

HS6230 使用介绍-V1.0



V1.0

2023/05/30

修改历史

版本	历史版本	日期	作者
V1.0	初始版本	2023-05-30	钟 科

OnMicro Confidential

目录

修改历史.....	1
图表目录.....	3
表格目录.....	4
1. 概述.....	5
2. HS6230 SPI 驱动编写介绍.....	6
3. HS6230 读写 RF 寄存器的操作.....	8
4. HS6230 操作 RF 函数接口介绍.....	9
5. HS6230 的 APP 测试程序.....	11
5.1 HS6230 载波测试.....	12
5.2 HS6230 睡眠功耗测试.....	13
5.3 HS6230 按键值读取.....	14
5.4 HS6230 按键唤醒 MCU.....	15
5.5 HS6230 发送数据包的操作.....	16
5.6 HS6230 发送数据包重用功能.....	17
5.7 HS6230 切换地址，切换功率，切换频点，发送数据包的操作.....	19
6. HS6230 通信频点的要求.....	20
7. HS6230 通信地址的要求.....	20
8. HS6230 硬件设置.....	21
9. HS6230 功耗数据.....	21

图表目录

图表 1 HS6230 的 SPI 驱动配置	6
图表 2 SPI 操作的接口函数	7
图表 3 封装起来后的对外 SPI 读写函数	8
图表 4 操作 RF 的接口函数	10
图表 5 HS6230 初始化校准函数	11
图表 6 HS6230 载波测试图片	12
图表 7 HS6230 载波测试代码	13
图表 8 HS6230 测试睡眠功耗的代码	14
图表 9 HS6230 按键原理图和按键真值表	15
图表 10 HS6230 的按键检测功能测试代码	15
图表 11 HS6230 按键唤醒 MCU 的操作	16
图表 12 HS6230 发送数据包的操作	17
图表 13 HS6230 数据包重用命令	18
图表 14 不使用数据包重用命令	18
图表 15 HS6230 数据包重用命令的测试代码	19
图表 16 HS6230 切换地址，功率，频点，发数据包的操作	20
图表 17 HS6230 不使用内部按键时 2.4G 模式的睡眠功耗	22
图表 18 HS6230 广播间隔 30ms 的功耗	22

表格目录

表 1-1 xxxx.....	6
表 1-2 xxxx.....	6

OnMicro Confidential

1. 概述

HS6230 是北京昂瑞微电子有限公司(以下简称“昂瑞微”)2019 年推出的一款 2.4G 单发芯片。是在原来 HS6200 的基础上裁减掉接收部分，另外加入了一些新的功能。本使用手册是针对 HS6230 如何使用做详细的介绍，尽力写详细，使没接触过 HS6230 的客户也能很快上手。

HS6230 的调制方式是 FSK，只有 1M 和 250K 的速率，没有 2M 速率。信道从 2.4G 到 2.4835G。数据包格式兼容 HS6200，NORDIC 还有 BLE 标准 31 字节广播包。当然 HS6230 也兼容昂瑞微所有 SOC 芯片的 2.4G 私有协议通信。晶体使用的是 16M， $\pm 60\text{ppm}$ ，负载电容小于 12pf 的晶体。可以不需要外挂负载电容，通过调整内部电容来调整频偏，当然这个频偏跟板子也是有很大关系的，并不一定能通过调整寄存器来把频偏校得准。支持 3 线 SPI 和 4 线 SPI，但是不能软件切换，需要硬件打线的时候确认好 SPI 是 3 线还是 4 线。可以硬件控制 CE，也可以软件控制 CE。可以使用 HS6230 内部的 2 个按键脚做按键检测，但是这 2 个脚做 2X4 的按键检测是有按键优先级和组合区别的，这个按键检测同样可以配置成唤醒 MCU 的功能。最大发射功率做到 12dB，但是使用大功率的时候要有 0.5pF 到 1pF 左右的电容到地。初始化流程简单，如果是要睡眠的话，初始化的地方要写一下寄存器保证睡眠的时候校准数据不丢失。睡眠的时候电流在 1~2uA 左右。12dB 发射的时候电流在 40 多 mA 左右。工作电压在 1.9v 到 3.6v。内部没有 LDO，所以供电电压低的时候发射功率会变小。

2. HS6230 SPI 驱动编写介绍

刚才前面有提到 HS6230 支持 3 线 SPI 和 4 线 SPI, 但是是在封装的时候就通过打线确认好了的。如果使用模块做调试的时候, 是通过 MOD 这个脚的接高(4 线)接低(3 线)来选择, **但是一定要注意更改后要重新给模块上电以使芯片重新正确 SPI 线数**。使用 4 线 SPI 的时候, 可以配合 MCU 的硬件 SPI 速度快, 也可以使用模拟的 4 线 SPI, 速度就慢一些, 但是 3 线 SPI 就只能是模拟的了。**在编写 3 线 SPI 驱动的时候一定要只有, SDIO (MISO, 非 MOSI 脚)**这个脚是写的时候做为输出, 读的时候做为输入, 有些人写驱动的时候, 再读这个操作的时候容易搞错, 不知道读的是哪一个脚的电平, 这个要特别注意。我们提供的 SDK 里面已经做好了硬件 4 线 SPI, 软件模拟 4 线 SPI, 以及软件模拟 3 线 SPI 的驱动, 客户只需要修改 bsp_spi.h 这个文件里面的宏定义即可。SDK 是基于 STM32F103RBT6 来写的, 开发板上面有 2 个 RF 模块接口, 所以客户会看到有 RF 端口选择这个宏, 但是这个开发板已经没有了, 不提供给客户, 客户需自行找开发板。包括是否使用 IRQ 脚, 是否硬件控制 CE, 都是宏控制的。

```
bsp_spi.h
7  *****
8
9  #ifndef BSP_SPI_H
10 #define BSP_SPI_H
11
12 #include "stm32f10x.h"
13
14 #define WRITE_HIGH      1 // CE写高
15 #define WRITE_LOW       0 // CE写低
16
17 #define SPI_PORT1        1 // 使用SPI0端口
18 #define SPI_PORT2        2 // 使用SPI1端口
19
20 #define SPI_HARD         0 // 硬件SPI
21 #define SPI_SOFT         1 // 软件模拟SPI
22
23 #define SPI_4WIRE        0 // 4线SPI
24 #define SPI_3WIRE        1 // 3线SPI
25
26 #define SPI_IRQ_DIS      0 // 不使用硬件IRQ脚
27 #define SPI_IRQ_EN       1 // 使用硬件IRQ脚
28
29 #define SPI_SOFTWARE_CE  0 // 软件控制CE
30 #define SPI_HARDWARE_CE  1 // 硬件控制CE
31
32 #define SPI_PORT_SELECT  SPI_PORT1 // SPI端口选择
33
34 #define SPI_HS_SELECT    SPI_SOFT // SPI是软件模拟还是硬件SPI
35
36 #if (SPI_HS_SELECT == SPI_HARD)
37 #define SPI_WIRE          SPI_4WIRE // 硬件SPI一定是4线的
38 #else
39 // #define SPI_WIRE          SPI_3WIRE
40 #define SPI_WIRE          SPI_4WIRE // 软件SPI可以是3线也可以是4线
41 #endif
42
43 #define SPI_IRQ           SPI_IRQ_DIS // 是否使用IRQ脚
44
45 #define SPI_CE            SPI_SOFTWARE_CE // 是否硬件控制CE
46
```

图表 1 HS6230 的 SPI 驱动配置

下图是 SPI 操作的接口函数，最终对外使用的是 SPI_SendByte 和 SPI_ReadByte 函数，这两个函数里面就区分了硬件 SPI 还是软件 SPI，是 3 线还是 4 线。

```

121
122 // 加入SPI操作中的延时是为了降低SPI的速率，尽量跟客户使用的MCU所能达到的SPI速率一致
123 #define SPI_DELAY() do{unsigned char tx = 0; \
124                     for(tx = 0; tx < 10; tx++) \
125                         __NOP(); \
126                     }while(0)
127
128 /* SPI初始化配置函数 */
129 void SPI_Configuration(void);
130
131 void SPI_ce_write(unsigned char ctrl);
132 void SPI_csn_write(unsigned char ctrl);
133
134 /* 调试的模块，3线SPI的时候有可能SDA脚是MOSI脚，也有可能是MISO脚，正常应该是MOSI脚 */
135 unsigned char SPI_mosi_read(void);
136 /* 调试的模块，3线SPI的时候有可能SDA脚是MOSI脚，也有可能是MISO脚，正常应该是MOSI脚 */
137 unsigned char SPI_miso_read(void);
138
139 #if (SPI_HS_SELECT == SPI_HARD)
140 /* 硬件4线SPI写入和读取函数 */
141 unsigned char hw_4wire_spi_wrd(unsigned char Data);
142 #else
143 #if (SPI_WIRE == SPI_4WIRE)
144 /* 软件模拟4线SPI写入和读取函数 */
145 unsigned char sw_4wire_spi_wrd(unsigned char Data);
146 #else
147 /* *****SPI端口的读写操作函数，根据实际情况选择移植对应的函数即可***** */
148 /* 如果是使用3线制模拟SPI，则移植下面这3个函数 */
149 /* 切换IO口输入输出模式的操作 */
150 void sw_3wire_gpio_Switch_Dir(unsigned int GPIO_Pinx, unsigned char GPIO_mode);
151 /* 软件模拟3线SPI的时候写入和读取是分开操作的，因为要切换MOSI脚的输入输出模式。 */
152 /* 软件模拟3线SPI写入1字节数据 */
153 unsigned char sw_3wire_sendbyte(unsigned char Data);
154 /* 软件模拟3线SPI读取1字节数据 */
155 unsigned char sw_3wire_readbyte(unsigned char Data);
156 /* *****SPI端口的读写操作函数，根据实际情况选择移植对应的函数即可***** */
157 #endif
158 #endif
159
160 /* SPI写入一个字节内容 */
161 unsigned char SPI_SendByte(unsigned char data);
162 /* SPI读取一个字节内容 */
163 unsigned char SPI_ReadByte(unsigned char data);
164

```

图表 2 SPI 操作的接口函数

```

bsp_spi.c
294
295 unsigned char SPI_SendByte(unsigned char data)
296 {
297     #if (SPI_HS_SELECT == SPI_HARD)
298         hw_4wire_spi_wrd(data);
299     #else
300     #if (SPI_WIRE == SPI_4WIRE)
301         sw_4wire_spi_wrd(data);
302     #else
303         sw_3wire_sendbyte(data);
304     #endif
305     #endif
306 }
307
308 unsigned char SPI_ReadByte(unsigned char data)
309 {
310     #if (SPI_HS_SELECT == SPI_HARD)
311         return hw_4wire_spi_wrd(data);
312     #else
313     #if (SPI_WIRE == SPI_4WIRE)
314         return sw_4wire_spi_wrd(data);
315     #else
316         return sw_3wire_readbyte(data);
317     #endif
318     #endif
319 }

```

图表 3 封装起来后的对外 SPI 读写函数

客户移植的时候主要是针对下面这些函数进行移植，这些函数也是真正跟底层有关的。

1. SPI_Configuration() 初始化配置 SPI，不管是硬件还是模拟的 IO 口的初始化配置。
2. SPI_ce_write() 操作 CE 拉高拉低的函数，不管是硬件控制还是寄存器控制。
3. SPI_csn_write() 操作 CSN 拉高拉低的函数。
4. SPI_sck_write() 操作 SCK 拉高拉低的函数。
5. SPI_mosi_write() 模拟 SPI 的时候 MOSI 拉高拉低的操作。
6. SPI_mosi_read() 和 SPI_miso_read() 模拟 4 线和 3 线 SPI 的时候数据输入脚的信号读取函数。
7. hw_4wire_spi_wrd() 硬件 SPI 时的 SPI 读写操作函数。
8. sw_4wire_spi_wrd() 软件模拟 4 线 SPI 时的读写操作。
9. sw_3wire_sendbyte() 3 线模拟 SPI 时的写函数。
10. sw_3wire_readbyte() 3 线模拟 SPI 时的读函数。
11. sw_3wire_gpio_Switch_Dir() 3 线模拟 SPI 是 SDIO 脚的输入输出配置切换函数。

3. HS6230 读写 RF 寄存器的操作

bsp_spi_rf.c/h 这 2 个文件是最基本的操作 RF 的 SPI 读写寄存器和写命令的操作接口，要操作 RF，都是要基于这些函数的基础上去实现的。下面是接口函数的介绍。

1. `SPI_Operation()` 这个函数是写命令的函数，像清 `fifo` 这样的操作，就是在 `CSN` 拉低拉高的过程中值里面写一个字节。
2. `SPI_wr_cmd()` 这个函数是在切换 `bank` 的时候用的，主要是这个函数的操作不用或上 `0x20` 这个写寄存器的命令。
3. `SPI_write_byte()` 和 `SPI_wr_buffer()` 这 2 个函数是写 1 个字节和写多个字节操作的函数。
4. `SPI_read_byte()` 和 `SPI_read_buffer()` 这 2 个函数是读 1 个字节和读多个字节操作的函数。

4. HS6230 操作 RF 函数接口介绍

`bsp_rf.c/h` 这 2 个文件是 RF 操作的接口，包括初始化 RF，`bank` 切换，切换通信地址，切换发射功率，切换通信频点，设置晶体负载电容值，开启/关闭数据白化功能，获取芯片 `chip ID`，切换 RF 工作模式，清状态寄存器，清 `TX FIFO/RX FIFO`，批量配置寄存器，发送数据包，读取数据包，进入和退出睡眠的操作，打印 RF 所有寄存器配置等等。详细的可以看 `bsp_rf.h`。

```

bsp_rf.h
98
99 /* 切换寄存器Bank */
100 void HS6230_bank_Switch(RF_Bank_TypeDef bank);
101 /* 设置CE为高 */
102 void HS6230_CE_High(void);
103 /* 设置CE为低 */
104 void HS6230_CE_Low(void);
105 /* 设置CE脉冲 至少得时20us时间 */
106 void HS6230_CE_Pulse(void);
107 /* 读状态寄存器 */
108 unsigned char HS6230_Read_Status(void);
109 /* 切换RF通信地址 */
110 void HS6230_ChangeAddr_Reg(unsigned char* AddrBuf, unsigned char len);
111 /* 切换RF输出功率 */
112 void HS6230_Change_Pwr(RF_Pwr_TypeDef Pwr_lev);
113 /* 切换RF通信频点 */
114 void HS6230_Change_CH(unsigned char ch_index);
115 /* 切换晶体负载电容值 */
116 void HS6230_Change_Xtalcc(unsigned char xtal_cc);
117 /* 使能白化 */
118 void HS6230_Enable_Scramble(unsigned char en_flag);
119 /* 读取RF芯片的ID */
120 unsigned char HS6230_Get_Chip_ID(void);
121 /* RF工作模式切换 */
122 void HS6230_ModeSwitch(RF_ModeTypeDef mod);
123 /* 清除状态寄存器的标志位, 把status(0x07)寄存器的标志位都清掉 */
124 void HS6230_Clear_All_Irq(void);
125 /* 清空TX_FIFO */
126 void HS6230_Flush_Tx(void);
127 /* TX_FIFO的内容重用, 要使用这个数据包重用功能, 要先在写TX_FIFO前写这个命令
128 ** 然后CE一直为高就会一直往外发相同的数据; 如果时CE脉冲方式, 则一个脉冲就
129 ** 发送区保相同的数据;
130 ** 需要注意的时, 使用这个命令, 不能情况tx_fifo, 也不能切换信道, 否则tx_fifo的内容就没有了。*/
131 void HS6230_R reuse_TX_PL(void);
132 /* RF发送数据包函数
133 buf:待发送的数据包数组;
134 len:待发送的数据包内容长度[1~32]字节;
135 cmd:发送数据包的命令, RF_W_TX_PAYLOAD, RF_W_TX_PAYLOAD_NOACK */
136 void HS6230_SendPack(unsigned char cmd, unsigned char* buf, unsigned char len);
137 /* 芯片进入低功耗模式 */
138 void HS6230_Enter_sleep(void);
139 /* 芯片退出低功耗模式 */
140 void HS6230_Leave_sleep(void);
141 /* 软件复位RF */
142 void HS6230_Soft_Rst(void);
143 /* 初始化RF */
144 void HS6230_Init(void);
145 /* 打印RF所有寄存器函数, 调试的时候可以使用, 产品上不需要移植这个 */
146 void HS6230_Dump_RF_Register(void);

```

图表 4 操作 RF 的接口函数

这里还需要注意一下的是 RF 初始化的函数, 下图框出来的地方, 软件复位初始化 RF 之后, 如果等了 35ms 都没有查询到校准完成标志, 则需要重新复位初始化 RF 避免因为 SPI 时序的问题导致 RF 初始化不成功, 从而死在等校准标志的地方。另外一处是校准完成之后, 要把 bank1 的 HS6230_BANK1_DAC_RANGE 和 HS6230_BANK1_PLL_CTL0 值读取出来, 然后再写回去寄存器里面, 这样 RF 芯片睡眠的时候校准数据就不会丢。

```

bsp_rtc
298 /* 初始化RF */
299 void HS6230_Init(void)
300 {
301     unsigned char temp[5];
302     unsigned char regval;
303
304     LOOP:
305     HS6230_CE_Low();
306     SPI_write_byte(HS6230_BANK0_CONFIG, 0x02); // 先断电, 然后在等待一下再开始等待校准
307     delay_ms(4);
308
309     printf("start init HS6230.\n");
310
311     HS6230_Soft_Rst();
312
313     HS6230_CE_High();
314     delay_us(100);
315     HS6230_CE_Low();
316
317     run_time = 0;
318     while((SPI_read_byte(HS6230_BANK0_SETUP_VALUE) & 0x01) == 0x00) // 等待芯片校准结束
319     {
320         if(run_time > 35)
321             goto LOOP; // 若果等待35ms都没有检测到校准完成标志, 则重新复位初始化RF, 避免SPI有问题导致RF初始化有问题
322     }
323
324     /******保存校准值, 主要是校准值如果不写入寄存器的话, 那么睡眠醒来之后校准值会丢******/
325     HS6230_bank_Switch(Rf_Bank1);
326
327     regval = SPI_read_byte(HS6230_BANK1_DAC_RANGE);
328
329     SPI_read_buffer(HS6230_BANK1_PLL_CTL0, temp, 4);
330     temp[0] = 0x07;
331     temp[1] = 0x80;
332     temp[2] |= 0x48; // (BIT6+BIT3);
333     SPI_wr_buffer(HS6230_BANK1_PLL_CTL0, temp, 4);
334
335     SPI_write_byte(HS6230_BANK1_DAC_RANGE, regval);
336     /******保存校准值, 主要是校准值如果不写入寄存器的话, 那么睡眠醒来之后校准值会丢******/
337
338     HS6230_bank_Switch(Rf_Bank0);
339
340     SPI_write_byte(HS6230_BANK0_SETUP_VALUE, 0x01);
341
342     HS6230_Enable_Scramble(1);
343
344     HS6230_Change_Xtalcc(0x0f);
345
346     HS6230_Clear_All_Irq();
347     HS6230_Flush_Tx();
348
349     HS6230_ModeSwitch(Rf_PTX_Mode);
350
351     // HS6230_CE_High();
352     printf("init HS6230 ok.\n");
353 }

```

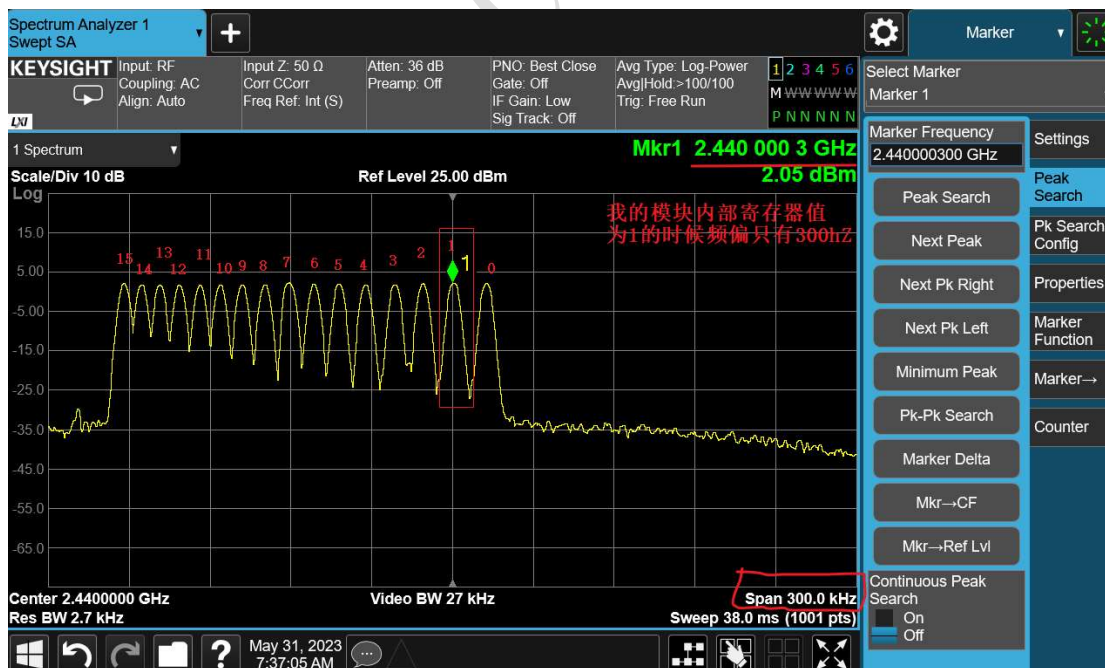
图表 5 HS6230 初始化校准函数

5. HS6230 的 APP 测试程序

从 main 函数初始化完之后, 所以的 SDK 都是按照 test_app();的入口进入测试程序的。由于 HS6230 只是单发的芯片, 所以只有发送的测试程序。发送程序里面有固定 32 字节的发送和动态数据 1~32 字节的发送。还有载波测试的程序。当然 BLE 的工程里面有发 BLE 广播包的程序。Other 的测试工程里面还有像测试睡眠功耗; 获取按键值; 按键唤醒 MCU, 数据包重用等的测试例程。

5.1 HS6230 载波测试

载波测试可以看出 RF 射频信号的频偏，调制信号看不出频偏。1M 速率的时候频偏在 $\pm 160\text{k}$ 范围内是可以的，当然频偏越小越好。注意做为 BLE 广播的时候，频偏最好控制在 $\pm 30\text{k}$ 以内。控制 HS6230 发送载波的位是在 RF_SETUP 这个寄存器的 bit7，这个位设置为高，CE 拉高就发载波，CE 变低了就没有载波。然后频点根据客户需要去设置，可以通过调整 HS6230 内部寄存器来调整频偏。当然内部寄存器可调整的范围有限，不是并不是所有的板子或者说晶振都可以通过调整内部寄存器来把频偏调整回来，这个跟硬件关系很大，确实调不过来的只能在外部加晶体试试或者是换晶体试试。内部寄存器的值是从 0 到 15 可调。注意看频偏的时候频谱仪的 SPAN 要打到百 k 级，比如 200k，这样看频偏才准。当然一开始测试的时候如果是打到百 k 级的 SPAN，可能你会看不到信号可以先设置到 10M，去抓到频点的 peak 之后再设置中心频率到 peak 点，然后再设置 SPAN 到百 k 级。从下图中可以看出，数值越大频率越往小走，数值越小频率越往大走，然后步进值不是等幅步进的，数值从小到大变化的时候步进幅度越来越小。下面的测试是循环修改晶体负载电容值测出来的，实际如果固定一个频点的话，是只有一个载波信号出来的。从下图中可以看出，当设置寄存器值为 1 的时候测的频偏最小，只偏了 300Hz。



图表 6 HS6230 载波测试图片

```

101  /* 载波测试程序 */
102  void HS6230_Wave_Test(void)
103  {
104      unsigned char i;
105
106      printf("RF Carrier start!\n");
107
108      HS6230_Init(); // RF初始化
109
110      // HS6230_Dump_RF_Register();
111
112      HS6230_CE_Low();
113      HS6230_Change_Pwr(pwr_level 15);
114      // 调整晶体负载电容值, 以调整频偏
115      HS6230_Change_Xtalcc(1);
116      // 设置频点
117      HS6230_Change_CH(40);
118      // 选择工作模式为载波模式
119      HS6230_ModeSwitch(Rf_Carrier_Mode);
120      HS6230_CE_High();
121
122      while(1)
123      {
124          for(i = 0; i < 16; i++)
125          {
126              HS6230_Change_Xtalcc(i);
127              delay_ms(1000);
128          }
129      }
130
131      HS6230_CE_Low();
132  }

```

图表 7 HS6230 载波测试代码

5.2 HS6230 睡眠功耗测试

下面是测试睡眠功耗的代码，代码里面也有注释说明，比较一目了然。

注意：在睡眠的时候如果不用按键，要记得关闭，要不然会有漏电。当然如果在睡眠的时候需要 HS6230 的按键唤醒 MCU，那就不能关闭按键检测功能，否则就没有按键功能自然也就没办法唤醒 MCU 了。

```

test_app.c
134  /* 睡眠测试程序 */
135  void HS6230_Sleep_Test(void)
136  {
137      unsigned char i = 0;
138
139      printf("HS6230 Sleep start!\n");
140
141      HS6230_Init(); // RF初始化
142
143      // HS6230_Dump_RF_Register();
144
145      HS6230_CE_Low();
146      HS6230_ModeSwitch(Rf_PTX_Mode);
147      HS6230_Change_Pwr(pwr_level_15);
148      // 选择工作模式
149      HS6230_Change_CH(40);
150
151      // SETUP_VALUE寄存器的bit1是不使能按键检测, 避免漏电
152      i = SPI_read_byte(HS6230_BANK0_SETUP_VALUE);
153      i &= ~0x02;
154      SPI_write_byte(HS6230_BANK0_SETUP_VALUE, i);
155
156      while(1)
157      {
158          HS6230_Enter_sleep(); // 进入睡眠
159
160          delay_ms(1000);
161          delay_ms(1000);
162          delay_ms(1000); // 延时3000ms, 方便观察睡眠电流
163
164          HS6230_Leave_sleep(); // 退出睡眠
165
166          // 发送数据, 用于观察退出睡眠之后的工作电流情况, 4ms发一包, 10个字节。
167          for(i = 0; i < 250; i++)
168          {
169              HS6230_Flush_Tx();
170              HS6230_Clear_All_Irq();
171
172              HS6230_SendPack(HS6230_W_TX_PAYLOAD_NOACK, rf_tx_data, 10); /*write TX_FIFO*/
173
174              HS6230_CE_Pulse();
175
176              delay_ms(4);
177          }
178      }
179
180      HS6230_CE_Low();
181  }

```

图表 8 HS6230 测试睡眠功耗的代码

5.3 HS6230 按键值读取

按键的原理图如下图所示, 从下面的原理图可以看出小车的前进和后退按键再 key1 脚上, 左转和右转在 key2 脚上, 因为前进和后退按键不会同时按下(即使是同时按下那也是前进优先), 同理左转和右转按键也不会同时按下(即使是同时按下也是左转优先)。剩下的 F1, F2, 和另外 2 个按键也都分配在不同的脚, 就是避免同时按下的情况。要使用 HS6230 的按键检测功能, 需要在程序上去打开按键检测。

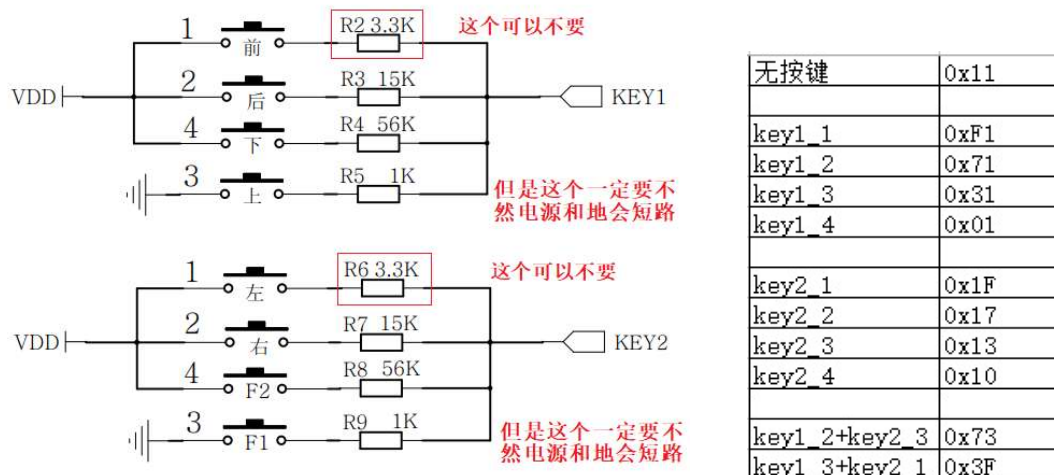


图 9 HS6230 按键原理图和按键真值表

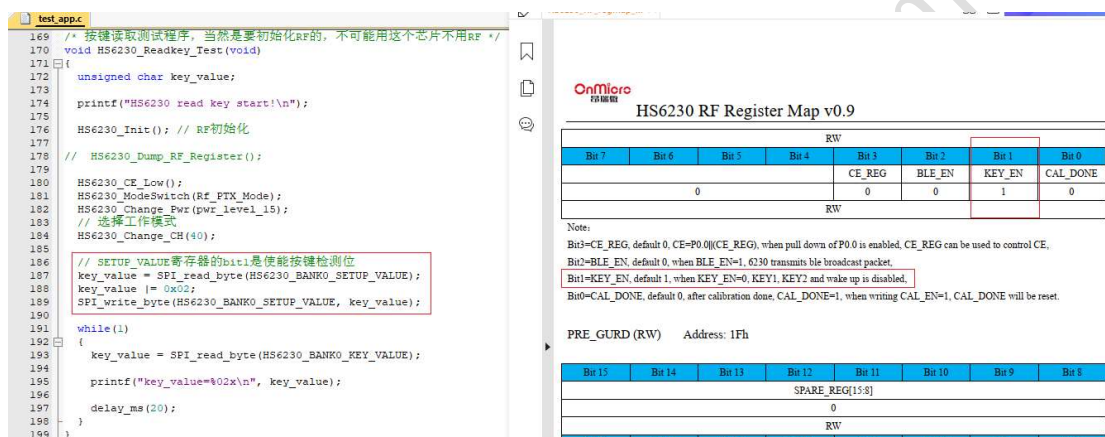
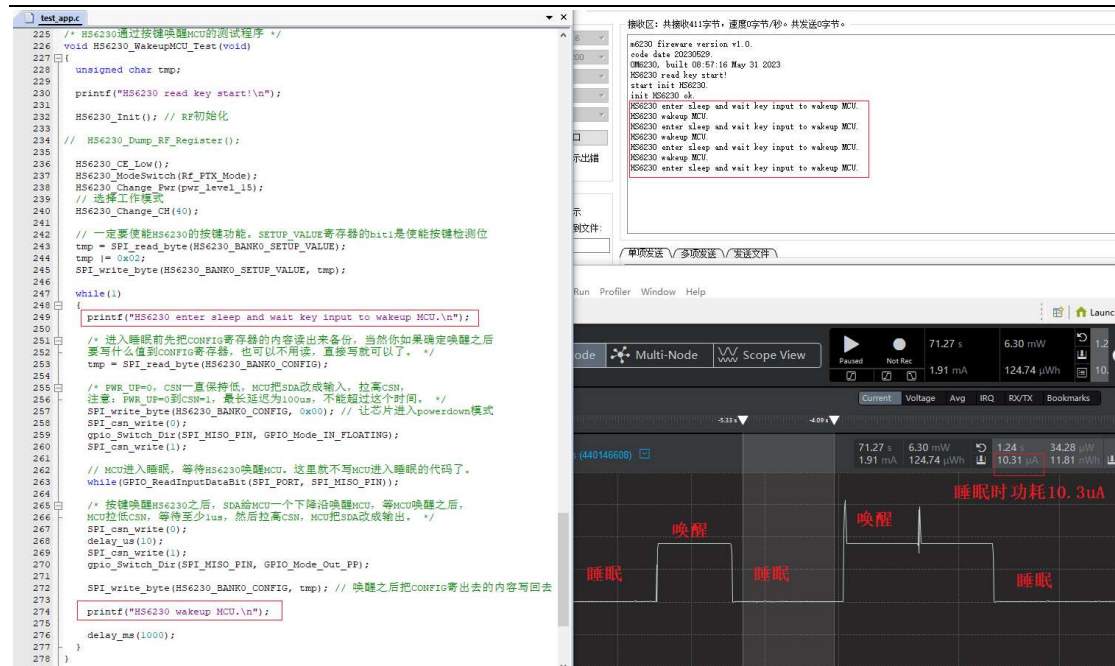


图 10 HS6230 的按键检测功能测试代码

5.4 HS6230 按键唤醒 MCU

HS6230 做按键唤醒 MCU, 一定要使能 HS6230 的按键检测功能然后设置步骤按照代码里面的注释说明操作, 时序上面要满足要求, 否则无法达到预定要求。



图表 11 HS6230 按键唤醒 MCU 的操作

5.5 HS6230 发送数据包的操作

HS6230 发送数据的操作，这里面有设置数据包长度，设置通信地址，设置通信频点，设置发射功率，写数据包内容，激活发送，查询是否发送完成等操作。

正确的发数据包的操作流程是：CE 拉低，要切换频点，或者切换地址，或者切换功率等的操作都操作完成，之后清 FIFO，清状态寄存器，然后写 TX FIFO，然后 CE 脉冲(CE 高电平至少 20us)，或者是拉高 CE 激活发送(发送完成之后再拉低 CE)，可以等待 1ms 时间再去查状态寄存器，或者是一直去查状态寄存器看看是否发送完成。

```

test_app.c
36 void HS6230_TX_Test(unsigned char dynpd)
37 {
38     unsigned char sta, tx_len, rx_len, i;
39
40     printf("TX Test start!\n");
41
42     HS6230_Init(); // RF初始化
43
44     HS6230_CE_Low(); // 拉低CE
45     SPI_write_byte(HS6230_BANK0_RX_PW_P0, 0x20); // 设置通信数据包长度
46     if(0 == dynpd) // 如果不使能动态数据包长度,
47         SPI_write_byte(HS6230_BANK0_FEATURE, 0x00);
48     else if(1 == dynpd) // 如果使能动态数据包长度,
49         SPI_write_byte(HS6230_BANK0_FEATURE, 0x04);
50     SPI_write_byte(HS6230_BANK0_CONFIG, 0x0f);
51     HS6230_ChangeAddr_Reg(com_addr_table, sizeof(com_addr_table)); // 设置通信地址
52     HS6230_Change_CH(test_ch); // 设置通信频点
53     HS6230_Change_Pwr(pwr_level_15); // 设置发射功率
54
55     // 选择工作模式为发送模式
56     HS6230_ModeSwitch(Rf_PTX_Mode); // 设置工作模式为发送模式
57     HS6230_Flush_Tx(); // 清空TX FIFO
58     HS6230_Clear_All_Irq(); // 清状态寄存器
59
60     // HS6230_Dump_RF_Register(); // 打印RF所有寄存器的内容
61
62     while(1)
63     {
64         if(1 == dynpd) // 如果是要做动态数据包长度测试
65         {
66             tx_len ++;
67             if(tx_len > 32)
68                 tx_len = 1; // 至少得发1个字节的数据, 最多不超过32字节数据
69         }
70         else
71             tx_len = 32;
72
73         HS6230_SendPack(HS6230_W_TX_PAYLOAD_NOACK, rf_tx_data, tx_len); // 写TX_FIFO数据
74         rf_tx_data[0] ++;
75
76         HS6230_CE_High(); // 拉高CE启动数据发送
77         delay_ms(1); // 延时1ms的作用是等RF完成数据发送, 当然实际是不需要那么长时间的。
78         HS6230_CE_Low();
79
80         sta = HS6230_Read_Status(); // 读状态寄存器看是否发送完成
81         if(HS6230_STATUS_TX_DS & sta)
82         {
83             printf("tx ds!\n");
84         }
85         else if(HS6230_STATUS_MAX_RT & sta)
86         {
87             printf("max rt!\n");
88         }
89
90         delay_ms(10);
91     }
92 }

```

图表 12 HS6230 发送数据包的操作

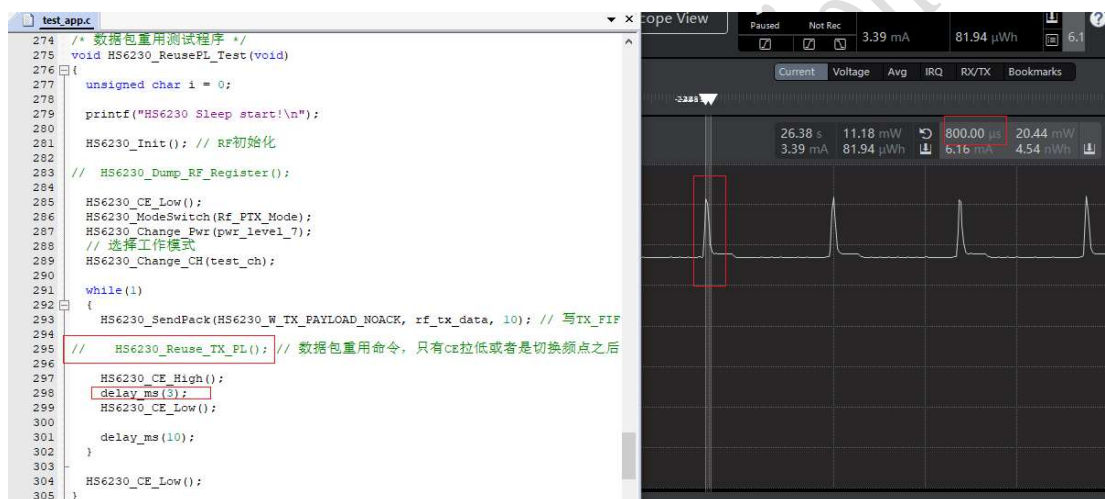
5.6 HS6230 发送数据包重用功能

HS6230 的数据包是可以设置重用的, 也就是说相同的数据包可以不用重复写入 TX FIFO。只要不拉低 CE, 不清空 FIFO, 数据包就一直在发, 而且是不需要重新建立射频锁定的。需要注意的是要先写 FIFO, 近跟着是写 REUSE_TX_PL(0xE3)命令, 然后拉高 CE 启动发送, 直到拉

低 CE 才停止发送。



图表 13 HS6230 数据包重用命令



图表 14 不使用数据包重用命令

从上面两个图中可以看出使用数据包重用和不使用数据包重用, 发射数据的时候的电流波形持续时间是不同的, 而且数据包重用的电流波形是一直平稳的, 说明数据发完之后又继续发不停歇。然后不使用数据包重用命令发数据的时候, 实际上发完数据, 芯片就进入 standby 模式了, 不管此时的 CE 有没有拉低下来。

```

test_app.c
274  /* 数据包重用测试程序 */
275  void HS6230_ReusePL_Test(void)
276  {
277      unsigned char i = 0;
278
279      printf("HS6230 Sleep start!\n");
280
281      HS6230_Init(); // RF初始化
282
283      // HS6230_Dump_RF_Register();
284
285      HS6230_CE_Low();
286      HS6230_ModeSwitch(Rf_PTX_Mode);
287      HS6230_Change_Pwr(pwr_level_7);
288      // 选择工作模式
289      HS6230_Change_CH(test_ch);
290
291      while(1)
292      {
293          HS6230_SendPack(HS6230_W_TX_PAYLOAD_NOACK, rf_tx_data, 10); // 写TX_FIFO数据
294
295          HS6230_Reuse_TX_PL(); // 数据包重用命令, 只有CE拉低或者是切换频点之后才停止发数据
296
297          HS6230_CE_High();
298          delay_ms(3);
299          HS6230_CE_Low();
300
301          delay_ms(10);
302      }
303
304      HS6230_CE_Low();
305  }

```

图表 15 HS6230 数据包重用命令的测试代码

5.7 HS6230 切换地址，切换功率，切换频点，发送数据包的操作

下图中的操作涉及到切换通信地址，通信频点，发射功率。是在做产品是用到的常规操作。

```

test_app.c
274  /* 切换地址, 切换功率, 切换频点发数据测试程序 */
275  void HS6230_Change_addr_pwr_ch_Test(void)
276  {
277      unsigned char ch_index = 0;
278      unsigned char test_addr_table[5] = {0x46, 0x0b, 0xaf, 0x43, 0x98};
279
280      printf("HS6230 Change addr pwr ch test!\n");
281
282      HS6230_Init(); // RF初始化
283
284      // HS6230_Dump_RF_Register();
285
286      HS6230_CE_Low();
287      // 选择工作模式
288      HS6230_ModeSwitch(Rf_PTX_Mode);
289
290      while(1)
291      {
292          if(ch_index == 0)
293              HS6230_ChangeAddr_Reg(com_addr_table, sizeof(com_addr_table)); // 设置通信地址
294          else
295              HS6230_ChangeAddr_Reg(test_addr_table, sizeof(test_addr_table)); // 设置通信地址
296
297          HS6230_Change_CH(test_ch); // 切换通信频点
298
299          HS6230_Change_Pwr(pwr_level_15); // 切换最大发射功率
300          HS6230_Flush_Tx(); // 清空TX FIFO
301          HS6230_SendPack(HS6230_W_TX_PAYLOAD_NOACK, rf_tx_data, 10); // 写TX_FIFO数据
302          HS6230_CE_Pulse(); // CE脉冲激活发送
303          while(!(HS6230_STATUS_TX_DS & HS6230_Read_Status())); // 等待数据发送完成
304
305          HS6230_Change_Pwr(pwr_level_7); // 切换中间发射功率
306          HS6230_Flush_Tx(); // 清空TX FIFO
307          HS6230_SendPack(HS6230_W_TX_PAYLOAD_NOACK, rf_tx_data, 10); // 写TX_FIFO数据
308          HS6230_CE_Pulse(); // CE脉冲激活发送
309          while(!(HS6230_STATUS_TX_DS & HS6230_Read_Status())); // 等待数据发送完成
310
311          (ch_index > 3)?ch_index = 0:ch_index ++; // 频点计数器自增
312
313          delay_ms(10);
314      }
315  }

```

图 16 HS6230 切换地址, 功率, 频点, 发数据包的操作

6. HS6230 通信频点的要求

昂瑞微的 2.4G RF 对通信频点有要求, 使用的是 16M 的晶体, 所以设置通信频点的时候 16 的整数倍的频点最好是避开不使用, 因为会影响接收的灵敏度, 有几个 dB 的差异。16M 的整数倍频点, 2400, 2416, 2432, 2448, 2464, 2480 这些频点接收灵敏度差一些。

7. HS6230 通信地址的要求

昂瑞微的 2.4G RF 的通信地址有要求。

1. 不能出现连续的 6 个 bit 的 0 或者是连续的 6 个 bit 的 1, 像: 0xc0(1100 0000); 0x58, 0x17(0101 1000 0001 0111); 0x4f, 0xc3(0100 1111 1100 0011), 这样的地址就是不对的。

2. 也不能出现连续的 24bit 的 0101 或者 1010 这样的地址，像：0xaa,0xaa,0xaa(1010 1010 1010 1010 1010 1010)这样的地址是不对的。
3. 也不能把地址的这几个字节都设置成一样，比如 5 个字节都设置成 0xbc,0xbc,0xbc,0xbc,0xbc 这样的地址。

8. HS6230 硬件设置

1. HS6230 硬件设计上只需要电源处的 1uF~10uF +0.1uF 的 2 个电容挂在电源脚上。
2. 16M, ± 60 ppm, 12pF 以下的晶体。
3. 天线匹配处如果不考虑过认证，串一个 1pf 左右的电容，避免由于焊接天线的烙铁漏电导致烧坏 RF 芯片的天线脚。
4. 如果使用大功率发送的时候，天线处要挂一个 1pf 左右的电容到地。
5. 如果要过认证，需要预留 π 型天线匹配电路。
6. SPI 接口根据 RF 的打线决定是 3 线 SPI 还是 4 线 SPI。
7. 如果有条件的话，可以把 IRQ 脚打出来，这样可以通过 IRQ 脚的中断信号及时处理 RF 的发送完成与否的处理。
8. 如果使用 HS6230 的按键检测功能，一定要注意按键的接法，避免按键同时按下的时候造成电源地短路，还要注意组合按键的优先级，而且接地的按键按下的时候组合按键优先级可能会出现反转的情况，取决于电平落在哪个电压范围。

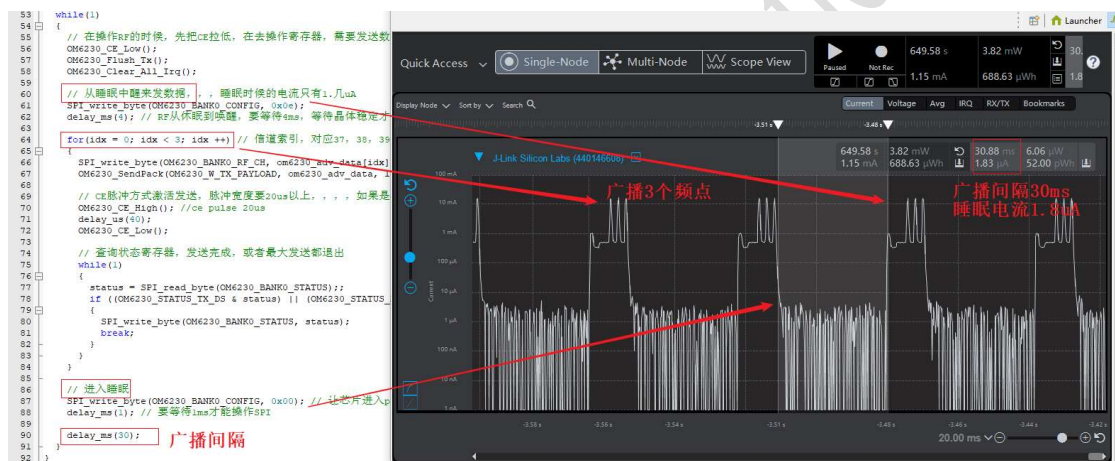
9. HS6230 功耗数据

HS6230 不使用内部按键时 2.4G 模式的睡眠功耗



图表 17 HS6230 不使用内部按键时 2.4G 模式的睡眠功耗

HS6230 广播间隔 30ms 的功耗



图表 18 HS6230 广播间隔 30ms 的功耗