第5章

闪烁的 LED 实验

【本章学习目标】

（1）动手完成闪烁的 LED 灯实验，熟练掌握 C 语言鸿蒙 OS 设备程序项目的架构和内容。

（2）动手完成呼吸灯实验，熟练掌握 C 语言鸿蒙 OS 设备程序项目的开发步骤和方法。

（3）了解闪烁的 LED 灯和呼吸灯程序的工作原理。

第 3 章的 C 语言鸿蒙 OS 设备开发实验点亮了一个 LED 灯（发光二极管），本章利用循环控制实现一个闪烁的 LED 灯。

5.1 闪烁的 LED 灯

5.1.1 闪烁的 LED 灯程序项目的结构和内容

闪烁的 LED 灯 C 语言鸿蒙 OS 程序项目由 1 个 C 语言源程序文件 FLASHING.c、2 个 BUILD.gn 文件和 1 个 config.json 文件组成。各个文件在计算机磁盘上的存储情况如图 5-1 所示。

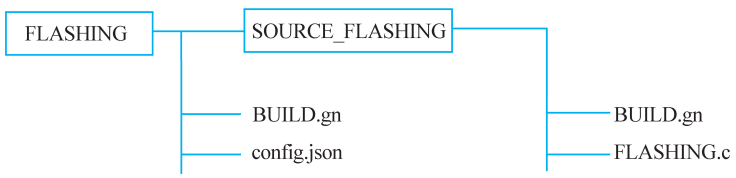

图 5-1 闪烁的 LED 灯 C 语言鸿蒙 OS 程序项目结构和内容示意图

图中外带方框的 FLASHING 和 SOURCE_FLASHING 是文件夹，而且 SOURCE_FLASHING 文件夹是 FLASHING 文件夹下面的子文件夹。在 FLASHING 文件夹下有 BUILD.gn 和 config.json 两个文件，在 SOURCE_FLASHING 子文件夹下也有一个 BUILD.gn 文件和一个 FLASHIN.c 文件。其中，config.json 文件内容和第 3 章的点亮一只 LED 灯项目的 config.json 文件内容基本相同，只有第一行“product_name”后面的项目名称有区别，因为是两个不同的项目。点亮一只 LED 灯项目的项目名称为 LED，闪烁的 LED 灯项目的项目名称为 FLASHING。各文件的内容分别如下所示。

(1) FLASHING 文件夹下的 BUILD.gn 文件内容。

```
group("FLASHING")
{
    deps = [
          "SOURCE_FLASHING:FLASHING",
          "//device/bossay/hi3861_l0/sdk_liteos:wifiiot_sdk",
          "../common/iot_wifi:iot_wifi",
         ]
}
```

(2) FLASHING 文件夹下的 config.json 文件内容。

```
{
     "product_name": "FLASHING",
```

从第 2 行开始,剩余内容跟点亮一只 LED 灯项目的 config.json 文件从第 2 行开始的各行内容相同,在此不再赘述。

```
}
```

(3) SOURCE_FLASHING 文件夹下的 BUILD.gn 文件内容。

```
static_library("FLASHING")
{
    sources = [ "FLASHING.c",  ]
    include_dirs = [
                "//utils/native/lite/include",
                "//base/iot_hardware/peripheral/interfaces/kits",
                "//device/bossay/hi3861_l0/iot_hardware_hals/include",
                "//device/bossay/hi3861_l0/sdk_liteos/include"
               ]
}
```

(4) SOURCE_FLASHING 文件夹下的 FLASHING.c 文件内容。

```
#include <stdio.h>
#include "ohos_init.h"
#include "iot_gpio.h"
#include "iot_gpio_ex.h"

#define LED_GPIO 9
static void led(void* args)
{
  printf("led running...");
  IoTGpioInit(LED_GPIO);
  IoTGpioSetDir(LED_GPIO,IOT_GPIO_DIR_OUT);
  IoTGpioSetFunc(LED_GPIO,IOT_GPIO_FUNC_GPIO_9_GPIO);
  int v = 1;
  while(1)
  {
    IoTGpioSetOutputVal(LED_GPIO,v);
    v = 1-v;
```

```
        usleep(200 * 1000);
    }
}

APP_FEATURE_INIT(led);
```

5.1.2　编辑、编译、烧录、运行闪烁的 LED 灯程序


5.1.2

参照本书第 4 章网页编译的方法，将源程序 FLASHING.c 的代码复制到网页进行编译，生成可执行目标代码；或参照第 3 章的方式，利用 VS Code 的 DevEco 工具建立闪烁的 LED 灯程序项目，编辑程序代码、编译生成可执行目标代码。然后使用 USB-Type 连线连接计算机和开发板，利用 Hiburn 工具烧录可执行目标代码到开发实验板，按下开发板上的 RESET 复位键运行项目程序，程序运行效果如图 5-2 所示。

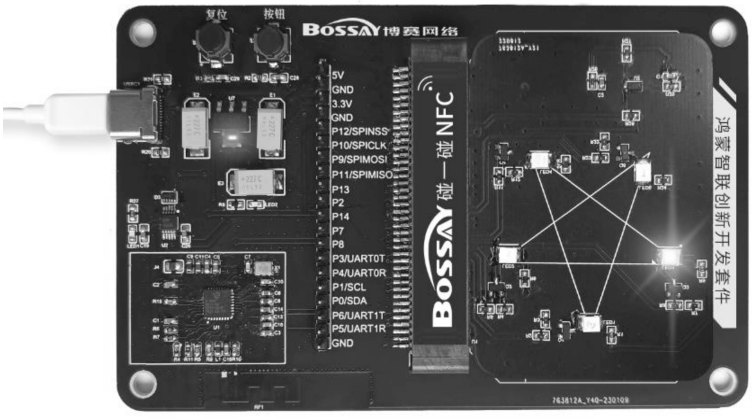

图 5-2　闪烁的 LED 效果图

5.1.3　闪烁的 LED 灯程序的工作原理


5.1.3

图 5-3 是一个 32 引脚的 Hi3861 芯片。大多数引脚有多重功能，例如，本次实验使用的 GPIO9 实际上是 27 号引脚，可以作为通用输入输出端口。

闪烁的 LED 灯程序代码开头使用 include 引用了四个类库，stdio 为标准输入输出库，包含 printf 函数的定义，ohos_init 包含宏 APP_FEATURE_INIT 的定义，iot_gpio 包含控制 Hi3861 芯片的引脚 GPIO 驱动函数的定义，iot_gpio_ex 包含 IOT_GPIO_FUNC_GPIO_9_GPIO 的定义。

APP_FEATURE_INIT 指出程序入口为 led 函数。该函数中使用 printf 打印了提示信息，并对 Hi3861 芯片的第 27 引脚（即 LED_GPIO）的 GPIO 功能进行了初始化。通过 IoTGpioSetDir 函数设定

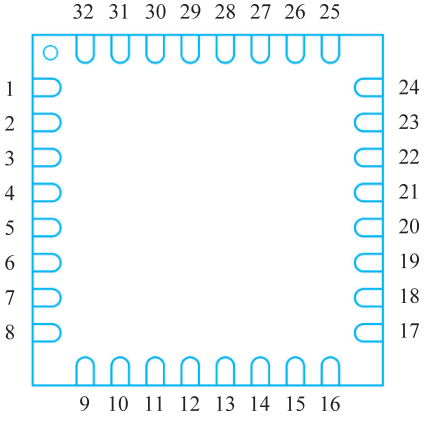

图 5-3　Hi3861 引脚图

第 27 引脚为输出方向,功能为通用输入输出。之所以要进行这些设计,是因为芯片中的引脚是有限的,引脚多了芯片的面积增大、外部电路设计也会变得复杂。

这个 27 端口一共有 9 种功能,在使用时需要用程序将信息传输到芯片,当前想使用哪一种功能。在 iot_gpio_ex 中对这些功能进行了定义。

程序的主体是一个 while 循环,关键变量是 v,其初始值为 1,使用 IoTGpioSetOutputVal 将 v 的值设置到如图 5-4 所示的 GPIO09 上。第一次循环 v 的值为 1,随后 v=1-v, v 的值变为 0,下次循环将 0 输出后,v 的值又翻转为 1。就这样,v 的值一直在 1 和 0 之间翻转,GPIO 输出的电压也在 1 对应的高电平和 0 代表的低电平之间反复转换,GPIO09 连接的灯也会在亮和灭之间切换,形成闪烁的效果。

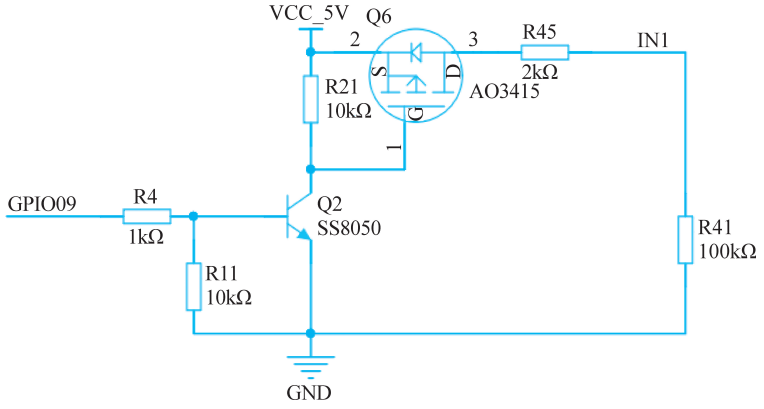

图 5-4 GPIO9 相关电路

函数 usleep 为延迟函数,单位为 μs,程序里为 0.2s。可以通过修改 usleep 的参数观察闪烁频率的变化。

5.2 C 语言鸿蒙 OS 设备开发实验:呼吸灯

5.2.1 呼吸灯项目的程序代码

呼吸灯 C 语言鸿蒙 OS 程序项目由 1 个 C 语言源程序文件 BREATHE.c、2 个 BUILD.gn 文件和 1 个 config.json 文件组成。各个文件在计算机磁盘上的存储情况如图 5-5 所示,文件内容分别如下所示。

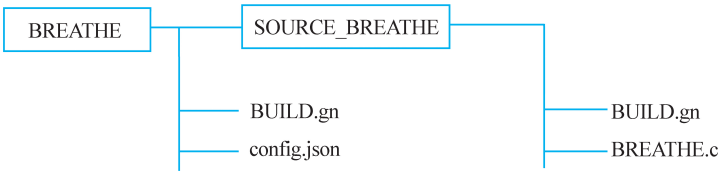

图 5-5 呼吸灯 C 语言鸿蒙 OS 程序项目结构和内容示意图

(1) BREATHE 文件夹下的 BUILD.gn 文件内容。

```
group("BREATHE")
{
    deps = [
          "SOURCE_BREATHE:BREATHE",
          "//device/bossay/hi3861_l0/sdk_liteos:wifiiot_sdk",
          "../common/iot_wifi:iot_wifi",
         ]
}
```

(2) BREATHE 文件夹下的 config.json 文件内容。

```
{
     "product_name": "BREATHE",
    从第 2 行开始,剩余内容跟点亮一只 LED 灯项目的 config.json 文件从第 2 行开始的各行内容相同,在此不再赘述。
}
```

(3) SOURCE_BREATHE 文件夹下的 BUILD.gn 文件内容。

```
static_library("BREATHE")
{
    sources = [ "BREATHE.c",  ]
    include_dirs = [
                "//utils/native/lite/include",
                "//base/iot_hardware/peripheral/interfaces/kits",
                "//device/bossay/hi3861_l0/iot_hardware_hals/include",
                "//device/bossay/hi3861_l0/sdk_liteos/include"
               ]
}
```

(4) SOURCE_BREATHE 文件夹下的 BREATHE.c 文件内容。

```
#include <stdio.h>
#include "ohos_init.h"
#include "iot_gpio.h"
#include "iot_gpio_ex.h"
#include "iot_pwm.h"
#define PWM_GPIO 9
void pwm_entry()
{
  printf("pwm_entry called \n");
  IoTGpioInit(PWM_GPIO);
  IoTGpioSetDir(PWM_GPIO,IOT_GPIO_DIR_OUT);
  IoTGpioSetFunc(PWM_GPIO,IOT_GPIO_FUNC_GPIO_9_PWM0_OUT);
  IoTPwmInit(0);
  int i;
  while(1)
   {
    for(i=0;i<100;i++)
    {
```

```
            IoTPwmStart(0,i,40000);
            usleep(1000 * 10);
        }
        for(i=100;i>=0;i--)
        {
            IoTPwmStart(0,i,40000);
            usleep(1000 * 10);
        }
    }
}
APP_FEATURE_INIT(pwm_entry);
```

5.2.2

5.2.2 编辑、编译、烧录、运行呼吸灯程序

参照本书第 4 章网页编译的方法,将源程序 BREATHE.c 的代码复制到网页进行编译,生成可执行目标代码;或参照第 3 章的方式,利用 VS Code 的 DevEco 工具建立呼吸灯程序项目,编辑程序代码、编译生成可执行目标代码。然后使用 USB-Type 连线连接计算机和开发板,利用 Hiburn 工具烧录可执行目标代码到开发实验板,按下开发板上的 RESET 复位键运行项目程序。

5.2.3

5.2.3 呼吸灯程序的电气工作原理

呼吸灯是像呼吸一样慢慢变亮又慢慢变暗的周期性亮度变化的灯。单片机中无法输出这种不断变化的模拟信号,必须配合相应的电路才能实现,那么怎样控制 LED 的亮度呢?单片机提供了一种通用的解决方案:脉冲宽度调制(Pulse Width Modulation,PWM)方案,即使用周期性的脉冲输出代替持续稳定的电压输出。图 5-6 给出了使用 PWM 模拟 2.5V、3.75V 和 1V 电压输出的样例。该样例中 5V 是基准电压,如果只有一半的时间输出高电平(占空比 50%),那么对于同一个负载来说,功率就是原本的一半,那么就可以模拟 2.5V 电压。同样的原理,如果只有 20%的时间输出高电平,结果就是原本电压的 1/5,即 1V。如果有 75%的时间输出高电平,则为 3.75V。

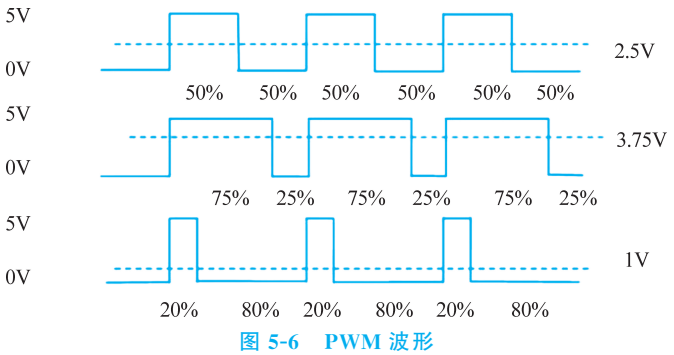

图 5-6 PWM 波形

当然,这样输出的电压是周期性变化的,如果想保持特定电压的稳定输出,就需要将 PWM 输出的信号通过带有电容的滤波电路来实现,对于驱动一个 LED 来说,显然不需要通过这样的滤波电路。那么使用周期性变化的信号来驱动点亮 LED 会不会让肉眼看到闪

烁呢？这个问题等价于看电影时能不能看到闪烁一样，答案是只要闪烁得够快，肉眼就看不到闪烁。

Hi3861 芯片不仅支持 6 个硬件 PWM 输出，而且也可以通过程序代码来控制 PWM 的输出，PWM 的输出不过是周期性地快速拉高拉低电平而已，使用 GPIO 操作就可以实现，这种是软件 PWM。硬件 PWM 的好处是可以达到更高的频率，频率越高越不容易看到闪烁，另外，只需要启动硬件 PWM，程序就可以做别的事情，而不需要进入一个不断地拉高和拉低引脚电平的循环，这样 CPU 空闲以后就可以做别的事情。

鸿蒙系统提供一组 API（应用编程接口）函数实现引脚的 PWM 功能控制。首先通过 IoTGpioInit 初始化引脚，随后使用 IoTGpioSetDir 将方向设为输出，最重要的是使用 IoTGpioSetFunc 将功能设置为 PWM。前面提到支持 6 路硬件 PWM，这意味着只有 6 个引脚可以支持硬件 PWM，它们是 GPIO9、GPIO10、GPIO5、GPIO12、GPIO13、GPIO14，分别对应 PWM0 ～ PWM5，可以从 Hi3861 的用户指南中查到。这里使用 GPIO9 引脚（PWM0）进行实验。

函数 IoTPwmInit 传入 PWM 的编号 0 实现了初始化，函数 IoTPwmStart 则实现了 PWM 的启动。其中，三个参数中的第一个参数传入 0 表示操作的是 PWM0，第二个参数 i 为占空比，最后一个参数为频率，传入了 40000。

为了实现灯慢慢变亮的效果，程序将占空比从 0 逐渐调整到了 100，再慢慢降低至 0。第一个循环实现从 0 到 99，第二个循环实现 100 到 0。

外层嵌套着的 while 循环保证这个过程持续运行，这样一个会呼吸的灯就做好了。

5.3　习　　题

bossay 实验板上呈五角星分布的 5 个 LED 灯的 GPIO 分别是 GPIO9、GPIO10、GPIO11、GPIO12、GPIO13。

（1）设计程序实现流水灯功能，让实验板上呈五角星分布的 LED 灯循环依次点亮。

（2）设计程序实现让实验板上呈五角星分布的 LED 灯全部闪烁。