第3章

轻量级系统设备开发实验

一个人想做点事业，非得走自己的路。

要开创新路子，最关键的是你会不会自己提出问题，能正确地提出问题就是迈开了创新的第一步。

——李政道

3.1 轻量级系统设备开发实验概览

1. 实验目的

本章实验的目的是掌握 OpenHarmony 轻量级系统设备开发的基本功能，包括 GPIO 输入输出、PWM 输出、I2C 输入输出、AT 指令、WiFi 连接和 MQTT 传输数据等。

2. 实验设备

(1) 硬件设备包含一台主流配置的计算机，以及 HiSpark 的 WiFi IoT 套件中的 Hi3861 核心板和环境监测板。

(2) 软件环境主要是 Windows 10 操作系统、DevEco Device Tool 3.0 和 OpenHarmony 3.0 LTS 等。详细的环境配置和安装过程请参考第 1 章。

3. 实验内容

包含 8 个实验，分别为：

(1) Hi3861 GPIO 输出实验。

(2) Hi3861 GPIO 查询方式输入实验。

(3) Hi3861 GPIO 中断方式输入实验。

(4) Hi3861 PWM 输出实验。

(5) Hi3861 I2C 读取 AHT 实验。

(6) Hi3861 AT 指令实验。

(7) Hi3861 WiFi 连接实验。

(8) Hi3861 MQTT 客户端实验。

4. 实验过程

本章所有实验遵循以下步骤。

(1) 环境准备。

(2) 工程配置。

(3) 在 OpenHarmony SDK 相关目录新建源文件。

(4) 构建编译环境。

(5) 编译工程。

(6) 烧写到 Hi3861 核心板。

(7) 运行系统。

(8) 观察实验板现象以及通过 DevEco Device Tool 串口终端查看结果。

OpenHarmony SDK 源码的主要目录如表 3.1 所示。

表 3.1　源码的目录

目　录　名	描　　述
applications	应用程序样例,包括 camera 等
ark	方舟编译
base	基础软件服务子系统集 & 硬件服务子系统集
build	组件化编译、构建和配置脚本
docs	说明文档
domains	增强软件服务子系统集
drivers	驱动子系统
foundation	系统基础能力子系统集
kernel	内核子系统
prebuilts	编译器及工具链子系统
test	测试子系统
third_party	开源第三方组件
utils	常用的工具集
vendor	厂商提供的软件
build.py	编译脚本文件

3.2　Hi3861 GPIO 输出实验

1. 实验内容

本实验是在 Hi3861 核心板上实现通过 GPIO 控制 LED 灯的功能。

2. 实验代码

在 OpenHarmony 3.0 LTS 版本中,在\applications\sample\wifi-iot\app 目录下创建 led_demo 目录,在 led.demo 目录下创建名为 led_example.c 的文件。

通过 GPIO 控制 LED 的 led_example.c 文件完整代码如下。

```
#include <stdio.h>
        #include <unistd.h>
        #include "ohos_init.h"
        #include "cmsis_os2.h"
```

```
#include "iot_gpio.h"

#define LED_INTERVAL_TIME_US 300000
#define LED_TASK_STACK_SIZE 512
#define LED_TASK_PRIO 25
/* 实现 IOT 外设控制,首先需要通过查阅原理图明确接线关系.经过查阅,发现 hispark pegasus 的 LED 与芯片的 9 号管脚相连 */
#define LED_TEST_GPIO 9 //for hispark_pegasus
enum LedState {
    LED_ON = 0,
    LED_OFF,
    LED_SPARK,
};

//在循环任务中通过周期性亮灭形式实现 LED 闪烁
enum LedState g_ledState = LED_SPARK;

static void *LedTask(const char *arg)
{
    (void)arg;
    while (1) {
        switch (g_ledState) {
            case LED_ON:
            IoTGpioSetOutputVal(LED_TEST_GPIO, 0);
            usleep(LED_INTERVAL_TIME_US);
            break;
            case LED_OFF:
            IoTGpioSetOutputVal(LED_TEST_GPIO, 1);
            usleep(LED_INTERVAL_TIME_US);
            break;
            case LED_SPARK:
            IoTGpioSetOutputVal(LED_TEST_GPIO, 0);
            usleep(LED_INTERVAL_TIME_US);
            IoTGpioSetOutputVal(LED_TEST_GPIO, 1);
            usleep(LED_INTERVAL_TIME_US);
            break;
            default:
            usleep(LED_INTERVAL_TIME_US);
            break;
        }
    }

    return NULL;
}

static void LedExampleEntry(void)
{
    osThreadAttr_t attr;

    /* 管脚初始化 */
    IoTGpioInit(LED_TEST_GPIO);
```

```
        /* 配置 9 号管脚为输出方向 */
        IoTGpioSetDir(LED_TEST_GPIO, IOT_GPIO_DIR_OUT);

        attr.name = "LedTask";
        attr.attr_bits = 0U;
        attr.cb_mem = NULL;
        attr.cb_size = 0U;
        attr.stack_mem = NULL;
        attr.stack_size = LED_TASK_STACK_SIZE;
        attr.priority = LED_TASK_PRIO;

        /* 启动任务 */
        if (osThreadNew((osThreadFunc_t)LedTask, NULL, &attr) == NULL) {
                printf("[LedExample] Falied to create LedTask!\n");
        }
    }

    SYS_RUN(LedExampleEntry);
```

在包含示例程序 led_example.c 的目录下，创建 BUILD.gn 文件，文件内容如下。

```
//创建名为 led_demo 的静态库
    static_library("led_example") {
            //编译该静态库需要的源代码文件列表
            sources = [
            "led_example.c"
             ]
             //编译该静态库需要的包含目录列表
             include_dirs = [
             "//utils/native/lite/include",//ohos_init.h 所在目录
             "//kernel/liteos_m/kal/cmsis",//cmsis_os 2.h所在目录
             "//base/iot_hardware/peripheral/interfaces/kits",//iot_gpio.h 所在目录
             ]
    }
```

在上级目录下，修改 applications/sample/wifi-iot/app/BUILD.gn 文件，使 led_example.c 参与编译。

```
import("//build/lite/config/component/lite_component.gni")
    lite_component("app") {
            features = [
            "iothardware: led_example"
            ]
    }
```

3. 实验结果

烧录完成后。复位 Hi3861 核心板，板载 LED 灯不停闪烁，如图 3.1 所示。

4. 扩展

Hi3861 的 GPIO 接口具有以下功能特点。

(1) 时钟源可选择：工作模式晶体时钟频率 24MHz/40MHz、低功耗模式 32kHz 时钟

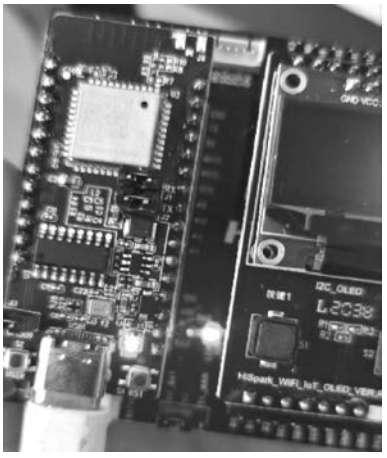

图 3.1 GPIO 控制 LED 灯闪烁

频率。

(2) 1 组 GPIO,共 15 个独立的可配置管脚。

(3) 每个 GPIO 管脚都可单独控制传输方向。

(4) 每个 GPIO 可以单独被配置为外部中断源。

(5) GPIO 用作中断时有 4 种中断触发方式,中断时触发方式可配:上升沿触发,下降沿触发,高电平触发,低电平触发。

(6) GPIO 上报一个中断,CPU 查询上报的 GPIO 编号。

(7) 中断支持独立屏蔽的功能,脉冲中断支持可清除功能。

3.3 Hi3861 GPIO 查询方式输入实验

1. 实验内容

本实验是在 Hi3861 核心板上,完成通过 GPIO 实现按键对 LED 灯的控制功能。

2. 实验代码

在 OpenHarmony 3.0 LTS 版本中,在\applications\sample\wifi-iot\app 目录下创建 iotkey 目录,在 iotkey 目录下创建名为 keyPolling.c 的文件。

通过按键控制 LED 的 keyPolling.c 文件完整代码列表如下。

```
#include <stdio.h>

#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "iot_gpio.h"
#include "iot_gpio_ex.h"

#define LED_INTERVAL_TIME_US 300000
#define LED_TASK_STACK_SIZE 4096
```

```
#define LED_TASK_PRIO 25
#define LED_TEST_GPIO 9 //for hispark_pegasus
#define KEY_GPIO 5
enum KeyState {
      KEY_DOWN = 0,
      KEY_UP,
};
enum LedState {
      LED_ON = 0,
      LED_OFF,
      LED_SPARK,
};

enum LedState g_ledState = LED_SPARK;

static void * KeyPollingTask(const char * arg)
{
       (void)arg;
       printf("\n Task is running!\n");
       while (1) {
             IotGpioValue value = IOT_GPIO_VALUE0;
             IoTGpioGetInputVal(KEY_GPIO, &value);
             if (value == KEY_UP){
                    switch (g_ledState) {
                          case LED_ON:
                          IoTGpioSetOutputVal(LED_TEST_GPIO, 0);
                          usleep(LED_INTERVAL_TIME_US);
                          break;
                          case LED_OFF:
                          IoTGpioSetOutputVal(LED_TEST_GPIO, 1);
                          usleep(LED_INTERVAL_TIME_US);
                          break;
                          case LED_SPARK:
                          IoTGpioSetOutputVal(LED_TEST_GPIO, 0);
                          usleep(LED_INTERVAL_TIME_US);
                          IoTGpioSetOutputVal(LED_TEST_GPIO, 1);
                          usleep(LED_INTERVAL_TIME_US);
                          break;
                          default:
                          usleep(LED_INTERVAL_TIME_US);
                          break;
                    }
             }

       }

       return NULL;
}

static void KeyPollingEntry(void)
{
```

```
    osThreadAttr_t attr;

    IoTGpioInit(LED_TEST_GPIO);
    IoTGpioSetDir(LED_TEST_GPIO, IOT_GPIO_DIR_OUT);

    IoTGpioDeinit(KEY_GPIO);
    IoTGpioInit(KEY_GPIO);
    IoSetFunc(IOT_IO_NAME_GPIO_5, IOT_IO_FUNC_GPIO_5_GPIO);
    IoTGpioSetDir(KEY_GPIO, IOT_GPIO_DIR_IN);
    IoSetPull(KEY_GPIO,1);


    attr.name = "KeyPollingTask";
    attr.attr_bits = 0U;
    attr.cb_mem = NULL;
    attr.cb_size = 0U;
    attr.stack_mem = NULL;
    attr.stack_size = LED_TASK_STACK_SIZE;
    attr.priority = LED_TASK_PRIO;

    if (osThreadNew((osThreadFunc_t)KeyPollingTask, NULL, &attr) == NULL) {
        printf("[LedExample] Falied to create LedTask!\n");
    }
}

SYS_RUN(KeyPollingEntry);
```

在包含示例程序 keyPolling.c 的目录下，创建 BUILD.gn 文件，文件内容如下。

```
//定义名为 iotkey 的静态库
    static_library("iotkey") {
        //编译该静态库需要的源代码文件列表
        sources = [
        "keyPolling.c"
        ]
        //编译该静态库需要的包含目录列表
        include_dirs = [
        "//utils/native/lite/include",
        "//kernel/liteos_m/kal/cmsis",
        "//base/iot_hardware/peripheral/interfaces/kits",
        ]
    }
```

在上级目录下，修改 applications/sample/wifi-iot/app/BUILD.gn 文件，使 keyPolling.c 参与编译。

```
import("//build/lite/config/component/lite_component.gni")
    lite_component("app") {
        features = [
        "iotkey: iotkey"
        ]
    }
```

3. 实验结果

烧录完成后。复位 Hi3861 核心板，在 DevEco Device Tool 的串口监视窗口，输出如下内容。

```
ready to OS start
sdk ver: Hi3861V100R001C00SPC025 2020 - 09 - 03 18: 10: 00
FileSystem mount ok.
wifi init success!
hilog will init.
hievent will init.
hievent init success.

Task is doing!
hiview init success. No crash dump found!
```

同时板载 LED 灯不停闪烁，按下 S2 按键时 LED 灯停止闪烁。

3.4　Hi3861 GPIO 中断方式输入实验

1. 实验内容

本实验是在 Hi3861 核心板上，完成通过 GPIO 实现按键中断事件对 LED 灯的控制功能。

2. 实验代码

在 OpenHarmony 3.0 LTS 版本中，在\applications\sample\wifi-iot\app 目录下创建 iotkey 目录，在 iotkey 目录下创建名为 keyInt.c 的文件。

通过按键中断事件控制 LED 的 keyInt.c 文件完整代码如下。

```
#include <stdio.h>

#include <unistd.h>

#include "ohos_init.h"
#include "cmsis_os2.h"
#include "wifiiot_gpio.h"
#include "wifiiot_gpio_ex.h

#define LED_INTERVAL_TIME_US 300000
#define LED_TASK_STACK_SIZE 512
#define LED_TASK_PRIO 25
#define LED_TEST_GPIO 9 //for hispark_pegasus
#define KEY_GPIO 5
enum KeyState {
      KEY_DOWN = 0,
      KEY_UP,
};
enum LedState {
```

```
    LED_ON = 0,
    LED_OFF,
    LED_SPARK,
};

enum LedState g_ledState = LED_SPARK;

static volatile WifiIotGpioValue value = WIFI_IOT_GPIO_VALUE0;

static void OnKeyDown(char *arg)
{
    (void)arg;
    //WifiIoTGpioSetIsrMask(KEY_GPIO, 1);
    value = !value;
    //WifiIoTGpioSetIsrMask(KEY_GPIO, 0);
}

static void *KeyIntTask(const char *arg)
{
    (void)arg;

    while (1) {

        if (!value){
            switch (g_ledState) {
                case LED_ON:
                GpioSetOutputVal(LED_TEST_GPIO, 0);
                usleep(LED_INTERVAL_TIME_US);
                break;
                case LED_OFF:
                GpioSetOutputVal(LED_TEST_GPIO, 1);
                usleep(LED_INTERVAL_TIME_US);
                break;
                case LED_SPARK:
                GpioSetOutputVal(LED_TEST_GPIO, 0);
                usleep(LED_INTERVAL_TIME_US);
                GpioSetOutputVal(LED_TEST_GPIO, 1);
                usleep(LED_INTERVAL_TIME_US);
                break;
                default:
                usleep(LED_INTERVAL_TIME_US);
                break;
            }
        }

    }

    return NULL;
}

static void KeyIntEntry(void)
```

```
{
        osThreadAttr_t attr;

        GpioInit(LED_TEST_GPIO);
        GpioSetDir(LED_TEST_GPIO, WIFI_IOT_GPIO_DIR_OUT);

        GpioInit(KEY_GPIO);
        GpioSetDir(KEY_GPIO, WIFI_IOT_GPIO_DIR_IN);
        IoSetFunc(WIFI_IOT_IO_NAME_GPIO_5,WIFI_IOT_IO_PULL_UP);
        GpioRegisterIsrFunc(KEY_GPIO,
        WIFI_IOT_INT_TYPE_EDGE,
        WIFI_IOT_GPIO_EDGE_FALL_LEVEL_LOW,
        OnKeyDown,NULL);

        attr.name = "KeyIntTask";
        attr.attr_bits = 0U;
        attr.cb_mem = NULL;
        attr.cb_size = 0U;
        attr.stack_mem = NULL;
        attr.stack_size = LED_TASK_STACK_SIZE;
        attr.priority = LED_TASK_PRIO;

        if (osThreadNew((osThreadFunc_t)KeyIntTask, NULL, &attr) == NULL) {
              printf("[InterruptExample] Falied to create KeyIntTask!\n");
        }
}

SYS_RUN(KeyIntEntry);
```

在包含示例程序 keyInt.c 的目录下，修改 BUILD.gn 文件，文件内容如下。

```
//定义名为 iotkey 的静态库
        static_library("iotkey") {
                //编译该静态库需要的源代码文件列表
                sources = [
                "keyInt.c"
                ]
                //编译该静态库需要的包含目录列表
                include_dirs = [
                "//utils/native/lite/include",
                "//kernel/liteos_m/kal/cmsis",
                "//base/iot_hardware/peripheral/interfaces/kits",
                ]
        }
```

在上级目录下，修改 applications/sample/wifi-iot/app/BUILD.gn 文件，使 keyInt.c 参与编译。

```
import("//build/lite/config/component/lite_component.gni")
        lite_component("app") {
                features = [
```