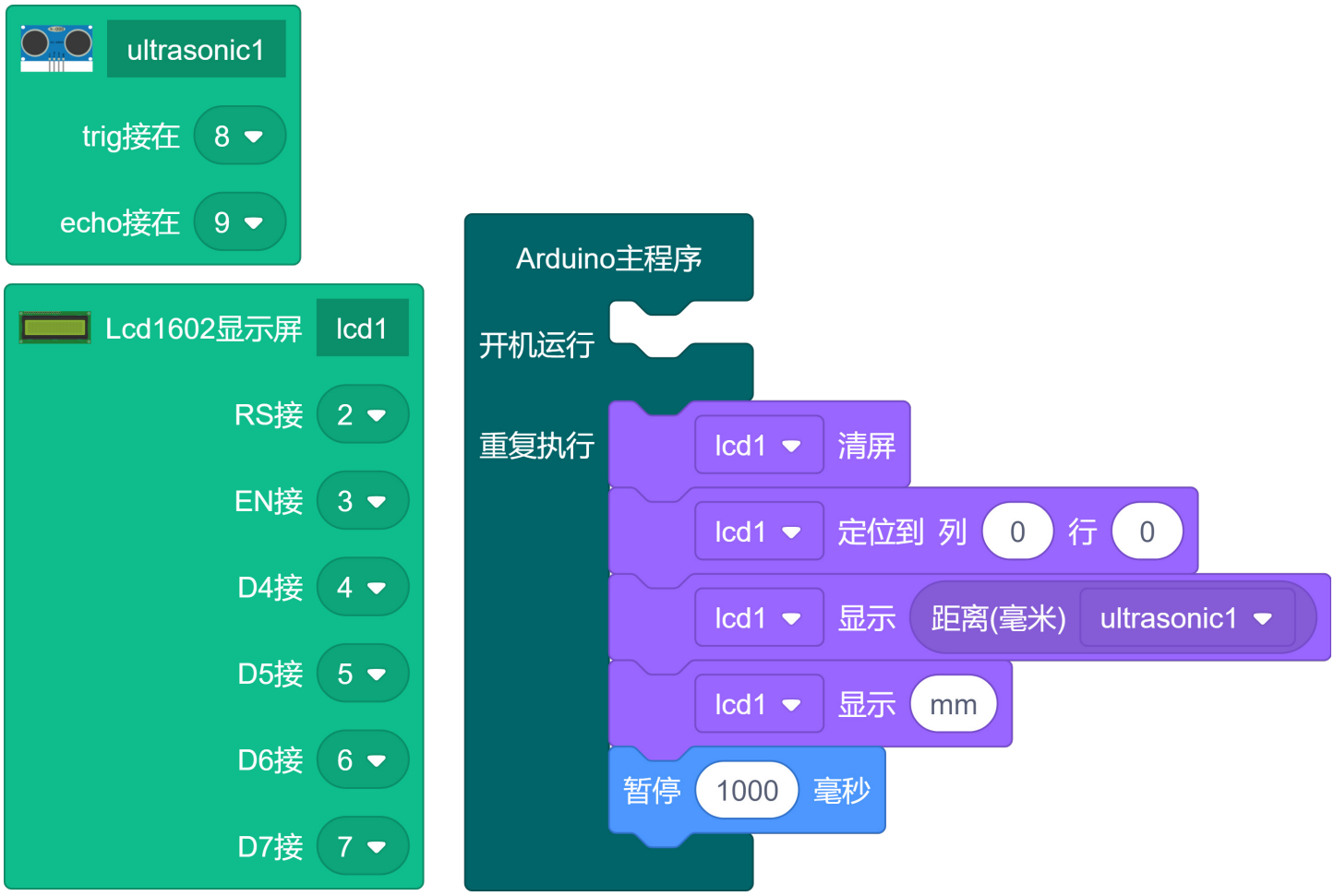


程序搭建

oseppBlock 积木知识

	超声波模块积木。 Trig（控制端），控制发出的超声波信号。 Echo（接收端）接收反射回来的超声波信号。
	超声波模块返回值，单位是毫米 mm。

oseppBlock 代码



Arduino 代码

```
#include <oseppRobot.h>
#include <LiquidCrystal.h>

OseppUltrasonic ultrasonic1(8, 9); // 定义超声波引脚
LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚

void setup()
{
    lcd1.begin(16, 2); //LCD 初始化
}

void loop()
{
    lcd1.clear();                    // 清屏
    lcd1.setCursor(0, 0);            // 显示光标定位
    lcd1.print(ultrasonic1.ping()); // 显示距离
    lcd1.print("mm");                // 单位
    delay(1000);                     // 延时
}
```

运行结果

代码上传成功后，如果超声波模块前面有物体，并且距离在 2-4000mm 之间。LCD 就会显示这个物体与超声波模块之间的距离。你可以改变物体与超声波模块的距离查看 LCD 显示是否产生变化。

解析

采用 IO 口 TRIG 触发测距，给至少 10us 的高电平信号。

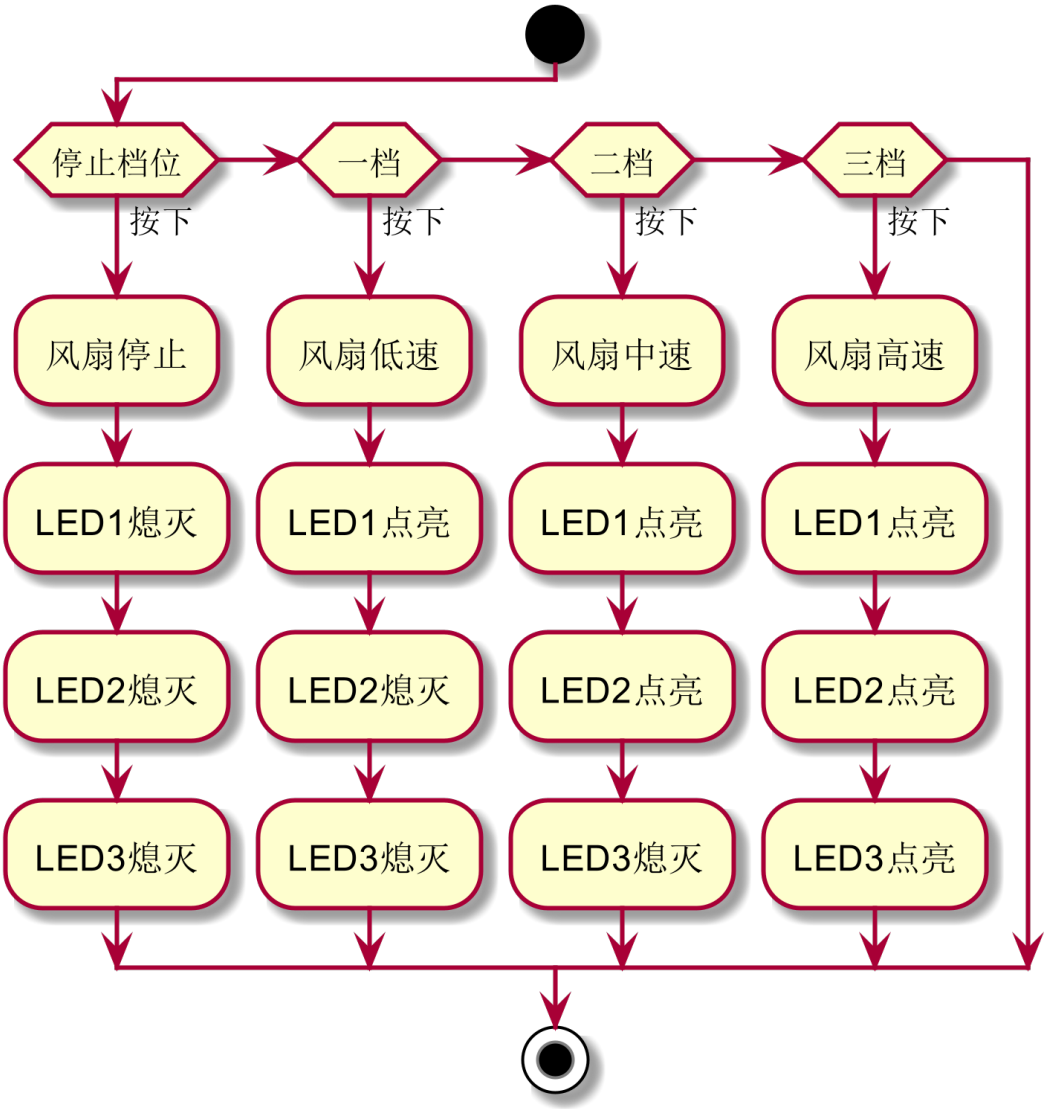
模块自动发送 8 个 40khz 的方波，自动检测是否有信号返回。
有信号返回，通过 IO 口 ECHO 输出一个高电平，高电平持续的时间就是超声波从发射到返回的时间。

$$\text{测试距离} = (\text{高电平时间} * \text{声速}(340\text{M}/\text{S}))/2$$

综合应用 1

档位控制小风扇

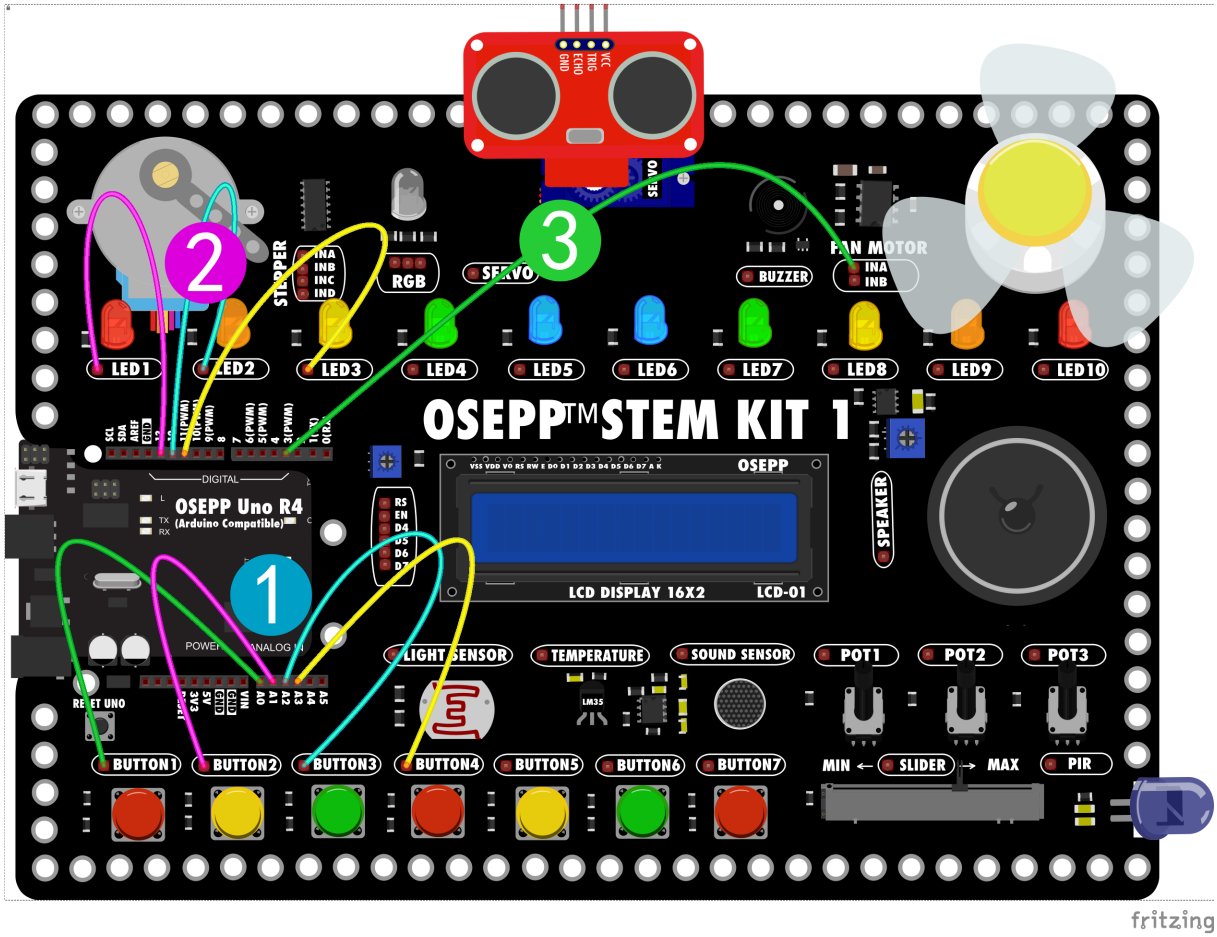
三个档位控制的小风扇。一个停止档位，剩下三个分别为低速，中速和高速档位。



部署

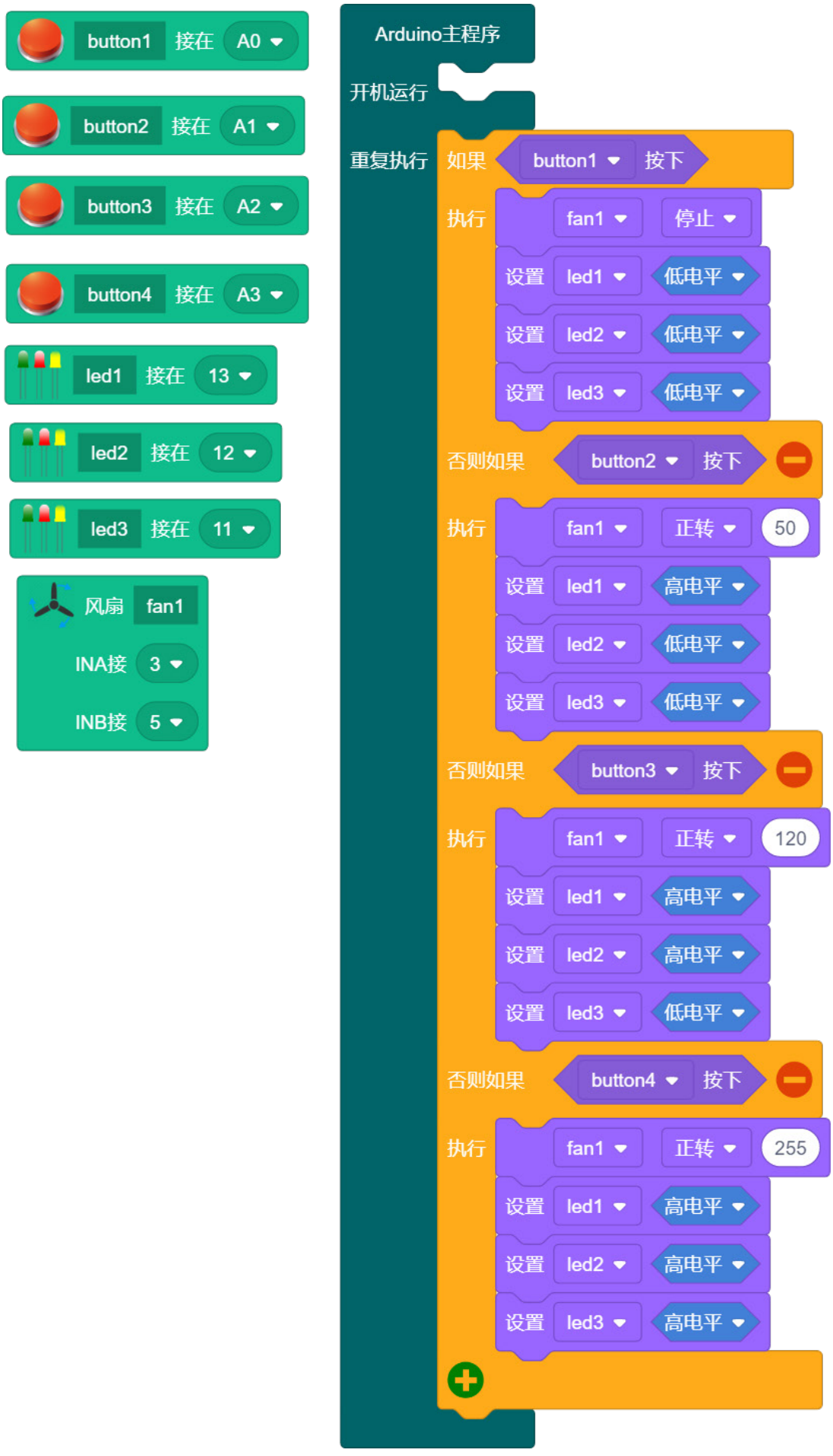
1. 开关 Button1-Button4 分别连接到 OSEPP UNO 的 A0-A4 引脚。
2.LED1-LED3 分别连接到 OSEPP UNO 的 13-11 号引脚。
3. 马达小风扇 INA 连接到 OSEPP UNO 的 3 号引脚。

连线图



程序搭建

oseppBlock 程序



Arduino 程序

```
void setup()
{
    //button1
    pinMode(A0, INPUT); // 定义开关 1
    //button2
    pinMode(A1, INPUT); // 定义开关 2
    //button3
    pinMode(A2, INPUT); // 定义开关 3
    //button4
    pinMode(A3, INPUT); // 定义开关 4
    //led1
    pinMode(13, OUTPUT); // 定义 LED1
    //led2
    pinMode(12, OUTPUT); // 定义 LED2
    //led3
    pinMode(11, OUTPUT); // 定义 LED3
    //fan1
    pinMode(3, OUTPUT); // 定义风扇引脚
    pinMode(5, OUTPUT);
}

void loop()
{
    if (digitalRead(A0) == LOW) // 如果开关 1 按下
    {
        analogWrite(3, 0); // 风扇停止运行
        analogWrite(5, 0);
        digitalWrite(13, LOW); //LED1 熄灭
        digitalWrite(12, LOW); //LED2 熄灭
        digitalWrite(11, LOW); //LED3 熄灭
    }
    else if (digitalRead(A1) == LOW) // 如果开关 2 按下
    {
        analogWrite(3, 50); // 风扇低速运行
        analogWrite(5, 0);
        digitalWrite(13, HIGH); //LED1 点亮
        digitalWrite(12, LOW);  //LED2 熄灭
        digitalWrite(11, LOW);  //LED3 熄灭
    }
    else if (digitalRead(A2) == LOW) // 如果开关 3 按下
    {
        analogWrite(3, 120); // 风扇中速运行
        analogWrite(5, 0);
        digitalWrite(13, HIGH); //LED1 点亮
        digitalWrite(12, HIGH); //LED1 点亮
        digitalWrite(11, LOW);  //LED3 熄灭
    }
```

```
    else if (digitalRead(A3) == LOW) // 如果开关 4 按下
    {
        analogWrite(3, 255); // 风扇高速运行
        analogWrite(5, 0);
        digitalWrite(13, HIGH); //LED1 点亮
        digitalWrite(12, HIGH); //LED1 点亮
        digitalWrite(11, HIGH); //LED1 点亮
    }
}
```

运行结果

Button2 按下风扇低速同时点亮 1 个 LED，Button3 按下风扇中速同时点亮 2 个 LED，Button4 按下风扇高速同时点亮 3 个 LED，Button1 按下风扇停止同时 LED 熄灭。

综合应用 2

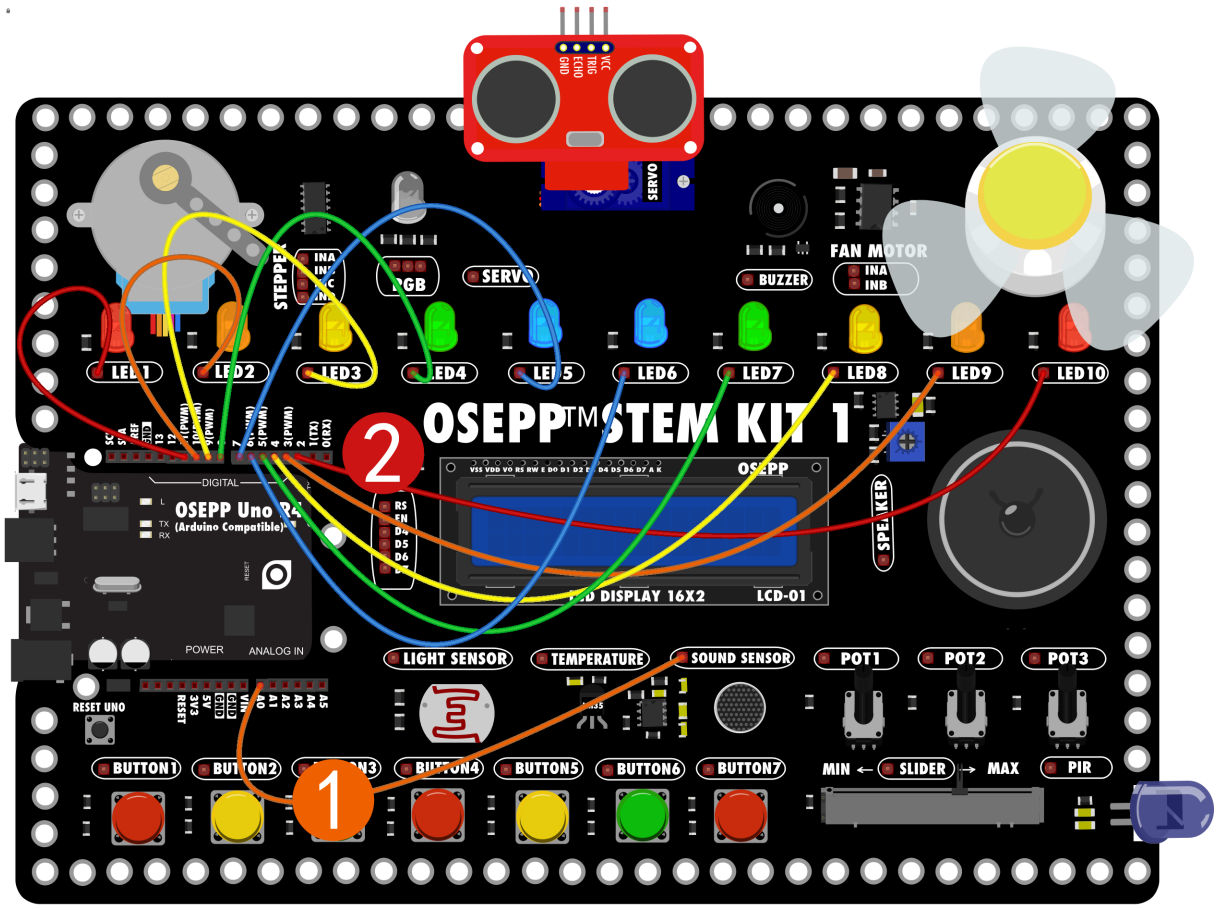
音乐律动灯

当音乐声响起，学习板上的 LED 也会跟着音乐闪烁。是不是很有趣呢！
我们可以利用麦克风采集声音信号，然后经过程序处理。
让学习板上的 LED 随着音乐节奏嗨起来。

部署

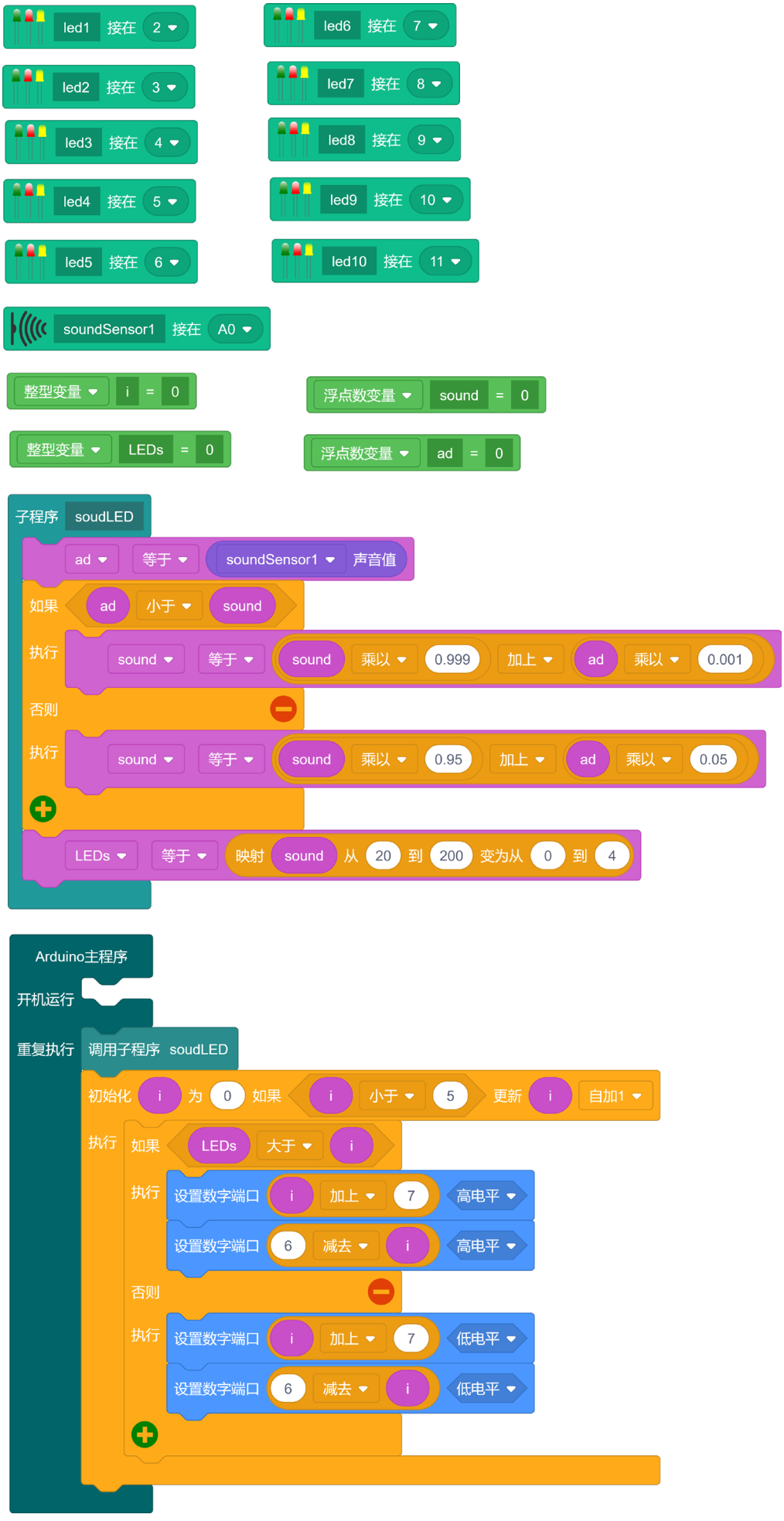
1. 麦克风 Sound Sensor 接到 A0 引脚。
2.LED 连接 , 左边 LED1 对应 11 号引脚，最右边 LED10 对应接到 2 号引脚。

序号	LED 端子	osepp UNO 引脚号
1	LED1	11
2	LED2	10
3	LED3	9
4	LED4	8
5	LED5	7
6	LED6	6
7	LED7	5
8	LED8	4
9	LED9	3
10	LED10	2



程序搭建

oseppBlock 程序



Arduino 程序

```
int i = 0;        //LED 引脚变量
float sound = 0; // 计算映射值变量
int LEDs = 0;     // 映射值变量
float ad = 0;     // 麦克风电压值

void soudLED() // 计算映射值子程序
{
  ad = analogRead(A0); // 读取麦克风声音值
  if (ad < sound)       // 如果声音值小于上次
  {
    sound = sound * 0.999 + ad * 0.001;
    // 一阶低通滤波器算法，也就是声音变小时，LED 回落慢一些。
  }
  else
  {
    sound = sound * 0.95 + ad * 0.05;
    // 一阶低通滤波器算法，减慢 LED 响应速度。但是比上面声音变小时要快一些。
  }
  LEDs = map(sound, 20, 200, 0, 4); // 映射计算出来的值，映射值
}

void setup()
{
  //led1
  pinMode(11, OUTPUT); // 定义 LED 引脚模式
  //led2
  pinMode(10, OUTPUT);
  //led3
  pinMode(9, OUTPUT);
  //led4
  pinMode(8, OUTPUT);
  //led5
  pinMode(7, OUTPUT);
  //led6
  pinMode(6, OUTPUT);
  //led7
  pinMode(5, OUTPUT);
  //led8
  pinMode(4, OUTPUT);
  //led9
  pinMode(3, OUTPUT);
  //led10
  pinMode(2, OUTPUT);
  //soundSensor1
  pinMode(A0, INPUT); // 定义麦克风引脚模式
}
```

```
void loop()
{
  soudLED();                // 计算映射值子程序，利用麦克风的值计算出来 0-4 之间的数
  for (i = 0; i < 5; i++) // 循环 i
  {
    if (LEDs > i) // 如果麦克风映射值大于 i
    {
      digitalWrite(i + 7, HIGH); // 点亮 7-11 号引脚的 LED
      digitalWrite(6 - i, HIGH); // 点亮 6-2 号引脚的 LED
    }
    else // 否则
    {
      digitalWrite(i + 7, LOW); // 熄灭 7-11 号引脚的 LED
      digitalWrite(6 - i, LOW); // 熄灭 6-2 号引脚的 LED
    }
  }
}
```

运行结果

上传好代码后，用手机在麦克风旁边播放音乐，学习板上面的 LED 就会随着音乐节奏闪烁起来。

解析

可以尝试更改代码里面的，20,200 这个值。20 对应的是起点的 LED，可以调节触发的灵敏度。200 对应的是整个范围的灵敏度，声音大时就把值改大点。或者声音小时 LED 点亮数量少，就把这个值改小一点。
一边 5 个 LED，所以后面取值 0~4，为 5 个数。

对应代码：

```
LEDs = map(sound, 20, 200, 0, 4);
```

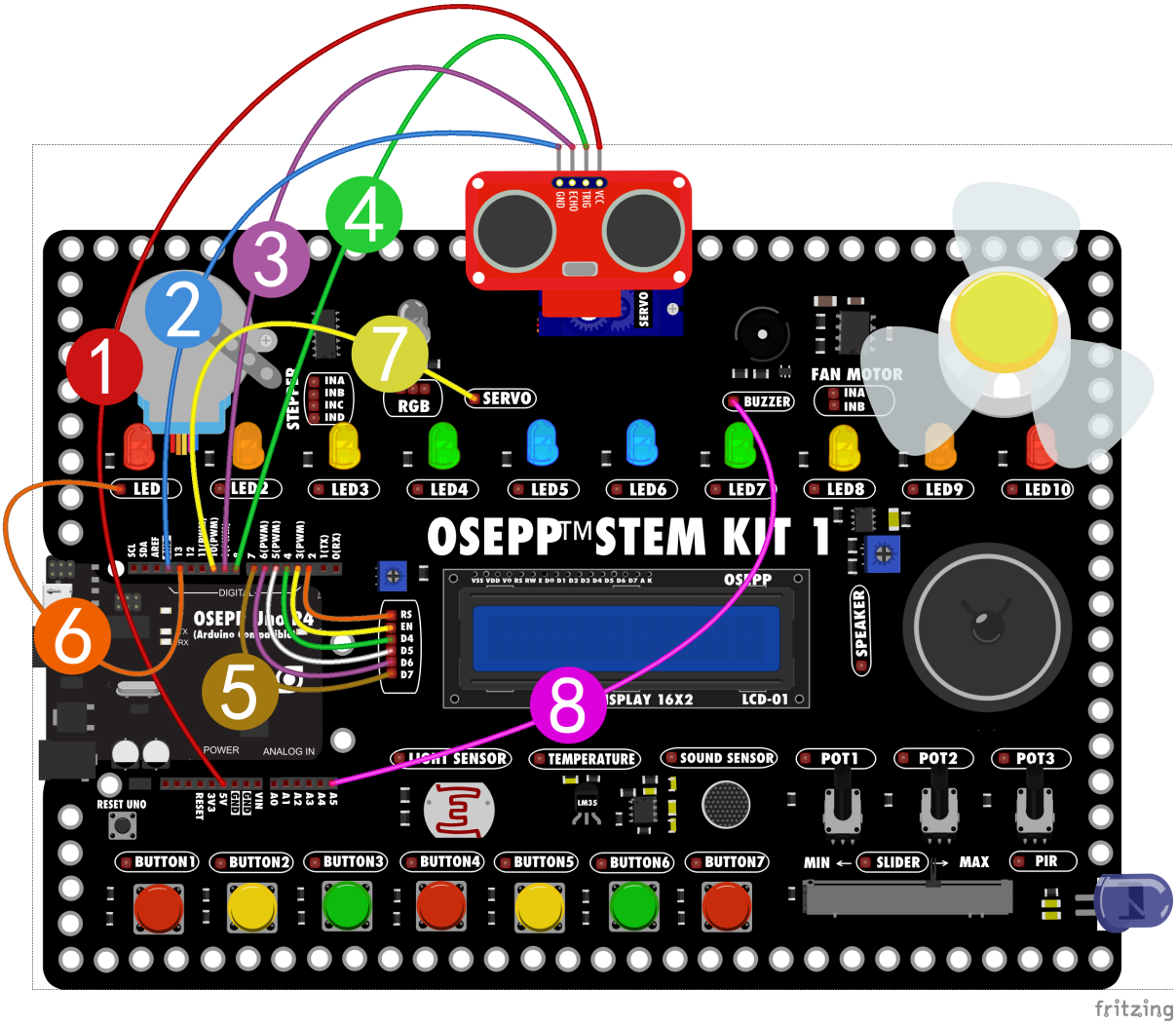
综合应用 3

超声波巡航

雷达车上的天线转来转去的扫描，是不是感觉很高科技。我们也可以制作一个，只是这里用的是超声波来扫描。利用舵机来做巡航动作，超声波来扫描物体。当超声波扫描到障碍物时，就会停下来，并且显示障碍物的距离。

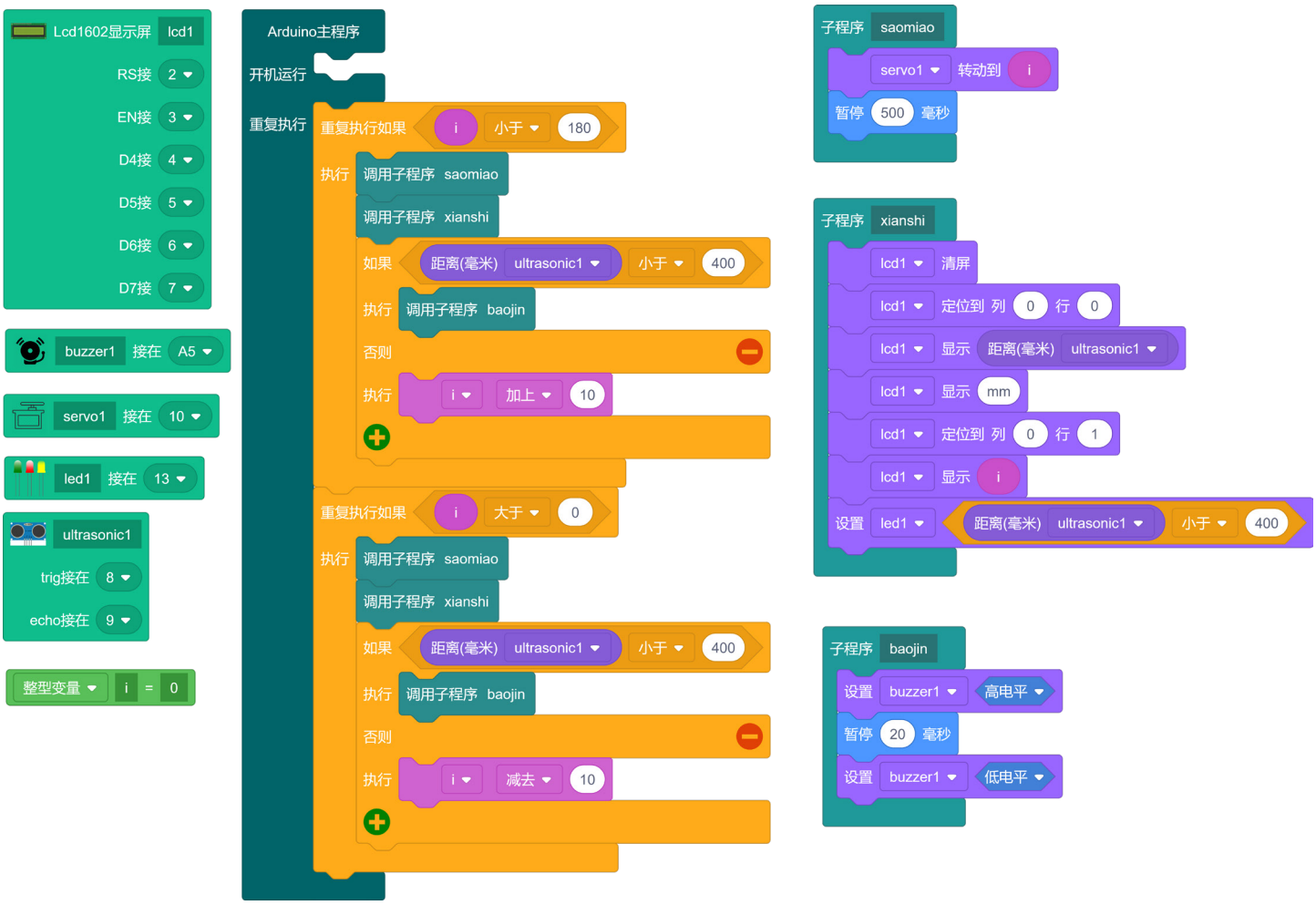
部署

1. 超声波模块 VCC 接 OSEPP UNO 的 5V。
2. 超声波模块 GND 接 OSEPP UNO 的 GND。
3. 超声波模块 ECHO 接到 OSEPP UNO 的 9 号引脚。
4. 超声波模块 TRIG 接到 OSEPP UNO 的 8 号引脚。
5. LCD 的接口 RS-D7 分别连接到 OSEPP UNO 的 2~7 号引脚。
6. LED1 连接到 OSEPP UNO 的 13 号引脚。
7. 舵机连接到 OSEPP UNO 的 10 号引脚。
8. 蜂鸣器连接到 OSEPP UNO 的 A5 引脚。



程序搭建

oseppBlock 程序



Arduino 程序

```
#include <LiquidCrystal.h>
#include <Servo.h>
#include <oseppRobot.h>

LiquidCrystal lcd1(2, 3, 4, 5, 6, 7); // 定义 LCD 引脚
Servo servo1;
OseppUltrasonic ultrasonic1(8, 9); // 定义超声波模块引脚
int i = 0;                          // 定义变量 i

void saomiao() // 子程序，巡航扫描
{
    servo1.write(i); // 舵机转到变量 i
    delay(500);      // 延时 500 毫秒
}

void xianshi() // 显示子程序
{
    lcd1.clear();                                  //LCD 清屏
    lcd1.setCursor(0, 0);                          //LCD 光标定位
    lcd1.print(ultrasonic1.ping());                //LCD 显示当前距离
    lcd1.print("mm");                              // 距离单位
    lcd1.setCursor(0, 1);                          //LCD 光标定位
    lcd1.print(i);                                 //LCD 显示当前舵机角度
    digitalWrite(13, ultrasonic1.ping() < 400); // 距离小于 400，LED 点亮
}

void baojin() // 报警声音子程序
{
    digitalWrite(A5, HIGH); // 蜂鸣器输出声音
    delay(20);              // 延时 20 毫秒
    digitalWrite(A5, LOW);  // 蜂鸣器停止输出
}

void setup()
{
    lcd1.begin(16, 2); //LCD 初始化
    //buzzer1
    pinMode(A5, OUTPUT);
    servo1.attach(10); // 定义蜂鸣器引脚
    //led1
    pinMode(13, OUTPUT); // 定义 LED1 引脚
}
```

```
void loop()
{
    while (i < 180) // 如果变量 i 小于 180
    {
        saomiao();                      // 执行扫描子程序
        xianshi();                      // 执行显示子程序
        if (ultrasonic1.ping() < 400) // 如果距离小于 400
        {
            baojin(); // 执行报警程序
        }
        else
        {
            i += 10; // 否则变量 i 加 10
        }
    }
    while (i > 0) // 如果变量 i 大于 0
    {
        saomiao();                      // 执行扫描子程序
        xianshi();                      // 执行显示子程序
        if (ultrasonic1.ping() < 400) // 如果距离小于 400
        {
            baojin(); // 执行报警程序
        }
        else
        {
            i -= 10; // 否则变量 i 减去 10
        }
    }
}
```

运行结果

程序启动后，安装在舵机上面的超声波模块会在 0-180 度之间巡航扫描。如果在 0-180 度这个范围内有障碍物，并且障碍物的距离小于 400mm 时，舵机会停下，LED 发光同时蜂鸣器发出报警声。当障碍物被移除后，LED 熄灭报警声解除，继续巡航扫描。