第 2-1 讲：点灯实验

1. 学习目的

1. 掌握 LED 驱动电路的设计：控制方式、限流电阻的计算和确定。
2. 掌握 STC8A8K64D4 单片机 GPIO 口四种工作模式。
3. 编写驱动 LED 指示灯点亮和熄灭的程序，再此基础上，更进一步，编写流水灯的程序。

2. 硬件电路设计

LED(Light Emitting Diode)是发光二极管的简称，在很多设备上常用他来作为一种简单的人机接口，如网卡、路由器等通过 LED 向用户指示设备的不同工作状态。所以，我们习惯把这种用于指示状态的 LED 称为 LED 指示灯。

IK-64D4 开发板上设计了 4 个 LED 指示灯，我们可以通过编程驱动 LED 指示灯点亮、熄灭、闪烁，从而达到状态指示的目的，LED 指示灯驱动电路如下图所示。

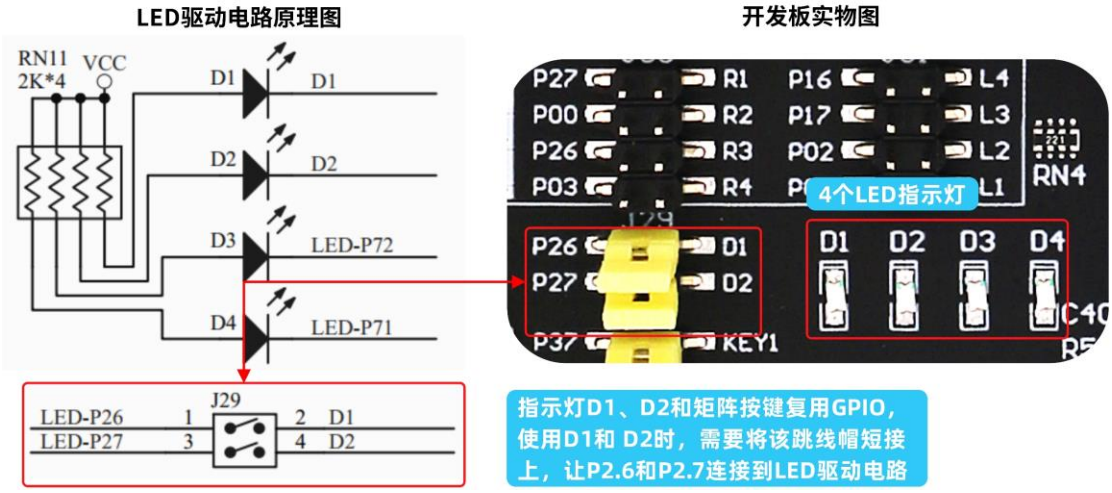


图 1：LED 指示灯驱动电路

4 个 LED 指示灯占用的单片机的引脚如下表：

表 1：LED 引脚分配

LED	颜色	引脚	说明
D1	蓝色	P2.6	非独立 GPIO
D2	蓝色	P2.7	非独立 GPIO
D3	蓝色	P7.2	独立 GPIO
D4	蓝色	P7.1	独立 GPIO

✧ 注：独立 GPIO 表示开发板没有其他的电路使用这个 GPIO，非独立 GPIO 说明开发板有其他电路用到了该 GPIO。读者在使用非独立 GPIO 使用时需要注意电路的连接，以避免多个电路使用了同一个 GPIO。

LED 指示灯驱动电路是一个很常见、简单的电路，但他也是一个典型的单元电路，对于初学者来说，类似常用的典型电路必须要掌握，不但要知其然、还要知其所以然。

接下来，我们来分析一下这个简单的 LED 指示灯驱动电路。

LED 驱动电路设计的时候，需要我们考虑两个方面：控制方式和限流电阻的选取。

2.1. 控制方式

✧ 说明：STC8A8K64D4 支持 3.3V 和 5V 电压，这里，我们以 3.3V 电压为例来说明，使用 5V 电压时原理是一样的。

LED 指示灯控制方式分为高电平有效和低电平有效两种，高电平有效是单片机 IO 输出高电平时点亮 LED，低电平有效是单片机 IO 输出低电平时点亮 LED。

1. 低电平有效的控制方式

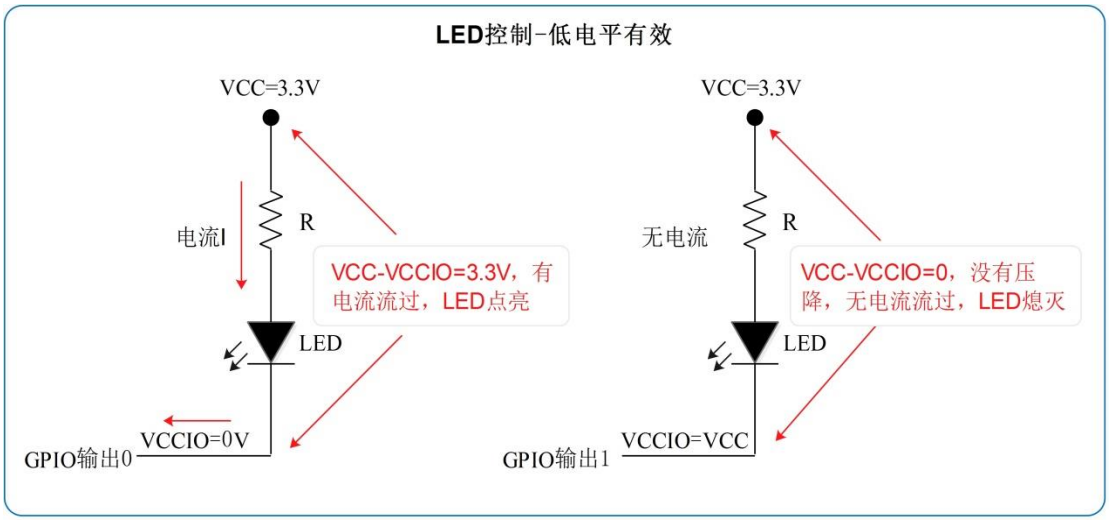


图 2：LED 控制-低电平有效原理

低电平有效控制方式中，当单片机的 GPIO 输出低电平(逻辑 0)的时候，LED 和电阻 R 上的压降等于（VCC-VCCIO = 3.3V），这时候，因为存在压降，同时，这个电路是闭合回环的，这就达到了电流产生的两个要素，LED 上会有电流流过，LED 被点亮。

当单片机的 GPIO 输出高电平(逻辑 1)的时候，LED 和电阻 R 上的压降等于 0V（VCC-VCCIO = 0V），这时候，因为 LED 上没有压降，当然不会有电流流过，所以 LED 熄灭。

低电平有效控制方式中，电流经过限流电阻和 LED “灌入”GPIO，对于一般单片机来说，“灌电流”比较大，即低电平有效时驱动能力较强。

2. 高电平有效的控制方式

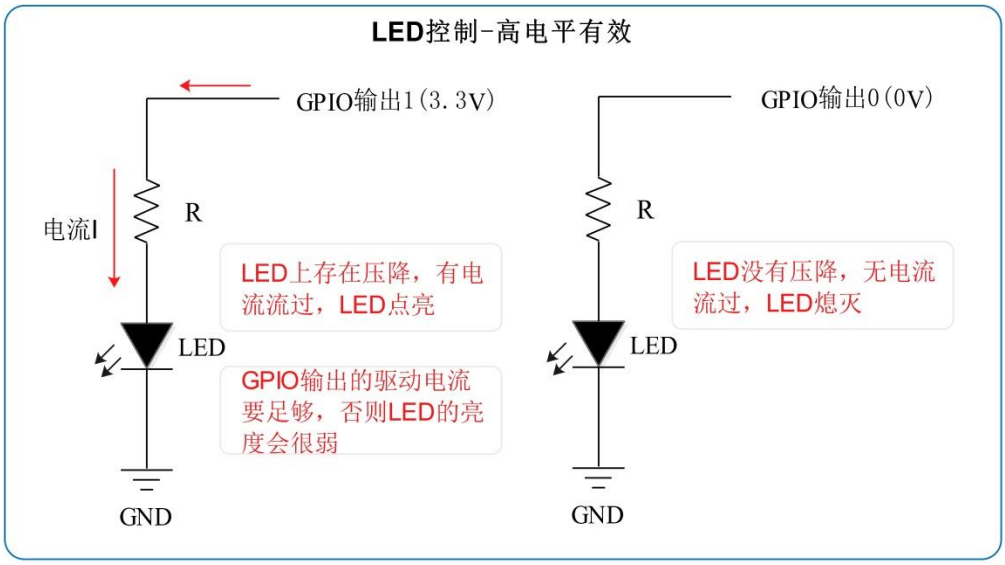


图 3：LED 控制-高电平有效原理

高电平有效控制方式中，由单片机的 GPIO 输出电流驱动 LED，当单片机的 GPIO 输出高电平(逻辑 1)的时候，LED 上存在压降，会有电流流过，这时 LED 被点亮，但要注意，单片机的 GPIO 要能提供足够的输出电流，否则，电流过小，会导致 LED 亮度很弱。

当单片机的 GPIO 输出低电平(逻辑 0)的时候，LED 和电阻 R 上的压降等于 0V，这时候，LED 上没有电流流过，LED 熄灭。

3. 选择哪种方式来控制 LED

大多数情况下，我们会选择使用低电平有效的控制方式，如开发板中的 LED 指示灯就是低电平有效。这是因为：单片机 IO 低电平时的灌入电流一般比高电平时的拉电流要大，能提供足够的电流驱动 LED，另外单片机上电或复位启动时，IO 口一般都是高阻输入，用低电平有效的控制方式可以确保 LED 在上电或复位启动时处于熄灭状态。

2.2. LED 限流电阻的选取

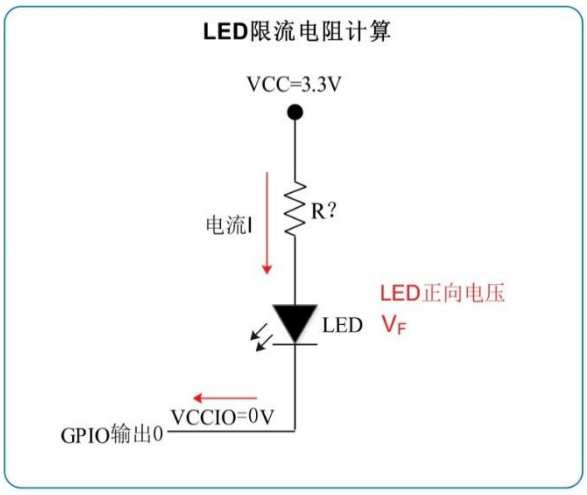
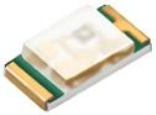


图 4：LED 限流电阻计算

由上图可以看出，LED 限流电阻的计算公式如下：

$$R = \frac{VCC - V_F}{I}\ \Omega$$

其中，VCC=3.3V，V_F 是 LED 的正向压降，LED 的数据手册都会给出正向电流为 20mA 时测试的 V_F 的范围，下图是一款 0603 LED 的实物图和参数。在参数表中可以看到正向电流为 20 mA 时 V_F 最小值是 1.6V，典型值是 2.0V，最大值是 2.6V。



0603 LED

参数名称 Parameter	符号 Symbol	条件 Condition	最小值 Min.	典型值 Typ.	最大值 Max.	单位 Unit
反向电流 Reverse Current	I_R	V_R=5V	-	-	10	μA
视角度 View Angle	$2\theta_{1/2}$	-	-	130	-	deg.
正向电压 Forward Voltage	V_F	电流为20mA时测试的V_F范围	1.6	2.0	2.6	V
峰值波长 Peak Wavelength	λ_P			630		nm
主波长 Dominant Wavelength	λ_d	I_F=20mA	615	622	630	nm
半波宽度 Spectrum Radiation Bandwidth	$\Delta\lambda$		-	15	-	nm
光强 Luminous Intensity	I_V		80	120	220	mcd

计算时 V_F 的值可以用典型值来进行估算，对于电流，需要根据经验值和对 LED 亮度的要求相结合来确定，一般经验值是(2~5)mA，不过要注意，只要亮度符合自己的要求，电流低于 2mA 也没有任何问题。

电流为 2mA 时限流电阻值计算如下：

$$R = \frac{3.3 - 2.0}{0.002}\ \Omega\ = 650\ \Omega$$

计算出的电阻是 650Ω，650Ω 不是一个常用的电阻值，所以我们需要选择一个电阻值为 650Ω 左右常用的电阻器，在 IK-64D4 开发板上，我们选择的电阻值是最常用的 1K 的电阻器，选择限流电阻后，还需要实际测试观察 LED 的亮度是否符合自己的需求，经过实际测试观察，IK-64D4 开发板上使用 1K 限流电阻时亮度符合我们的要求，由此，限流电阻的阻值确定为 1K。当然，如果我们觉得亮度不够，可以将电阻值适当减小一些，如使用 680Ω 或 510Ω 的电阻器作为限流电阻。

3. GPIO 输出驱动原理

3.1. GPIO 数量

STC8A8K64D4 系列单片机 GPIO 口数量取决于芯片的具体型号和封装，开发板使用的是 STC8A8K64D4，封装为 LQFP64，共有 64 个引脚。其中，电源和 ADC 参考电压使用了如下 5 个引脚，剩余的 59 个引脚均可配置为 GPIO 使用。

1) AVref+：ADC 外部参考电压源输入脚
2) ADC_AGnd：ADC 地。
3) ADC_AVcc：ADC 电源引脚
4) VCC：电源引脚。
5) GND：电源地。

STC8A8K64D4 的 59 个 GPIO 分属 8 个端口 P0~P7，如下表所示。这里，有必要说明一下，8 个端口中，P4 和 P5 的 GPIO 数量不是 8 个，其中 P4 是没有 P4.5、P4.6 和 P4.7 的，而 P5 是没有 P5.6 和 P5.7 的，读者使用 GPIO 时，应注意这一点。

表 2：STC8A8K64D4 单片机的 GPIO

端口	GPIO	数量
P0	P0.0~P0.7	8
P1	P1.0~ P1.7	8
P2	P2.0~P2.7	8
P3	P3.0~P3.7	8
P4	P4.0~P4.4	5
P5	P5.0~P5.5	6
P6	P6.0~P6.7	8
P7	P7.0~P7.7	8

3.2. 功能描述

STC8A8K64D4 系列单片机所有 GPIO 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出，这 4 种工作模式是通过寄存器 PnM1 和 PnM0 的组合配置的（n 为端口号）。

本章讲解是 GPIO 输出，和输出相关的工作模式有准双向口/弱上拉、推挽输出/强上拉、和开漏输出，接下来，我们细看一下这 3 种工作模式。

3.2.1. 准双向口/弱上拉

准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态，这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。

准双向口有 3 个上拉晶体管（强上拉、弱上拉和极弱上拉晶体管）适应不同的需要。为了方便描述，我们将这些晶体管分为两部分，如下图所示。

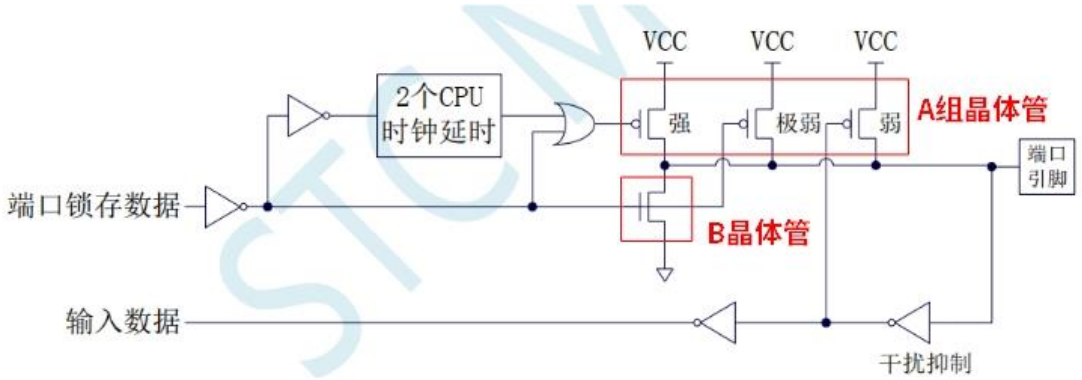


图 5：准双向口 I/O 结构图

1. 输出

- 当端口锁存数据为 1 时：A 组晶体管中的 3 个上拉晶体管中的一个导通，此时 B 晶体管**截止**，由于上拉晶体管的作用，端口引脚输出 1。
- 当端口锁存数据为 0 时：A 组晶体管中的 3 个上拉晶体管中的一个导通，但此时 B 晶体管**导通**，因此，端口引脚输出 0。

2. 输入

准双向口（弱上拉）在读外部状态前，要先锁存为“1”，才可读到外部正确的状态。也就是作为输入时，要先向端口锁存器写入“1”，此时，A 组晶体管中的 3 个上拉晶体管中的一个导通，B 晶体管截止。

- 当端口引脚为 1 时：A 组晶体管中的弱上拉晶体管导通，提供基本驱动电流使得引脚状态为 1，此时读取的输入数据为 1。
- 当端口引脚为 0 时：A 组晶体管中的弱上拉晶体管截至，极弱上拉晶体管导通，此时端口引脚输入 0 将引脚状态拉低，从而读取的输入数据为 0。

3.2.2. 推挽输出

强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为 1 时可提供持续的强上拉。所以，推挽输出一般用于需要更大驱动电流的情况。

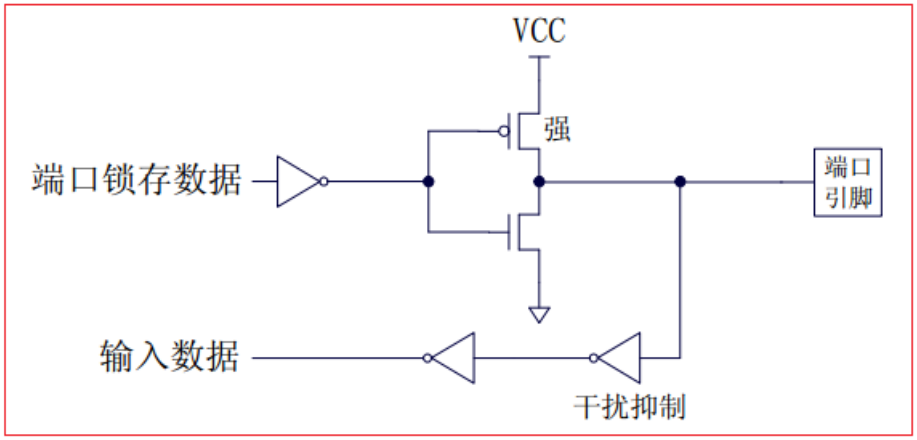


图 6：推挽输出 I/O 结构图

对于 LED 控制电路，如果采用的是高电平有效的控制方式，则控制 LED 的单片机

GPIO 口必须配置成推挽输出才可以驱动 LED。

3.2.3. 高阻输入

高阻输入模式下电流既不能流入也不能流出，输入口带有一个施密特触发输入以及一个干扰抑制电路。STC8A8K64D4 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式。

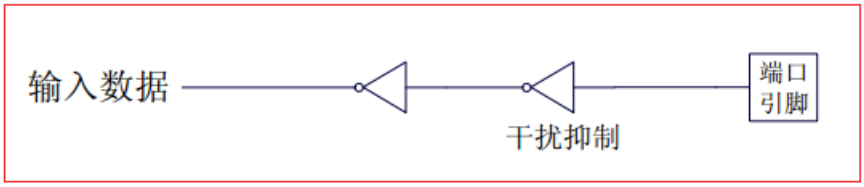


图 7：高阻输入 I/O 结构图

3.2.4. 开漏输出

开漏模式既可读外部状态也可对外输出（高电平或低电平），但是需要注意的是，如要正确读外部状态或需要对外输出高电平，必须外加上拉电阻。这是因为，开漏模式下单片机虽然可以写 0 写 1，但引脚只能输出低电平，无法输出高电平，其原理如下图所示。

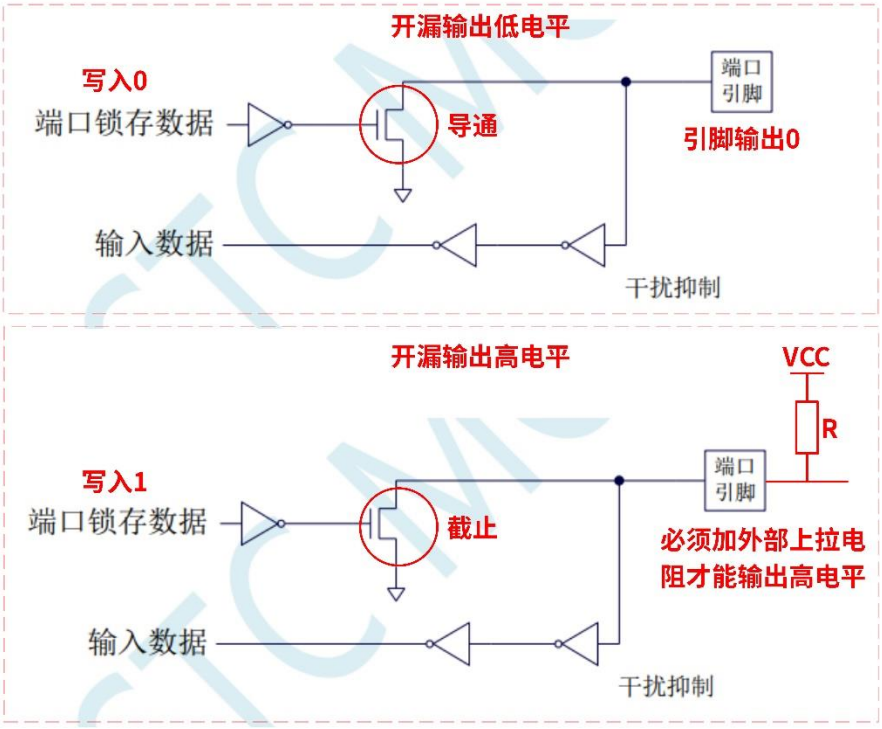


图 8：开漏模式 I/O 结构图

4. 软件设计

4.1. GPIO 应用步骤

GPIO 的使用相对比较简单，使用前先配置 GPIO 的工作模式，之后根据配置的模式操

作即可（输出：写端口数据寄存器，输入：读端口数据寄存器）。

4.1.1. GPIO 配置

STC8A8K64D4 提供了 7 个用于操作 GPIO 的寄存器，如下表所示：

表 3：GPIO 相关寄存器

序号	寄存器名	功能描述
1	端口数据寄存器（Px，x=0~7）	读写端口状态： • 写 0：输出低电平到端口缓冲区。 • 写 1：输出高电平到端口缓冲区。 • 读：直接读端口管脚上的电平。
2	端口模式配置寄存器（PxM0，PxM1，x=0~7）	PxM0，PxM1 的组合用来配置 GPIO 的工作模式（准双向口/弱上拉、推挽输出/强上拉、高阻输入、开漏输出）。
3	端口上拉电阻控制寄存器（PxPU，x=0~7）	端口内部 4.1K 上拉电阻控制位（注：P3.0 和 P3.1 口上的上拉电阻可能会略小一些）。 • 0：禁止端口内部的 4.1K 上拉电阻 • 1：使能端口内部的 4.1K 上拉电阻。
4	端口施密特触发控制寄存器（ PxNCS，x=0~7）	端口施密特触发控制位： • 0：使能端口的施密特触发功能。（上电复位后默认使能施密特触发）。 • 1：禁止端口的施密特触发功能。。
5	端口电平转换速度控制寄存器（PxSR，x=0~7）	用于控制端口电平转换的速度： • 0：电平转换速度快，相应的上下冲会比较大 • 1：电平转换速度慢，相应的上下冲比较小。
6	端口驱动电流控制寄存器（PxDR，x=0~7）	控制端口的驱动能力 • 0：增强驱动能力 • 1：一般驱动能力。
7	端口数字信号输入使能控制寄存器（PxIE，x=0~7）	数字信号输入使能控制： • 0：禁止数字信号输入。若 I/O 被当作比较器输入口、ADC 输入口、触摸按键输入口或者为外部晶振接入脚等模拟口时，进入时钟停振模式前，必须设置为 0，否则会有额外的耗电。 • 1：使能数字信号输入。若 I/O 被当作数字口时，必须设置为 1，否则 MCU 无法读取外部端口的电平。

STC8A8K64D4 的 I/O 模式是通过寄存器 PnM1 和 PnM0 的组合配置的，如下表所示。

表 4：GPIO 模式配置

PnM1	PnM0	I/O 口工作模式
0	0	准双向口（传统 8051 端口模式，弱上拉），灌电流可达 20mA，拉电流为 270～150μA（存在制造误差）。
0	1	推挽输出（强上拉输出，可达 20mA， 要加限流电阻）。
1	0	高阻输入（ 电流既不能流入也不能流出）。
1	1	开漏输出（Open-Drain），内部上拉电阻断开。开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

PnM1 和 PnM0 中的 n 表示的是 I/O 的端口，取值范围为 0~7，即 P0M1 和 P0M0 用来配置端口 P0 中的 I/O 的工作模式，P1M1 和 P1M0 用来配置端口 P1 中的 I/O 的工作模式，以此类推。

两个寄存器中的各个位对应于端口中的 I/O：

- 即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式。
- 即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式，以此类推。

■ 配置示例 1：配置 P0.0 为准双向口（P0M1 的位 0 设置为 0，P0M0 的位 0 设置为 0）

```
P0M1 &= 0xFE;    P0M0 &= 0xFE;
```

■ 配置示例 2：配置 P0.7 为推挽输出（P0M1 的位 7 设置为 0，P0M0 的位 7 设置为 1）

```
P0M1 &= 0x7F;    P0M0 |= 0x80;
```

4.1.2. GPIO 输出/输入

GPIO 配置为输出后，即可通过写端口数据寄存器（Px，x=0~7）中对应的位，让该 GPIO 输出高电平或低电平，示例如下。

```
P0.0 = 1; //P0.0 输出高电平
P0.0 = 0; //P0.0 输出低电平
```

GPIO 配置为输入后，即可通过读端口数据寄存器（Px，x=0~7）中对应的位，从而获取该 GPIO 的状态，示例如下。

```
uint8 temp;  //定义一个变量用于保存读取的 GPIO P0.0 的状态
temp = P0.0;  //读取 GPIO P0.0 的状态
```

4.2. 点灯实验

✧ 注：本节所用的工程即是“第 1-3-1 讲：新建工程模板之新建和配置工程”中新建的

工程，本节对应的实验源码是：“实验 2-1-1：点灯实验”。

4.2.1. 实验内容

1. 配置驱动 LED 指示灯 D3 的 GPIO P7.2 为准双向口。
2. 主循环中软件控制 P7.2 输出高低电平驱动 D3 闪烁：每 200ms 改变一次 P7.2 的输出电平，即 D3 以 200ms 的间隔闪烁。

4.2.2. 代码编写

1. 引用头文件

因为在“main.c”文件中使用了 STC8 的头文件“STC8.h”，所以需要引用下面的头文件。

代码清单：引用头文件

```
1.  //引用头文件
2.  #include    "STC8.H"
```

2. 定义驱动指示灯 D3 的引脚

一般地，在编写程序时，为了使得程序代码清晰，便于程序阅读，通常会使用“#define”命令定义引脚，也就是给这个引脚重新取个便于理解和记忆的名字。

本例中，指示灯 D3 使用了 GPIO P7.2 驱动，因此，我们将 P7.2 重新定义为“LED3_P72”（即指示灯 D3 由 GPIO P7.2 驱动）。

代码清单：定义驱动 LED 的引脚

```
1.  //定义控制指示灯 D3 的 IO,这样定义是为了更直观，当然，我们也可以不定义，直接用 P72
2.  #define LED3_P72  P72
```

编程知识点

使用“#define”命令定义引脚，有以下两个主要的好处：

1) 代码清晰，便于阅读、编写和维护。比如本例中，我们将驱动指示灯 D3 的引脚 P7.2 定义为“LED3_P72”，由这个名字，我们就可以知道：这个引脚是 P7.2，用来驱动指示灯 D3 的。当然，读者也可以根据自己的习惯来定义。
2) 可以做到一改全改，避免遗漏，省心省力。

因此，我们建议读者，尤其是刚入门的读者，要养成这样的编程习惯，这会提高程序开发的效率、降低出错的几率，同时，也能让你编写的程序美观、专业。

3. 配置引脚工作模式

开发板的 LED 驱动电路使用的是低电平驱动方式，因为，我们配置 P0.3 为准双向口即可，代码清单如下：

代码清单：配置引脚工作模式

```
1.  //配置 P7.2 为准双向口
2.  P7M1 &= 0xFB;    P7M0 &= 0xFB;
```

4. 输出驱动 LED 闪烁

驱动 LED 指示灯 D3 闪烁，只需 P7.2 以一定的间隔输出高低电平即可达到驱动 LED 指示灯闪烁的目的。编写代码的时候可以使用：重复“输出高电平→延时→输出低电平→延时”的方式实现，也可以通过：重复“翻转输出状态→延时”的方式实现。

因此，我们先编写延时函数，代码清单如下：

代码清单：软件延时函数

```
/*************************************************************
功能描述：软件延时函数
参    数：x [in]:延时毫秒数
返 回 值：无
*************************************************************/
void delay_ms(uint16 x)
{
    uint16 i,j;
    for(j=0;j<x;j++)
    {
        for(i=0;i<3400;i++);
    }
}
```

之后，在主循环中使驱动 LED 指示灯 D3 的引脚 P7.2 输出高低电平即可实现 D3 闪烁。

代码清单：指示灯 D3 闪烁

```
/*************************************************************************
功能描述：主函数
入口参数：无
返回值：int 类型
 *************************************************************************/
int main(void)
{
   P7M1 &= 0xFB;   P7M0 &= 0xFB;     //配置 P7.2 为准双向口

   while(1)
   {
      LED1_P72=0;       //P7.2 端口输出低电平，点亮用户指示灯 D3
      delay_ms(200);
      LED1_P72=1;       //P7.2 端口输出高电平，熄灭用户指示灯 D3
      delay_ms(200);
      //对于这样的 GPIO 输出状态翻转，也可以用下面的方式写
      //LED3_P72 = ~LED3_P72;       //P7.2 输出电平翻转
      //delay_ms(200);
   }
}
```

4.2.3. 硬件连接

用配套的 USB 数据线按照下图所示将开发板的连接到电脑，本例中使用 P7.2 控制指示灯 D3，因为 D3 是独立 GPIO 控制的，所以没有短接跳线帽的操作。

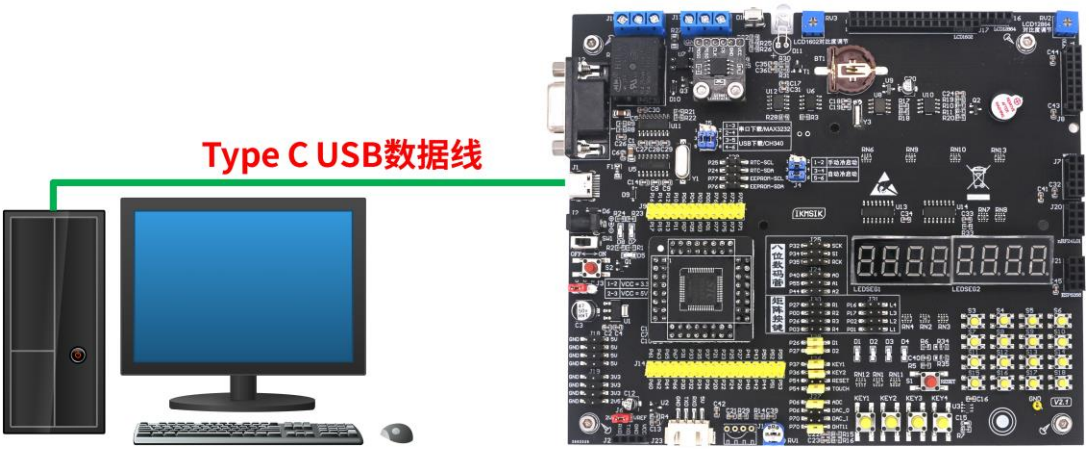


图 9：硬件连接

✧ 说明：在后续的实验中，都需要用 USB 线将开发板连接到电脑，因此，后续实验的硬件连接部分不再描述此部分内容，而只说明跳线帽的短接或传感器的连接。

4.2.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-1-1：点灯实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2. 双击“…\led_blinky\project”目录下的工程文件“led_blinky.uvproj”打开工程。
3. 点击编译按钮编译工程，编译成功后生成的 HEX 文件“led_blinky.hex”位于工程的“…\led_blinky\project\Objects”目录下。
4. 打开 STC-ISP 软件下载程序，下载时勾选使用内部 IRC 时钟，IRC 频率选择：24MHz。
5. 程序运行后，可以观察到蓝色 LED 灯 D3 以 200ms 的间隔闪烁。

4.3. 流水灯实验

✧ 注：本节的实验是在“实验 2-1-1：点灯实验”的基础上修改，本节对应的实验源码是：“实验 2-1-2：流水灯”。

4.3.1. 实验内容

1. 配置驱动 LED 指示灯 D1、D2、D3 和 D4 的 GPIO P2.6、P2.7、P7.2 和 P7.1 为准双向口。
2. 4 个 LED 指示灯以 100ms 的时间间隔循环轮流点亮和熄灭，达到流水灯效果。

4.3.2. 代码编写

1. 定义驱动 4 个指示灯的引脚

指示灯 D1、D2、D3、D4 分别由 GPIO P2.6、P2.7、P7.2 和 P7.1 驱动，这里，我们使用“#define”命令定义如下。

代码清单：定义驱动 LED 的引脚

```
1.  //定义控制指示灯 D1、D2、D3、D4 的 IO
2.  #define LED1_P26     P26
3.  #define LED2_P27     P27
4.  #define LED3_P72     P72
5.  #define LED4_P71     P71
```

2. 实现流水灯

流水灯就是按照一定的时间间隔有规律的轮流点亮 LED 指示灯，也就是 4 个 LED 对应的引脚轮流输出高低电平，并加上一段时间的延时，从而实现轮流点亮的效果，代码清单如下。

代码清单：流水灯

```
1.  /*************************************************************************
2.  功能描述：主函数
3.  入口参数：无
4.  返回值：int 类型
5.   *************************************************************************/
6.  int main(void)
7.  {
8.    P2M1 &= 0x3F;   P2M0 &= 0x3F;     //配置 P2.6、P2.7 为准双向口
9.    P7M1 &= 0xF9;   P7M0 &= 0xF9;     //配置 P7.2、P7.1 为准双向口
10.
11.   while(1)
12.   {
13.      LED1_P26=0;     //D1 点亮
14.      delay_ms(100);  //延时 100ms
15.      LED1_P26=1;     //D1 熄灭
16.
17.      LED2_P27=0;     //D2 点亮
18.      delay_ms(100);  //延时 100ms
19.      LED2_P27=1;     //D1 熄灭
20.
21.      LED3_P72=0;     //D3 点亮
22.      delay_ms(100);  //延时 100ms
23.      LED3_P72=1;     //D1 熄灭
24.
25.      LED4_P71=0;     //D4 点亮
26.      delay_ms(100);  //延时 100ms
27.      LED4_P71=1;     //D1 熄灭
28.
29.      delay_ms(100);  //延时 100ms
```

```
30.    }
31. }
```

4.3.3. 硬件连接

本实验需要使用 4 个 LED 指示灯，因此需要用跳线帽短接复用引脚的指示灯（D1 和 D2），而指示灯 D3 和 D4 是独立引脚，没有和其他电路复用引脚，是没有短接跳线帽的操作的。

图 10：跳线帽短接

4.3.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-1-2：流水灯”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2. 双击“…\led_blinky\project”目录下的工程文件“led_blinky.uvproj”。
3. 点击编译按钮编译工程，编译成功后生成的 HEX 文件“led_blinky.hex”位于工程的“…\led_blinky\project\Objects”目录下。
4. 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5. 程序运行后，可以观察到 4 个 LED 指示灯轮流闪烁的流水灯效果。

4.4. 编写 LED 驱动实验

✧ 注：本节的实验是在“实验 2-1-1：点灯实验”的基础上修改，本节对应的实验源码是：“实验 2-1-3：流水灯（自编驱动文件方式）”。

当我们开发的程序比较复杂的时候，通过 GPIO 直接操作 LED 显然比较麻烦，比较好的做法是将需要用到 LED 操作封装为函数，并将这些函数单独放到一个“.c”文件里面，其他文件中需要用到 LED 功能的时候，调用这些函数实现功能即可。

接下来，我们就用这种方式来实现实验 2-1-2 中的流水灯，读者可以体会一下，哪种方式更好。

4.4.1. 实验内容

1. 在“实验 2-1-1：点灯实验”的基础上编写指示灯的驱动文件。

2. 本实验实现的功能和“实验 2-1-2：流水灯”一样。

4.4.2. 规划

本例中，我们需要用到 LED 和延时，因此，我们新建名称为“delay.c”和“led.c”的文件，分别用于存放延时和 LED 操作相关的函数。另外，新建对应的头文件“delay.h”和“led.h”，以便于其他程序模块使用。

“delay.c”包含的函数分别如下表所示。

表 5：延时函数

文件名称	函数名称	功能
delay.c	delay_ms	毫秒延时函数（软件延时）。
	delay10us	10us 延时函数（软件延时）。

“led.c”包含的函数分别如下表所示。

表 6：LED 操作函数

文件名称	函数名称	功能
led.c	led_on	点亮指定的 LED 指示灯。
	led_off	熄灭指定的 LED 指示灯。
	led_toggle	翻转指定的 LED 的状态。
	leds_on	点亮所有的 LED 指示灯。
	leds_off	熄灭所有的 LED 指示灯。

4.4.3. 新建文件和添加头文件路径

1. 新建名称为“delay.c”和“led.c”的文件并保存到工程的“Source”文件夹，同时，为“delay.c”和“led.c”文件分别新建一个头文件“delay.h”和“led.h”也保存到工程的“Source”文件夹。每个文件夹加入到该组每个“.c”文件对应一个“.h”头文件，以方便其他文件引用。
2. Keil 工程“led_blinky”中新建一个名称为“SOURCE”组，并将“delay.c”和“led.c”加入到该组。
3. 添加头文件包含路径，如下图所示。

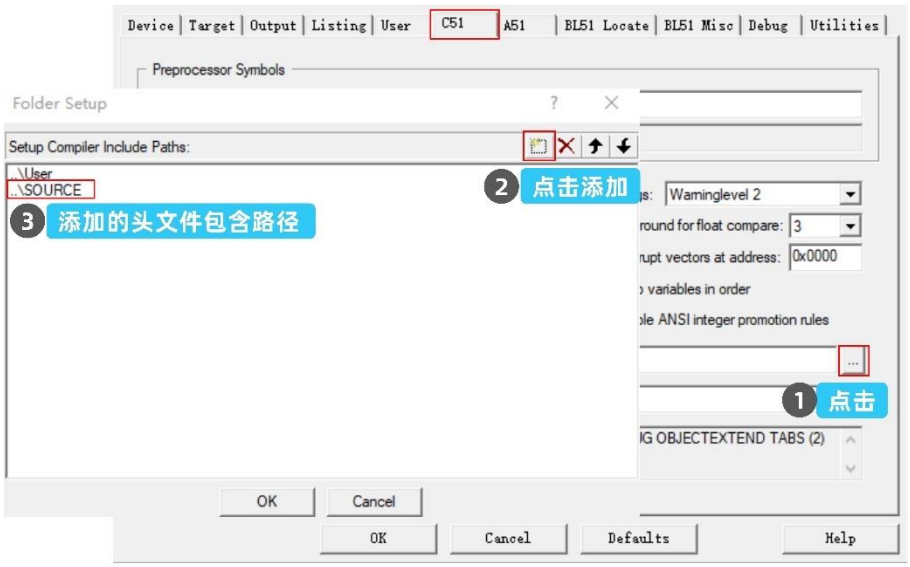


图 11：添加头文件包含路径

4.4.4. 编写 LED 驱动函数

为了软件操作方便，我们给每个指示灯按照原理图上的定义（D1、D2、D3、D4）赋予一个编号，注意这个编号不是物理引脚定义，而是为了软件操作方便，人为在软件层面对指示灯进行的编号。

代码清单：指示灯编号

```
1.  //定义指示灯的编号，注意这个编号是为了软件操作方便，人为对指示灯进行的编号
2.  #define LED_1          1
3.  #define LED_2          2
4.  #define LED_3          3
5.  #define LED_4          4
```

LED 操作的函数代码清单如下：

1. led_on：点亮指定的 LED 指示灯。

代码清单：led_on 函数

```
1.  /*******************************************************************************
2.  功能描述：点亮一个指定的指示灯(D1、D2、D3、D4)
3.  参    数：led_idx [in]: led 指示灯编号，可取值 LED_1、LED_2、LED_3、LED_4
4.  返 回 值：无
5.  *******************************************************************************/
6.  void led_on(u8 led_idx)
7.  {
8.    switch(led_idx)
9.    {
10.     case LED_1:
11.       LED1_P26=0;        //控制 P2.6 端口输出低电平，点亮用户指示灯 D1
12.       break;
```

```
13.       case LED_2:
14.         LED2_P27=0;         //控制 P2.7 端口输出低电平，点亮用户指示灯 D2
15.         break;
16.       case LED_3:
17.         LED3_P72=0;         //控制 7.2 端口输出低电平，点亮用户指示灯 D3
18.         break;
19.       case LED_4:
20.         LED4_P71=0;         //控制 P7.1 端口输出低电平，点亮用户指示灯 D4
21.         break;
22.       default:
23.         break;
24.    }
25. }
```

2. led_off：熄灭指定的 LED 指示灯。

代码清单：led_off 函数

```
1.  /*************************************************************************
2.  功能描述：熄灭一个指定的指示灯(D1、D2、D3、D4)
3.  参     数：led_idx [in]: led 指示灯编号，可取值 LED_1、LED_2、LED_3、LED_4
4.  返 回 值：无
5.   *************************************************************************/
6.  void led_off(u8 led_idx)
7.  {
8.    switch(led_idx)
9.    {
10.      case LED_1:
11.        LED1_P26=1;         //控制 P2.6 端口输出高电平，熄灭用户指示灯 D1
12.        break;
13.      case LED_2:
14.        LED2_P27=1;         //控制 P2.7 端口输出高电平，熄灭用户指示灯 D2
15.        break;
16.      case LED_3:
17.        LED3_P72=1;         //控制 P7.2 端口输出高电平，熄灭用户指示灯 D3
18.        break;
19.      case LED_4:
20.        LED4_P71=1;         //控制 P7.1 端口输出高电平，熄灭用户指示灯 D4
21.        break;
22.      default:
23.        break;
24.   }
25. }
```

3. led_toggle：翻转指定的 LED 的状态。

代码清单：led_ toggle 函数

```
/*************************************************************************
功能描述：翻转一个指定的指示灯的状态(D1、D2、D3、D4)
参    数：led_idx [in]: led 指示灯编号，可取值 LED_1、LED_2、LED_3、LED_4
返 回 值：无
*************************************************************************/
void led_toggle(u8 led_idx)
{
  switch(led_idx)
  {
    case LED_1:
      LED1_P26=~LED1_P26;    //P2.6 输出电平取反，翻转用户指示灯 D1
      break;
    case LED_2:
      LED2_P27=~LED2_P27;    //P2.7 输出电平取反，翻转用户指示灯 D2
      break;
    case LED_3:
      LED3_P72=~LED3_P72;    //7.2 输出电平取反，翻转用户指示灯 D3
      break;
    case LED_4:
      LED4_P71=~LED4_P71;    //P7.1 输出电平取反，翻转用户指示灯 D4
      break;
    default:
      break;
  }
}
```

4. leds_on：点亮开发板上所有的 LED 指示灯（D1、D2、D3 和 D4）。

代码清单：leds_on 函数

```
/*************************************************************************
功能描述：点亮开发板上的 4 个指示灯(D1、D2、D3、D4)
参    数：无
返 回 值：无
 *************************************************************************/
void leds_on(void)
{
   LED1_P26=0;         //控制 P0.3 端口输出低电平，点亮用户指示灯 D1
   LED2_P27=0;         //控制 P0.2 端口输出低电平，点亮用户指示灯 D2
   LED3_P72=0;         //控制 P0.1 端口输出低电平，点亮用户指示灯 D3
   LED4_P71=0;         //控制 P0.0 端口输出低电平，点亮用户指示灯 D4
   //也可以直接操作 GPIO 端口
```

```
    //P2 &= 0x3F;
    //P7 &= 0xF9;
}
```

5. leds_off：熄灭开发板上所有的 LED 指示灯（D1、D2、D3 和 D4）。

代码清单：leds_off 函数

```
/*************************************************************************
功能描述：熄灭开发板上的 4 个指示灯(D1、D2、D3、D4)
参    数：无
返 回 值：无
*************************************************************************/
void leds_off(void)
{
    LED1_P26=1;         //控制 P2.6 端口输出高电平，熄灭用户指示灯 D1
    LED2_P27=1;         //控制 P2.7 端口输出高电平，熄灭用户指示灯 D2
    LED3_P72=1;         //控制 P7.2 端口输出高电平，熄灭用户指示灯 D3
    LED4_P71=1;         //控制 P7.1 端口输出高电平，熄灭用户指示灯 D4

    //也可以直接操作 IO 端口
    //P2 |= 0Xc0;
    //P7 |= 0X06;
}
```

4.4.5. 用 LED 驱动函数实现流水灯

下面是用 LED 驱动函数实现的流水灯的代码清单，他的效果和“实验 2-1-2：流水灯”是一样的。

代码清单：LED 驱动函数实现流水灯

```
/*************************************************************************
功能描述：主函数
入口参数：无
返回值：int 类型
 *************************************************************************/
int main(void)
{
    P2M1 &= 0x3F;   P2M0 &= 0x3F;     //配置 P2.6、P2.7 为准双向口
    P7M1 &= 0XF9;   P7M0 &= 0XF9;     //配置 P7.1、P7.2 为准双向口

    while(1)
    {
        led_on(LED_1);  //点亮 D1
        delay_ms(100);  //延时 100ms
```

```
15.         leds_off();     //熄灭所有用户指示灯
16.         led_on(LED_2);  //点亮 D2
17.         delay_ms(100);  //延时 100ms
18.         leds_off();     //熄灭所有用户指示灯
19.         led_on(LED_3);  //点亮 D3
20.         delay_ms(100);  //延时 100ms
21.         leds_off();     //熄灭所有用户指示灯
22.         led_on(LED_4);  //点亮 D4
23.         delay_ms(100);  //延时 100ms
24.         leds_off();     //熄灭所有用户指示灯
25.         delay_ms(100);  //延时 100ms
26.     }
27. }
```

4.4.6. 实验步骤

1. 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-1-3：流水灯（自编驱动文件方式）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2. 双击“…\led_blinky\project”目录下的工程文件“led_blinky.uvproj”。
3. 点击编译按钮编译工程，编译成功后生成的 HEX 文件“led_blinky.hex”位于工程的“…\led_blinky\project\Objects”目录下。
4. 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5. 程序运行后，可以观察到 4 个 LED 指示灯轮流闪烁的流水灯效果（和实验 2-1-2：流水灯一样）。

✧ 说明：后续的实验涉及到的单片机外设程序编写，都会采用本节的方式编写代码。在后续的章节中将不再详细描述新建文件之类的操作（到这里，读者应该对这些常规操作很熟悉了）。