HiSpark WiFi-IoT NFC 驱动编程指南

文档版本　00B01

发布日期　2020/8/6

商标声明

 、HISILICON、海思和其他海思商标均为上海海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址：　　上海市青浦区金泽镇（西岑）水秀路 318 号 101 室　邮编：201718

网址：　　http://www.hisilicon.com

前　言

概述

本文档主要介绍基于海思 WiFi 芯片 Hi3861 开发的 HiSpark-WiFi-IoT 套件演示指导书。

产品版本

与本文档相对应的主芯片版本如下。

产品名称	产品版本
Hi3861	V100R001C00SPC021

读者对象

本文档（本指南）主要适用于以下工程师：

- 软件开发工程师
- 硬件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2020-08-6	00B01	第一次临时版本发布。

目 录

1 WIFI-IoT NFC 驱动&应用拉起功能实现

1.1 WIFI-IOT 开发板套件——NFC 通讯介绍

1. NFC 是一种近场无线通信技术，通信距离理论上可以在 10cm（实际中需要贴的很近），以 13.56MHz RFID 技术为基础，与现有的非接触式智能卡国际标准相兼容。数据传输速率 106kbit/s、212kbit/s、424kbit/s。

2．通信原理是基于感应近场，在近场区域内感应场强弱与电磁辐射源以及天线的距离相关，近则强远则弱。

3. NFC 通信一般有两种模式：主动模式和被动模式。

1）主动模式：两个设备发起端（标签设备）和目标端（读卡器）都必须发射出本身的射频场，这样他们才可以向对方系统设备之间发送数据。

2）被动模式：相对于主动模式，被动模式只有一方提供射频场，提供射频场的都是通信发起端设备，另一端目标不需要产生射频场，目标设备能量的产生由发起方射频场的感应电动势进行供电，目标端使用负载调制的方式，以相同的速率将数据回传给发起端设备。在整个双方通信过程中发起方设备的射频场必须存在，一旦关闭目标端设备的供电就会结束，数据交换无法进行。

4. HiSpark 开发板采用上海复旦微电子 FM11NC08I 动态 Tag （标签）NFC 芯片，通信协议采用 ISO/IEC 14443-A，内置 8Kbit EEPROM,芯片详情参照上海复旦微电子 FM11NC08I datasheet。

图 1.1-1 被动模式

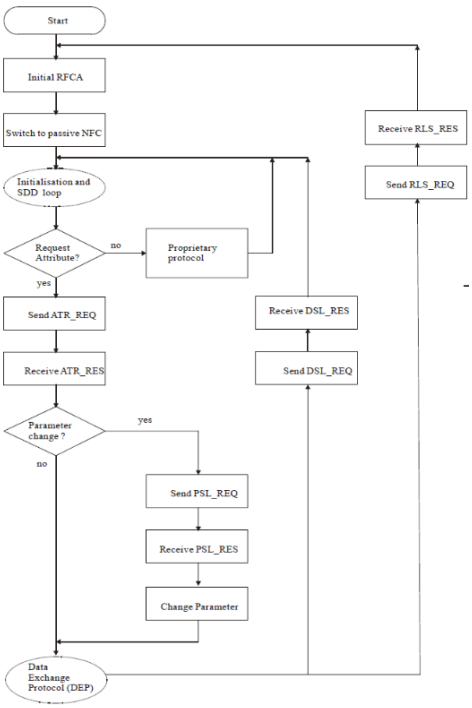
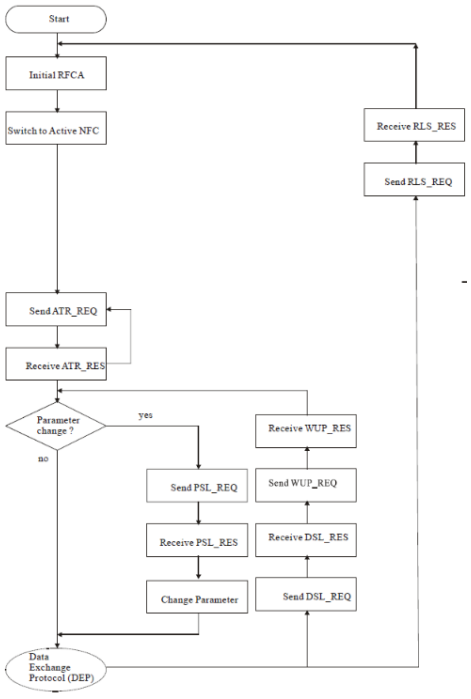


图 1.1-2 主动模式



1.2 FM11NC08I NFC 驱动实现逻辑

1. HiSpark 开发板中，代码中采用 I2C 通信驱动 NFC 芯片，通过写 EEPROM 可更改 NFC 芯片的配置，达到大容量读写的目的（芯片出厂默认可写 32 字节）。波特率为 400kbit/s, 可设置两种方式驱动，中断或查询方式。

2. HiSpark 开发板的 NFC 实现逻辑：代码中创建了一个 NFC 驱动的 task 任务，20ms 读取一次是否有 NFC 设备靠近，当有 NFC 设备（如的手机）靠近，HiSpark 上的 NFC 芯片回去读头信息，当读取到头信息后，HiSpark 上的 NFC 芯片给设备回复信息，建立通信，这时 NFC 设备可读取 HiSpark 上的 NDEF 信息拉起应用/打开网址等；也可向 HiSpark 上的 NFC 写信息，并保存在 HiSpark 开发板的 WiFI-IOT 芯片上，NFC 芯片本身不存储任何传输的信息。

3. HiSpark 开发板的 NFC 采用 I2C_0 通道进行数据通信，对应的 GPIO 引脚分别是 GPIO13（SDA）、GPIO14（SCL），波特率设置为 400kbit/s。引脚复用关系如下：

图 1.2-1 引脚复用

GPIO_10		UART2_CTS	SPI0_CLK			I2C0_SDA
GPIO_11		UART2_TXD	SPI0_RXD		ADC5	
GPIO_12		UART2_RXD	SPI0_CS1		ADC0	
GPIO_13	UART0_LOG_TXD	UART2_RTS			ADC6	I2C0_SDA
GPIO_14	UART0_LOG_RXD	UART2_CTS				I2C0_SCL

1.3 NFC 功能软件实现

1. LiteOS 打开 WIFI-IOT 的 SDK 后，首先初始化 GPIO 配置

在 hi_void app_io_init(hi_void)接口中配置 I2C GPIO 复用初始化：

图 1.3-1 I2C GPIO 复用初始化

```
/*i2c 0*/
hi_io_set_func(HI_IO_NAME_GPIO_13, HI_IO_FUNC_GPIO_13_I2C0_SDA);
hi_io_set_func(HI_IO_NAME_GPIO_14, HI_IO_FUNC_GPIO_14_I2C0_SCL);
```

2. hi_u32 hi_io_set_func(hi_io_name id, hi_u8 val)：配置某个 I/O 复用功能；

其中参数含义 hi_io_name id：选择要配置的 I/O 硬件引脚，枚举类型如下：

```
typedef enum {
    HI_IO_NAME_GPIO_0,      /**< GPIO0 */
    HI_IO_NAME_GPIO_1,      /**< GPIO1 */
    HI_IO_NAME_GPIO_2,      /**< GPIO2 */
    HI_IO_NAME_GPIO_3,      /**< GPIO3 */
    HI_IO_NAME_GPIO_4,      /**< GPIO4 */
    HI_IO_NAME_GPIO_5,      /**< GPIO5 */
    HI_IO_NAME_GPIO_6,      /**< GPIO6 */
    HI_IO_NAME_GPIO_7,      /**< GPIO7 */
    HI_IO_NAME_GPIO_8,      /**< GPIO8 */
    HI_IO_NAME_GPIO_9,      /**< GPIO9 */
    HI_IO_NAME_GPIO_10,     /**< GPIO10 */
    HI_IO_NAME_GPIO_11,     /**< GPIO11 */
    HI_IO_NAME_GPIO_12,     /**< GPIO12 */
    HI_IO_NAME_GPIO_13,     /**< GPIO13 */
    HI_IO_NAME_GPIO_14,     /**< GPIO14 */
    HI_IO_NAME_SFC_CSN,      /**< SFC_CSN */
    HI_IO_NAME_SFC_IO1,     /**< SFC_IO1 */
    HI_IO_NAME_SFC_IO2,     /**< SFC_IO2 */
    HI_IO_NAME_SFC_IO0,     /**< SFC_IO0 */
    HI_IO_NAME_SFC_CLK,      /**< SFC_CLK */
    HI_IO_NAME_SFC_IO3,     /**< SFC_IO3 */
    HI_IO_NAME_MAX,
} hi_io_name;
```

hi_u8 val：参数 val 为将 GPIO 口复用哪种功能（本文将其复用为 I2C 功能）。

3. HiSpark 开发板 NFC 驱动软件实现：

1）创建 NFC task 任务：

```
/*c08i nfc task*/
hi_void app_c08i_nfc_i2c_demo_task(hi_void)
{
    hi_u32 ret =0;
    hi_task_attr attr = {0};
    hi_task_lock();
    attr.stack_size = C08I_NFC_DEMO_TASK_STAK_SIZE;
    attr.task_prio = C08I_NFC_DEMO_TASK_PRIORITY;
    attr.task_name = (hi_char*)"app_i2c_nfc_demo";
    ret = hi_task_create(&g_c08i_nfc_demo_task_id, &attr, app_i2c_nfc_demo, HI_NULL);
    if (ret != HI_ERR_SUCCESS) {
        printf("Falied to create i2c c08i_nfc demo task!\n");
    }
    hi_task_unlock();
}
```

2）NFC 驱动实现（查询方式）：

```
#ifdef CHECK
    /*查询方式*/
    while (1) {
        irq_data_wl = 0;
        irq = fm11_read_reg(MAIN_IRQ);//查询中断标志
        if (irq & MAIN_IRQ_FIFO) {
            ret=fm11_read_reg(FIFO_IRQ);
            if(ret & FIFO_IRQ_WL)
            irq_data_wl = 1;
        }
        if (irq & MAIN_IRQ_AUX) {
            fm11_read_reg(AUX_IRQ);
            fm11_write_reg(FIFO_FLUSH,0xFF);
        }

        if (irq& MAIN_IRQ_RX_START) {
          irq_data_in = 1;
        }

         if (irq_data_in && irq_data_wl) {
            irq_data_wl =0;
            fm11_read_fifo(24,&rbuf[rlen]);//渐满之后读取24字节
            rlen += 24;
        }

        if (irq & MAIN_IRQ_RX_DONE) {
            temp =(hi_u32)( fm11_read_reg(FIFO_WORDCNT) & 0x3F);     //接收完全之后，查fifo有多少字节
            fm11_read_fifo(temp,&rbuf[rlen]);        //读最后的数据
            rlen += temp;
            irq_data_in = 0;
            break;
        }
        hi_sleep(1);
    }
#endif
```

驱动实现（中断方式）：

```
#ifdef INTERRUPT
    while (1) {
        irq_data_wl = 0;
        irq = fm11_read_reg(MAIN_IRQ);//查询中断标志

        if (irq & MAIN_IRQ_FIFO) {
            ret=fm11_read_reg(FIFO_IRQ);
            if(ret & FIFO_IRQ_WL)
            irq_data_wl = 1;
        }
        if (irq & MAIN_IRQ_AUX) {
            fm11_read_reg(AUX_IRQ);
            fm11_write_reg(FIFO_FLUSH,0xFF);
        }

        if (irq& MAIN_IRQ_RX_START) {

          irq_data_in = 1;
        }

         if (irq_data_in && irq_data_wl) {
            irq_data_wl =0;
            fm11_read_fifo(24,&rbuf[rlen]);//渐满之后读取24字节
            rlen += 24;
        }

        if (irq & MAIN_IRQ_RX_DONE) {
            temp =(hi_u32)( fm11_read_reg(FIFO_WORDCNT) & 0x3F);    //接收完全之后，查fifo有多少字节
            fm11_read_fifo(temp,&rbuf[rlen]);       //读最后的数据
            rlen += temp;
            irq_data_in = 0;
            break;
        }
        hi_sleep(1);
    }
#endif
```

当有 NFC 设备靠近时，中断机制启动：

```
/*NFC 中断句柄*/
hi_void *gpio_isr_handle(hi_void* param)
{
    hi_u8 ret = 0;

    printf("----- gpio isr handle -----\r\n");
    while (1) {
        hi_sem_wait(g_nfc_sem_id, HI_SYS_WAIT_FOREVER);
        ret = fm11_read_reg(MAIN_IRQ_REG);
        if (ret & MAIN_IRQ_ACTIVE) {
            FlagFirstFrame = 1;
        }
        hi_sleep(1);
    }
    return (HI_NULL);
}
/*NFC 中断方式通信task*/
hi_void gpio_isr_task(hi_void)
{
    hi_u32 ret;
    hi_task_attr attr = {0};
    /*创建二值信号量同步中断响应*/
    hi_sem_bcreate(&g_nfc_sem_id, HI_SEM_ZERO);

    hi_task_lock();
    attr.stack_size = NFC_INTERRUPT_TASK_STAK_SIZE;
    attr.task_prio = NFC_INTERRUPT_TASK_PRIORITY;
    attr.task_name = (hi_char*)"gpio_isr_task";
    ret = hi_task_create(&g_nfc_isr_task_id, &attr, gpio_isr_handle, HI_NULL);
    if (ret != HI_ERR_SUCCESS) {
        printf("Falied to create gpio_isr task!\n");
    }
    hi_task_unlock();
}
```

3）当需要大容量传输数据时，更改 NFC 配置：

```
/* NFC 芯片配置 ,平时不要调用 NFC init*/
hi_void nfc_init(hi_void)
{
    //  uint8_t wbuf[5]={0x05,0x72,0xF7,0x60,0x02};   //芯片默认配置
    hi_u8 wbuf[5]={0x05,0x78,0xF7,0x90,0x02};   //芯片默认配置
    /*读取字节的时候屏蔽csn引脚,写eep的时候打开*/
    hi_io_set_func(HI_IO_NAME_GPIO_9, HI_IO_FUNC_GPIO_9_GPIO);
    hi_gpio_set_dir(HI_GPIO_IDX_9, HI_GPIO_DIR_OUT);
    hi_gpio_set_ouput_val(HI_IO_NAME_GPIO_9, HI_GPIO_VALUE0);

    fm11_write_eep(0x3B1,1,&wbuf[1]);
    fm11_write_eep(0x3B5,1,&wbuf[3]);
}
```

设置完毕后，将 nfc_init()接口屏蔽，再次编译、烧录。

4）写 NFC 芯片的 EEPROM 如下：

```
/*写EEPROM*/
hi_void fm11_write_eep(hi_u16 addr,hi_u32 len,hi_u8 *wbuf)
{
    hi_u8 offset;
    if (addr < FM11_E2_USER_ADDR || addr >= FM11_E2_MANUF_ADDR) {
        return;
    }
    if (addr % FM11_E2_BLOCK_SIZE) {
        offset = FM11_E2_BLOCK_SIZE - (addr % FM11_E2_BLOCK_SIZE);
        if (len > offset) {
            eep_write_page(wbuf,addr,offset);

            addr += offset;
            wbuf += offset;
            len -= offset;
        } else {
            eep_write_page(wbuf,addr,len);
            len = 0;
        }
     }
    while (len) {
        if (len >= FM11_E2_BLOCK_SIZE) {
            eep_write_page(wbuf,addr,FM11_E2_BLOCK_SIZE);
            addr += FM11_E2_BLOCK_SIZE;
            wbuf += FM11_E2_BLOCK_SIZE;
            len -= FM11_E2_BLOCK_SIZE;
        } else {
            eep_write_page(wbuf,addr,len);
            len = 0;
        }
    }
}
```

5）NFC 驱动 Demo 如下：

```
/*app nfc demo*/
hi_void *app_i2c_nfc_demo(hi_void* param)
{
    hi_u32 ret =0;
    hi_i2c_init(HI_I2C_IDX_0, HI_I2C_IDX_BAUDRATE);//baud 400k
    hi_i2c_set_baudrate(HI_I2C_IDX_0, HI_I2C_IDX_BAUDRATE);
    /*更改NFC芯片信息，写EEPROM ,平时不要打开*/
    //nfc_init();
#ifdef  CHECK
    while (1) {

        rfLen = fm11_data_recv(fm327_fifo);     //读取rf数据(一帧)
        if(rfLen > 0){
            fm11_t4t();
            irq_data_in = 0;
        }
        hi_sleep(1);
    }
#endif

#ifdef INTERRUPT
    while (1) {
        if (FlagFirstFrame) {
            rfLen = fm11_data_recv(fm327_fifo);     //读取rf数据(一帧)
            if (rfLen > 0) {
                fm11_t4t();
            }
        irq_data_in = 0;
        // FlagFirstFrame =0;
        }
        hi_sleep(1);
    }
#endif
}
```

6）NFC 传输数据配置：

```
hi_u8 capability_container[15] =
{   0x00, 0x0F,         //CCLEN
    0x20,               //Mapping Version
    0x00, 0xF6,         //MLe 必须是F6  写成FF超过256字节就会分帧  但是写成F6就不会分帧
    0x00, 0xF6,         //MLc 必须是F6  写成FF超过256字节就会分帧  但是写成F6就不会分帧
    0x04,               //NDEF消息格式 05的话就是私有
    0x06,               //NDEF消息长度
    0xE1, 0x04,         //NDEF FILE ID       NDEF的文件标识符
    0x03, 0x84,         //NDEF最大长度
    0x00,               //Read Access           可读
    0x00                //Write Access          可写
};
```

7）标签格式（可根据用户需要进行更改）：

```
hi_u8 ndef_file_wechat[1024]= {
    0x00,0x20,
    0xd4, 0x0f,0x0e, 0x61, 0x6e, 0x64, 0x72, 0x6f,
    0x69, 0x64,0x2e, 0x63, 0x6f, 0x6d, 0x3a, 0x70,
    0x6b, 0x67,0x63, 0x6f, 0x6d, 0x2e, 0x74, 0x65,
    0x6e, 0x63,0x65, 0x6e, 0x74, 0x2e, 0x6d, 0x6d,
};
```