

HS6220 应用手册(初级应用篇)

V1.4

2021/11/16

修改历史

版本	历史版本	日期	作者
V1.0	初始版本	2020-11-10	钟 科
V1.1	修改了初始化配置	2020-12-01	钟 科
V1.2	增加了 6200 互通的配置	2020-12-10	钟 科
V1.3	修改了 6220 初始化配置	2021-03-25	钟 科
V1.4	添加天线端使用注意事项	2021-11-16	钟 科

目录

修改历史.....	1
图表目录.....	4
表格目录.....	5
1. 概述.....	6
2. HS6220-2.4G 应用初始化简介.....	7
3.HS6220 普通 2.4G 模式.....	7
3.1 2.4G 模式初始化配置流程.....	7
3.2 2.4G 使用时常用寄存器配置.....	9
3.3 2.4G 配置需要注意的一些事项.....	9
3.3.1 软件复位.....	9
3.3.2 CONFIG 寄存器.....	10
3.3.3 数据白化功能.....	10
3.3.4.FEATURE 寄存器.....	11
3.3.5.DYNPD 寄存器.....	12
3.3.6.接收端防死机处理.....	13
3.3.7.接收端接收处理数据的正确顺序.....	13
3.3.8.发送端正确的发送数据操作.....	14
3.3.9 IRQ 脚的输出控制功能.....	14
3.4 常用寄存器的配置介绍.....	15
3.4.1 CONFIG(BANK0--00)寄存器.....	15
3.4.2 EN_AA(BANK0--01)寄存器.....	15
3.4.3 EN_RXADDR(BANK0--02) 寄存器.....	15
3.4.4 PMU_CTL(BANK0--03)寄存器.....	16
3.4.5 SETUP_RETR(BANK0--04)寄存器.....	16
3.4.6 RF_CH(BANK0--05)寄存器.....	16
3.4.7 RF_SETUP(BANK0--06)寄存器.....	17
3.4.8 STATUS(BANK0--07)寄存器.....	17
3.4.9 RX_ADDR_P0(BANK0--0A)寄存器.....	18
3.4.10 TX_ADDR(BANK0--10)寄存器.....	18
3.4.11 RX_PW_P0(BANK0--11)寄存器.....	18
3.4.12 FIFO_STATUS(BANK0--17)寄存器.....	18
3.4.13 DYNPD(BANK0--1C)寄存器.....	18
3.4.14 FEATURE(BANK0--1D)寄存器.....	19
4.6200 和 6220 互通的配置.....	20
4.HS6220 使用注意事项.....	23
4.1 天线端防止烙铁漏电烧坏处理.....	23
4.2 晶体负载电容.....	23
4.3 SPI 接口电平.....	23
4.4 通信地址要求.....	23

OnMicro Confidential

图表目录

图表 1	2.4G 模式初始化流程.....	8
图表 2	2.4G 应用时寄存器配置.....	9
图表 3	软件复位.....	10
图表 4	6220 和 6200 CONFIG 寄存器.....	10
图表 5	白化开关.....	11
图表 6	FEATURE 寄存器.....	12
图表 7	DYNPD 寄存器.....	12
图表 8	接收端防死机处理.....	13
图表 9	接收端操作流程.....	13
图表 10	发送端操作流程.....	14
图表 11	IRQ 脚输出控制操作.....	14
图表 12	CONFIG 寄存器介绍.....	15
图表 13	EN_AA 寄存器介绍.....	15
图表 14	EN_RXADDR 寄存器介绍.....	16
图表 15	PMU_CTL 寄存器介绍.....	16
图表 16	SETUP_RETR 寄存器介绍.....	16
图表 17	RF_CH 寄存器介绍.....	17
图表 18	RF_SETUP 寄存器介绍.....	17
图表 19	STATUS 寄存器介绍.....	17
图表 20	FIFO_STATUS 寄存器介绍.....	18
图表 21	DYNPD 寄存器介绍.....	19
图表 22	FEATURE 寄存器介绍.....	19
图表 23	HS6200 初始化配置.....	20
图表 24	HS6220 初始化配置.....	21
图表 25	6220 收, 6200 发, 没有 ACK 的应用配置.....	22
图表 26	HS6220 发送, HS6200 接收, 没有 ACK 的应用配置.....	22
图表 27	HS6220 发送, HS6200 接收, 有 ACK 的应用配置.....	22
图表 28	HS6220 接收, HS6200 发送, 有 ACK 功能的应用配置.....	23

表格目录

表 1-1 xxxx.....	6
表 1-2 xxxx.....	6

OnMicro Confidential

1. 概述

HS6220 是北京昂瑞微电子有限公司(以下简称“昂瑞微”)2020 年最新推出的一款 2.4G 收发芯片。是在原来 HS6200 的基础上的升级版本，加入了一些新的功能。这个使用手册是针对做玩具的客户应用或者是普通的 2.4G 的应用场景所写，所以这里只介绍 2.4G 的基本功能。

这款 HS6220 的芯片是可以和昂瑞微公司之前所出的所有的 2.4G 芯片互通的，不存在不兼容的问题，只需要配置好相应的寄存器。SDK 中会提供几个配套的应用配置，用户可以根据参考配置，然后配置成自己产品中实际使用的配置，如果不会配置，那么请详细阅读此文档来进行配置。这个芯片有很多的新功能和新特点，这里也主要是讲玩具和其他 2.4G 普通应用的特点，主要的新特点有以下几点：

1. 可以做 3 线或者 4 线 SPI 通信，只需要配置一下寄存器。
2. 去掉了原来的 CE 硬件脚，改用寄存器控制或者是命令控制的形式。
3. IRQ 引脚可以通过 SPI 来控制输出，驱动马达或者 LED 灯。
4. 简化初始化流程。

2. HS6220-2.4G 应用初始化简介

HS6220 初始化流程较 HS6200 的初始化流程，已经精简了不少。由于现在客户手上的样片，有些寄存器值还未确定，所以需要在代码里面配置，后面出来的芯片，这些标注的寄存器默认是不需要配置的了。所以下面的初始化流程的介绍，也是在后面的芯片情况下介绍的初始化流程。大致可以这么概括：

- 1.上电延时 10ms。
- 2.进行一次软件复位是芯片确保在默认寄存器值下。
- 3.关掉 IO 复用功能，配置 SPI 线数。
- 4.给芯片上电，让晶振起振。
- 5.CE 脉冲(至少 40us)。
- 6.等待 RF 校准完成。
- 7.配置用户要使用的其他寄存器。

3.HS6220 普通 2.4G 模式

3.1 2.4G 模式初始化配置流程

下面的截图是 SDK 中的初始化过程，请按照我们 SDK 给的初始化流程走，初始化完成之后再根据您的需要进行其他的配置。要注意，DYNPD 寄存器的选择 SPI 线数的位，不要再应用配置中改掉了，还有 FEATRUE 寄存器的 bit4，CONFIG 寄存器的 bit7 一定要注意置”1”。


```
void HS6220_Init(void)
{
    unsigned char temp[5];

    HS6220_write_byte(HS6220_BANK0_FEATURE, SOFT_RST); // soft_reset
    // 上电进行软件复位, 复位之后左右的寄存器恢复默认值

    #if (HS6220_SPI_NWIRE == SPI_4_WIRE)
        // 默认是3线SPI的, 如果要使用4线SPI, 则上电之后要设置一下
        // 如果要用6220的IRQ脚控制马达的话, 要设置IRQ脚, 而且必须是在最前面设置他
        HS6220_write_byte(HS6220_BANK0_DYNPD, 0x08); // 0x08 | 0x40; //
    #else
        // 根据配置是3线还是4线SPI
        // 如果要用6220的IRQ脚控制马达的话, 要设置IRQ脚, 而且必须是在最前面设置他
        HS6220_write_byte(HS6220_BANK0_DYNPD, 0x00); // 0x00 | 0x40; //
    #endif

    HS6220_CE_Low();
    HS6220_write_byte(HS6220_BANK0_CONFIG, 0x8b); // power up 给芯片上电
    delay_ms(3); // wait 3 ms 必须至少等待3ms
    HS6220_write_byte(HS6220_BANK0_PMU_CTL, 0xac); // HS6220_PWRDWN = 00
    delay_ms(2); // 必须至少等待2ms 是芯片处于工作模式

    HS6220_Bank_Switch(HS6220_Bank1);
    HS6220_write_byte(HS6220_BANK1_TEST_PKDET, 0x20); // pll_vdiv2_sel = 0

    /* increase filter agc threshold start */
    temp[0] = 0x01;
    HS6220_wr_buffer(HS6220_BANK1_FAGC_CTRL_1, temp, 1);

    HS6220_Bank_Switch(HS6220_Bank0);

    HS6220_CE_High(); // CE脉冲至少40us, 使芯片启动校准流程
    delay_us(100);
    HS6220_CE_Low(); // 一定要注意, 校准的时候CE是低的

    while((HS6220_read_byte(HS6220_BANK0_RF_SETUP) & 0x20) == 0x00); // wait cal done
    HS6220_write_byte(HS6220_BANK0_RF_SETUP, 0x40); // cal_en = 0 清除校准使能位

    HS6220_Bank_Switch(HS6220_Bank1);
    temp[2] = 0x75; // bp_dac = 1 bp_rc = 1
    temp[1] = 0x98; // bp_vco_amp = 1 bp_vco_ldo = 1
    temp[0] = 0x20;
    HS6220_wr_buffer(HS6220_BANK1_CAL_CTL, temp, 3);

    HS6220_Bank_Switch(HS6220_Bank0);
    temp[0] = 0x46;
    temp[1] = 0x0b;
    temp[2] = 0xaf; // 设置TX RX的通信地址, 这里属于用户应用配置的内容了。
    temp[3] = 0x43;
    temp[4] = 0x98;
    HS6220_wr_buffer(HS6220_BANK0_RX_ADDR_P0, temp, 5); // set address
    HS6220_wr_buffer(HS6220_BANK0_TX_ADDR, temp, 5); // set address

    HS6220_Clear_All_Irq(); // 这里可以在使用的时候去写, , 直到这里, 目前CE还是处于拉低的状态
    HS6220_Flush_Tx();
}
```

图表 1 2.4G 模式初始化流程

1. 芯片上电等待10ms, 然后进行软复位(Soft_rst)。
2. 在bank0的DYNPD寄存器中把Bypass_io置0, 关掉IO复用功能, 否则MOSI会有电平冲突; 如果需要4线SPI, 把Spi4_en置1, 如果需要3线SPI, Spi4_en用默认值0。
3. bank0的CONFIG寄存器中把PWR_UP置1, 打开晶振, 然后延时3ms让晶振启动。
4. bank0的PMU_CTL中把RF_PWRDWN置00, 让芯片进入工作模式, 延时2ms, 让数字开电。

5. bank0的FEATURE寄存器中把vco_amp_tx_mux置0。
6. bank1的TEST_PKDET中把pll_vdiv2_sel置00。
7. 拉CE pulse, CE pulse宽度大于40us, 然后在bank0的RF_SETUP中查询CAL_DONE, CAL_DONE=1表示校准完成。
8. bank0的RF_SETUP中把CAL_EN置0。
9. bank1的CAL_CTL寄存器配置为0x28 0x75 0x98 0x20 (高字节在前)。

3.2 2.4G 使用时常常用寄存器配置

```

/* 初始化 hs6220 模块, 注意初始化里面, ce已经是拉低来了的 */
HS6220_Init();
HS6220_write_byte(HS6220_BANK0_FEATURE, 0x10); // feature和dynamic寄存器要根据实际应用配置, 而且收发端要对应上, 否则会导致收到的数据不对
HS6220_write_byte(HS6220_BANK0_EN_AA, 0x00); // 这些设置都是在CE拉低的状态下进行的
HS6220_write_byte(HS6220_BANK0_CONFIG, 0xfa); // 是否使能自动应答
HS6220_write_byte(HS6220_BANK0_RX_PW_P0, 10); // 注意config寄存器的最高位是"1", 其他位根据需要配置, 不建议把中断映射到IRQ脚,
HS6220_write_byte(HS6220_BANK0_RF_CH, 5); // 0x02, 因为IRQ脚可能还要做输出脚
HS6220_write_byte(HS6220_BANK0_EN_RXADDR, 0x03); // 0x03, scramble_en = 0, 0x0b, scramble_en = 1
HS6220_Flush_Tx();
HS6220_Flush_Rx();
HS6220_Clear_All_Irq();

```

配置数据包长度, 如果是定长的话, 只能收发所设置的长度的数据包, 动长的话就1~32字节都可以
白化标志在这里

图表 2 2.4G 应用时寄存器配置

3.3 2.4G 配置需要注意的一些事项

3.3.1 软件复位

软件复位的寄存器已经更改了, 6220 是在 BANK0 的 FEATUE 寄存器的 bit5 这个位。而 6200 是在 BANK1 的 PLL_CTL1 的 bit31 这个位。

FEATURE (RW) Address: 1Dh

HS6220的复位标志位

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Long_packet_en	Ble_en	Soft_rst	BP_GAU	Vco_amp_tx_mux	EN_DPL	EN_ACK_PA_Y	EN_DYN_A CK
0	0	0	1	1	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

PLL_CTL1 (RW) Address: 02h

HS6200的软件复位在BANK1的PLL_CTL1的bit31

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Soft_rst	SYNC_DET_DIS	PHY_FPGA	BP_HF	DC_BW		DC_RM_EN	BP_GAU
0	0	0	0	0		0	0
RW	RW	RW	RW	RW		RW	RW
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
REG_TX_PA_WAIT							

图表 3 软件复位

3.3.2 CONFIG 寄存器

芯片的 CONFIG 寄存器的最高位要置”1”，最高位不置”1”的话不能进行收发数据。也就是说原来 CONFIG 寄存器设置发送模式 0x0E 现在要设置 0x8A,原来设置成接收模式 0x0F 的话。6220 没有配置 CRC 是 1 字节还是 2 字节的位，6200 需要把 CRC 设置成 2 字节。然后根据客户是否开了 CRC 来配置 CRC 控制位。

CONFIG (RW) Address: 00h

HS6220的CONFIG寄存器

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Tx_gurd_en	MASK_RX_DR	MASK_TX_DS	MASK_MAX_RT	EN_CRC	CE_REG	PWR_UP	PRIM_RX
1	0	0	0	1	0	0	1
RW	RW	RW	W	RW	RW	RW	RW

最高位要是”1”

是否使能CRC

是否寄存器控制CE，还是命令控制CE

CONFIG (RW) Address: 00h

HS6200的CONFIG寄存器

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	MASK_RX_DR	MASK_TX_DS	MASK_MAX_RT	EN_CRC	CRCO	PWR_UP	PRIM_RX
0	0	0	0	1	0	0	0
RW	RW	RW	W	RW	RW	RW	RW

是否开CRC

CRC要设置成2字节

图表 4 6220 和 6200 CONFIG 寄存器

3.3.3 数据白化功能

白化标志位，6220 是放在 BANK0 的 EN_RXADDR 的 bit3 这个位，而 6200 是放在 BANK1 的 CAL_CTL 的 bit20 这个位。就目前来说，6200 使用的都是开了白化功能的，所以 6220 配置的时候也要开白化功能。

EN_RXADDR (RW) Address: 02h

HS6220的白化是在BANK0的bit3这个位

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reversed				scramble_en	ERX_P2	ERX_P1	ERX_P0
0				1	0	1	1
RW				RW	RW	RW	RW

CAL_CTL (RW) Address: 03h

HS6200的白化功能在BANK1的CAL_CTL的bit20这个位

8'h40							
RW							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Bp_ch_chg	afc_w_sel		Scramble_en	Bp_dc	Bp_dac	Bp_afc	Bp_rc
0	0		1	0	1	0	1
RW	RW		RW	RW	RW	RW	RW
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

12、CE 拉低(需确保此时 HS6200 处于 BANK0)

13、切换至 BANK1

14、配置 BANK1 寄存器(配置如下)

寄存器地址	配置值	备注
0x01	1M: 0x40, 0x01, 0x30, 0xE1 2M: 0x40, 0x01, 0x30, 0xE2	HS6200_BANK1_PLL_CTL0
0x02	0x00, 0x42, 0x10, 0x07	HS6200_BANK1_PLL_CTL1
0x03	0x29, 0x89, 0x75, 0x28, 0x50	HS6200_BANK1_CAL_CTL
0x11	0x52, 0xC2, 0x09, 0xAC	HS6200_BANK1_RX_CTRL
0x13	0x80, 0x14, 0x08, 0x29	HS6200_BANK1_FAGC_CTRL_1
0x1E	0x1F, 0x64, 0x00, 0x11	HS6200_BANK1_RF_IVGEN

6200的初始化配置里面开了白化中间字节的0x75中的第0x10就是白化位

图表 5 白化开关

3.3.4.FEATURE 寄存器

这个寄存器的话，6220 和 6200 是有区别的，6200 值使用了低 3bit，6220 是 8bit 都使用了，在 2.4G 模式下，bit3 这个位是要置”1”的。低 3bit 根据实际使用配置，需要注意的是收发端的配置要一直。

FEATURE (RW) Address: 1Dh

HS6220的feature寄存器的bit4要置“1”

低3bit根据实际使用配置

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Long_packet_en	Ble_en	Soft_rst	BP_GAU	Vco_amp_tx_mux	EN_DPL	EN_ACK_PA	EN_DYN_A
0	0	0	1	1	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

FEATURE (RW) Address: 1Dh

HS6200的feature寄存器根据实际使用配置

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved					EN_DPL	EN_ACK_PA	EN_DYN_A
0					0	0	0
RW					RW	RW	RW

图表 6 FEATURE 寄存器

3.3.5.DYNPD 寄存器

这个寄存器 HS6220 和 HS6200 是有区别的，6200 的 dynpd 寄存器的低 6bit 是对应的 6 个 pipe，而 6220 的 dynpd 寄存低 3bit 是对应 3 个 pipe 剩下的是其他功能 bit。这里需要注意的是，bit3 是 3 线,4 线 SPI 的控制位，如果使用的是 3 线 SPI 的话，这个 bit 要置“0”，如果是使用 4 线的话，要置“1”。如果配置不对的话，SPI 线用逻辑分析仪抓的时候数抓不到所读的数据的。

DYNPD (RW) Address: 1Ch

HS6220的DYNPD寄存器

3线4线SPI设置位

3个pipe对应位

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reg_output	Reg_output_en	Bypass_io	Xn_en	Spi4_en	DPL_P2	DPL_P1	DPL_P0
0	0	1	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

DYNPD (RW) Address: 1Ch

HS6200的DYNPD寄存器，只有对应的6个pipe

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved		DPL_P5	DPL_P4	DPL_P3	DPL_P2	DPL_P1	DPL_P0
0		0	0	0	0	0	0
RW		RW	RW	RW	RW	RW	RW

图表 7 DYNPD 寄存器

3.3.6.接收端防死机处理

在接收端的接收函数，需要做下面的 2 个处理，一个是 20~50ms 时间收不到数据，要进行一次睡眠唤醒；如果 300~500ms 的时间一次数据都收不到，则要软件复位重新初始化一次。

```
/* 以下的复位RF和让RF睡眠唤醒的动作是必须的，不可缺少 */
if(reset_lms_cnt > 300) // 如果300ms都没有收到1包数据，则复位RF重新初始化
{
    HS6220_Init();
    HS6220_write_byte(HS6220_BANK0_FEATURE, 0x10);
    HS6220_write_byte(HS6220_BANK0_EN_AA, 0x00);
    HS6220_write_byte(HS6220_BANK0_CONFIG, 0xfb);
    HS6220_write_byte(HS6220_BANK0_RX_PW_P0, 10);
    HS6220_write_byte(HS6220_BANK0_RF_CH, 5);
    HS6220_write_byte(HS6220_BANK0_EN_RXADDR, 0x03); // scramble_en = 0
    HS6220_CE_High();
    reset_lms_cnt = 0; // 清零复位RF计时器
}

// 发送端是4ms一个周期发一包数据，这里20ms的话就有5包数据的时间了
if((tim2_lms_cnt % 20) == 0) // 如果20ms都没有收到1包数据，则让RF睡眠唤醒一次
{
    HS6220_CE_Low();
    HS6220_Flush_Rx();
    HS6220_Clear_All_Irq();
    HS6220_write_byte(HS6220_BANK0_PMU_CTL, 0xAE); // RF进入深睡眠
    delay_ms(1); // 这里的1ms延时不可缺少，也不能小
    HS6220_write_byte(HS6220_BANK0_PMU_CTL, 0xAC); // 唤醒RF
    delay_ms(1); // 这里的1ms延时不可缺少，也不能小
    HS6220_CE_High();
}

/* 以上的复位RF和让RF睡眠唤醒的动作是必须的，不可缺少 */
```

图表 8 接收端防死机处理

3.3.7.接收端接收处理数据的正确顺序

```
HS6220_CE_High(); // CE拉高才能接收数据

for(;;)
{
    delay_us(100); // 延时100us不至于频繁读取状态寄存器
    status = HS6220_read_byte(HS6220_BANK0_STATUS); // 读状态寄存器看看是否有收到数据
    if ((HS6220_STATUS_RX_DR & status)) // 如果有收到数据
    {
        len = HS6220_ReceivePack(Rx Payload); // 读取FIFO里面数据的长度
        if(len == 10) // 如果FIFO里面是10个字节的数据
        {
            // 接收到数据并读取FIFO之后，这个操作必须的
            HS6220_CE_Low(); // CE拉低
            HS6220_Flush_Rx(); // 清空RX FIFO
            HS6220_Clear_All_Irq(); // 清状态寄存器标志
            HS6220_CE_High(); // CE拉高等待下次数据到来
        }
    }
}

// 要注意收到数据之后的正确操作，
// CE先拉低，
// 清FIFO，
// 清标志，
// 设置频点，
// 设置地址，
// .....
// CE拉高进入接收状态
```

图表 9 接收端操作流程

3.3.8.发送端正确的发送数据操作

<pre> HS6220_SendPack(HS6220_W_TX_PAYLOAD, Tx_Payload, 10); // 发送端：只有在有数据需要发送，的时候才给CE信号，其他时间CE都是拉低的-- // --如果需要切换到接收的话，那么在接收状态下CE要一直为高 HS6220_CE_High(); // ce pulse 20us delay_us(35); HS6220_CE_Low(); delay_us(3500); HS6220_Flush_Tx(); HS6220_Clear_All_Irq(); </pre>	<p>正常状态下CE是拉低的 设置频点, 设置地址, 设置功率, , , 等等。 写入TX FIFO数据, CE拉高 延时至少20us CE拉低 查询状态寄存器等待数据发送完成 (下一次数据发送)</p>	<p>正常状态下CE是拉低的 设置频点, 设置地址, 设置功率, , , 等等。 写入TX FIFO数据 CE拉高 延时1ms(1ms时间数据是可以发完的) CE拉低 (下一次数据发送)</p>
	CE脉冲发送方式	CE高电平发送方式

图表 10 发送端操作流程

3.3.9 IRQ 脚的输出控制功能

要使用 IRQ 脚做输出控制功能，需要在芯片上电初始化的时候就配置 IRQ 脚，以免误控制。IRQ 脚控制使能位在 DYNPD 寄存器的 bit6。然后控制 IRQ 脚是高电平还是低电平是在 DYNPD 寄存器的 bit7 这个位。

```

void HS6220_Init(void)
{
    unsigned char temp[5];

    HS6220_write_byte(HS6220_BANK0_FEATURE, SOFT_RST); // soft_reset

    #if (HS6220_SPI_NWIRE == SPI_4_WIRE)
        // 默认是3线SPI的，如果要使用4线SPI，则上电之后要设置一下
        HS6220_write_byte(HS6220_BANK0_DYNPD, 0x08 | 0x40); // 0x08; // //
        如果要用6220的IRQ脚控制马达的话，要设置IRQ脚，而且必须是在最前面设置他
    #else
        HS6220_write_byte(HS6220_BANK0_DYNPD, 0x00 | 0x40); // 0x00; // //
        如果要用6220的IRQ脚控制马达的话，要设置IRQ脚，而且必须是在最前面设置他
    #endif

    #define LED1_HIGH() do{HS6220_write_byte(HS6220_BANK0_DYNPD, HS6220_read_byte(HS6220_BANK0_DYNPD) | 0x80);
    delay_us(10);}while(0)
    #define LED1_LOW() do{HS6220_write_byte(HS6220_BANK0_DYNPD, HS6220_read_byte(HS6220_BANK0_DYNPD) & 0x7f);
    delay_us(10);}while(0)
        
```

图表 11 IRQ 脚输出控制操作

3.4 常用寄存器的配置介绍

3.4.1 CONFIG(BANK0--00)寄存器

做普通 2.4G 使用是 bit7 要置”1”，bit6~bit4 是 RF 有中断产生的时候，信号在 IRQ 脚出现，低电平有效，bit3 是开 CRC 功能，bit2 是 CE 控制方式，”0”是命令控制，”1”是寄存器控制，bit1 是芯片上电掉电位，”1”是给芯片上电，bit0 是发送接收模式控制位，”0”是发送，”1”是接收。

CONFIG (RW) Address: 00h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Tx_gurd_en	MASK_RX_DR	MASK_TX_DS	MASK_MAX_RT	EN_CRC	CE_REG	PWR_UP	PRIM_RX
1	0	0	0	1	0	0	1
RW	RW	RW	W	RW	RW	RW	RW

图表 12 CONFIG 寄存器介绍

3.4.2 EN_AA(BANK0--01)寄存器

这个寄存器是使能 pipe 的自动应答功能，低 3bit 对应 3 个 pipe。“1”是使能对应管道的自动应答，“0”是禁止对应 pipe 的自动应答。

EN_AA (RW) Address: 01h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reversed					ENAA_P2	ENAA_P1	ENAA_P0
0					1	1	1
RW					RW	RW	RW

图表 13 EN_AA 寄存器介绍

3.4.3 EN_RXADDR(BANK0--02) 寄存器

这个寄存器的 bit3 是白化功能开关，“1”是打开白化功能，“0”关闭白化功能.bit2~bit0 是对应 pipe 的使能。

EN_RXADDR (RW) Address: 02h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reversed				scramble_en	ERX_P2	ERX_P1	ERX_P0
0				1	0	1	1
RW				RW	RW	RW	RW

图表 14 EN_RXADDR 寄存器介绍

3.4.4 PMU_CTL(BANK0--03)寄存器

这个寄存器的 bit7~bit2 初始化的时候设置成 0xA8 应用时设置成 0xAC，然后 bit1~bit0 是工作模式，“00”是工作模式，“01”是深睡眠模式，“10”是浅睡眠模式。

PMU_CTL (RW) Address: 03h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Rtc32k_rdy_enb_reg	Rtc32k_rdy_enb_mm	Digldo_enb_dly_reg	Digldo_enb_mm	Digldo_enb_reg	rtc32k_en	RF_PWRDWN[1:0]	
1	0	1	0	1	0	01	
RW	RW	RW	RW	RW	RW	RW	

图表 15 PMU_CTL 寄存器介绍

3.4.5 SETUP_RETR(BANK0--04)寄存器

这个寄存器的 bit7~bit4 是重传的间隔时间， $0=256\mu s$ ， $T=(n+1)*256\mu s$ ，bit3~bit0 是重传的次数。是在开了自动重传的时候有效。

SETUP_RETR (RW) Address: 04h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ARD				ARC			
4'b0				4'b11			
RW				RW			

图表 16 SETUP_RETR 寄存器介绍

3.4.6 RF_CH(BANK0--05)寄存器

这个寄存器是设置通信信道的，写入信道值从 0~83，对应的是 2400 到 2483。

RF_CH (RW) Address: 05h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reg_Rf_ch							
0x32							
RW							

图表 17 RF_CH 寄存器介绍

3.4.7 RF_SETUP(BANK0--06)寄存器

这个寄存器的 bit7 是载波位, "1"是发载波, "0"是发数据, 要 CE 为高才会有载波出来。Bit6+bit2~0 是功率档位设置寄存器, bit5 是校准完成标志位, 校准完成之后这个位要清零。bit4 是使能校准位, "1"是使能校准。bit3 是速率控制位, "1"是 2M 速率, "0"是 1M 速率。

RF_SETUP (RW) Address: 06h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CONT_WAVE	PA_PWR[3]	CAL_DONE	CAL_EN	RF_DR_HIGH	Pa_power		
0	1	0	1	0	3'b000		
RW	RW	RW	RW	RW	RW		

图表 18 RF_SETUP 寄存器介绍

3.4.8 STATUS(BANK0--07)寄存器

正常读取这个寄存器的时候, 应该是 0x0E 这个值, bit7 是表示当前 BANK, "1"是 BANK1, "0"是 BANK0, 一定要确保我们是在 BANK0 下做通信操作。bit6 是接收到数据时为"1"。bit5 是发送完成时为"1"。bit4 是最大传输次数达到时为"1"。bit3 是表示当前 CE 的高低电平状态。bit2~1 是表示哪个 pipe 有数据, 都没有数据时是"11"。bit0 是 TX FIFO 是否满。

STATUS (RW) Address: 07h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BANK	RX_DR	TX_DS	SYNC_DS	CE	RX_P_NO		TX_FULL
0	0	0	0	0	2'b11		0
R	RW	RW	RW	R	R		R

图表 19 STATUS 寄存器介绍

3.4.9 RX_ADDR_P0(BANK0—0A)寄存器

这个寄存器是设置接收时的地址的，一共 5 个字节。RX_ADDR_P1 和 RX_ADDR_P2 的高 4 字节和 RX_ADDR_P0 完全一样，只有低字节不一样，低字节是 C2 和 C3。

3.4.10 TX_ADDR(BANK0--10)寄存器

这个寄存器是设置发送时的地址，一共 5 个字节。

3.4.11 RX_PW_P0(BANK0--11)寄存器

这个寄存器是设置 pipe0 收发数据包长度的，如果是设置成定长的数据包收发的话，那么 6220 只发送所设置的固定长度的数据包内容和接收固定长度的数据包内容，其他不符合长度的数据包是丢弃的。普通 2.4G 模式下最多 32 字节数据包长度。RX_PW_P1 和 RX_PW_P2 是一样的道理。

3.4.12 FIFO_STATUS(BANK0--17)寄存器

这个寄存器的 bit6 是表示是否重用数据包内容，如果你的数据包内容是重复使用的，不想每次都通过 SPI 写一次，则可以使能这个位，然后通过 REUSE_TX_PL 命令可以重用数据包。bit5 是 TX FIFO 满标志，bit4 是 TX FIFO 空标志，bit1 是 RX FIFO 满标志，bit0 是 RX FIFO 空标志。

FIFO_STATUS (RW) Address: 17h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	TX_REUSE_PL	TX_FULL	TX_EMPTY	Reserved		RX_FULL	RX_EMPTY
0	0	0	1	0		0	1
RW	R	R	R	RW		R	R

图表 20 FIFO_STATUS 寄存器介绍

3.4.13 DYNPD(BANK0—1C)寄存器

这个寄存器的 bit3 是 3/4 线 SPI 选择位，bit2~0 是对应的 pipe 是否使能动态数据

包长度。

DYNPD (RW) Address: 1Ch

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reg_output	Reg_output_en	Bypass_io	Xn_en	Spi4_en	DPL_P2	DPL_P1	DPL_P0
0	0	1	0	0	0	0	0
RW		RW	RW	RW	RW	RW	RW

2	0	DPL_P2	Enable dynamic payload length data pipe 2. (Requires EN_DPL and ENAA_P2)
1	0	DPL_P1	Enable dynamic payload length data pipe 1. (Requires EN_DPL and ENAA_P1)
0	0	DPL_P0	Enable dynamic payload length data pipe 0. (Requires EN_DPL and ENAA_P0)

图表 21 DYNPD 寄存器介绍

3.4.14 FEATURE(BANK0—1D)寄存器

这个寄存器的 bit5 是软件复位控制位。Bit4 是高斯滤波器开关位，做 2.4G 模式使用时 bit4 要置“1”，bit2 是使能动态数据包长度，bit1 是使能 ACK 包，需要提前先把 ACK 包写到接收端的 TX FIFO 里面去，配合 W_ACK_PAYLOAD 这个命令写数据到 TX FIFO，要动长 ACK 包的话配合 bit2 使用，bit0 是不带应答的。

FEATURE (RW) Address: 1Dh

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Long_packet_en	Ble_en	Soft_rst	BP_GAU	Vco_amp_tx_mux	EN_DPL	EN_ACK_PAY	EN_DYN_ACK
0	0	0	1	1	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

2	0	EN_DPL	Enables Dynamic Payload Length
1	0	EN_ACK_PAY	Enables Payload with ACK
0	0	EN_DYN_ACK	Enables the W_TX_PAYLOAD_NOACK command

图表 22 FEATURE 寄存器介绍

4.6200 和 6220 互通的配置

```

27  /*Calibration config infor*/
28  static const unsigned char RF_Calibration_Data[] =
29  {
30      /* Register Addr          DataLenth  Data*/
31      RF_BANK0_CONFIG,        1,      0x03,
32      RF_BANK0_RF_CH,         1,      0x32,
33      RF_BANK0_RF_SETUP,      1,      0x40,
34      0xFF /*Addr=0xFF,complete flag*/
35  };
36
37  static const unsigned char RF_Calibration_Analog[] =
38  {
39      /*Register Addr          DataLenth Data*/
40      RF_BANK1_PLL_CTL0,      4,      0x40,0x01,0x10,0xE5,
41      RF_BANK1_CAL_CTL,       5,      0x20,0x08,0x50,0x40,0x50,
42      RF_BANK1_IF_FREQ,       3,      0x00,0x00,0x1F,
43      RF_BANK1_FDEV,          1,      0x20,
44      RF_BANK1_DAC_CAL_HI,    1,      0x7f,
45      RF_BANK1_RF_IVGEN,      4,      0x1f,0x64,0x00,0x81,
46      0xFF /*Addr=0xFF,complete flag*/
47  };
48
49  static const unsigned char RF_Init_Analog[] =
50  {
51      RF_BANK1_PLL_CTL0,      4,      0x40,0x01,0x30,0xE1,
52      RF_BANK1_PLL_CTL1,      4,      0x00,0x42,0x10,0x06,
53      RF_BANK1_CAL_CTL,       5,      0x29,0x89,0x65,0x28,0x50,
54      RF_BANK1_RX_CTRL,       4,      0x52,0xC2,0x09,0xAC,
55      RF_BANK1_FAGC_CTRL_1,   4,      0x80,0x14,0x08,0x29,
56      RF_BANK1_RF_IVGEN,      4,      0x99,0x64,0x00,0x11,
57      0xFF /*Addr=0xFF,complete flag*/
58  };
59
60  /*config infor*/
61  static const unsigned char RF_Init_Data[] =
62  {
63      /*Register Addr          DataLenth Data*/
64      RF_BANK0_CONFIG,        1,      0x0f,
65      RF_BANK0_RX_PW_PO,      1,      0x20,
66      RF_BANK0_DYNPD,         1,      0x00,
67      RF_BANK0_FEATURE,        1,      0x00,
68      RF_BANK0_SETUP_VALUE,    5,      0x28,0x32,0x80,0x06,0x00,
69      RF_BANK0_PRE_GURD,       1,      0x77,
70      RF_BANK0_EN_AA,          1,      0x00,
71      RF_BANK0_EN_RXADDR,      1,      0x01,
72      RF_BANK0_SETUP_AW,       1,      0x03,
73      RF_BANK0_SETUP_RETR,     1,      0x00,
74      RF_BANK0_RF_CH,          1,      0x02,
75
76      RF_BANK0_RF_SETUP,       1,      0x47,
77      RF_BANK0_RX_ADDR_PO,     5,      0x46,0x0b,0xaf,0x43,0x98,
78      RF_BANK0_TX_ADDR,        5,      0x46,0x0b,0xaf,0x43,0x98,
79      0xFF /*Addr=0xFF,complete flag*/
80  };

```

图表 23 HS6200 初始化配置


```
void HS6220_Init(void)
{
    unsigned char temp[5];

    HS6220_write_byte(HS6220_BANK0_FEATURE, SOFT_RST); // soft_reset
    // 上电进行软件复位, 复位之后左右的寄存器恢复默认值

    #if (HS6220_SPI_NWIRE == SPI_4_WIRE)
        // 默认是3线SPI的, 如果要使用4线SPI, 则上电之后要设置一下
        // 如果要用6220的IRQ脚控制马达的话, 要设置IRQ脚, 而且必须是在最前面设置他
        HS6220_write_byte(HS6220_BANK0_DYNPD, 0x08); // 0x08 | 0x40; //
    #else
        // 根据配置是3线还是4线SPI
        // 如果要用6220的IRQ脚控制马达的话, 要设置IRQ脚, 而且必须是在最前面设置他
        HS6220_write_byte(HS6220_BANK0_DYNPD, 0x00); // 0x00 | 0x40; //
    #endif

    HS6220_CE_Low();
    HS6220_write_byte(HS6220_BANK0_CONFIG, 0x8b); // power up 给芯片上电
    delay_ms(3); // wait 3 ms 必须至少等待3ms
    HS6220_write_byte(HS6220_BANK0_PMU_CTL, 0xac); // HS6220_PWRDWN = 00
    delay_ms(2); // 必须至少等待2ms 是芯片处于工作模式

    HS6220_Bank_Switch(HS6220_Bank1);
    HS6220_write_byte(HS6220_BANK1_TEST_PKDET, 0x20); // pll_vdiv2_sel = 0

    /* increase filter & gc threshold start */
    temp[0] = 0x01;
    HS6220_wr_buffer(HS6220_BANK1_FAGC_CTRL_1, temp, 1);

    HS6220_Bank_Switch(HS6220_Bank0);

    HS6220_CE_High(); // CE脉冲至少40us, 使芯片启动校准流程
    delay_us(100);
    HS6220_CE_Low(); // 一定要注意, 校准的时候CE是低的

    while((HS6220_read_byte(HS6220_BANK0_RF_SETUP) & 0x20) == 0x00); // wait cal done
    HS6220_write_byte(HS6220_BANK0_RF_SETUP, 0x40); // cal_en = 0 清除校准使能位

    HS6220_Bank_Switch(HS6220_Bank1);
    temp[2] = 0x75; // bp_dac = 1 bp_rc = 1
    temp[1] = 0x98; // bp_vco_amp = 1 bp_vco_ldo = 1
    temp[0] = 0x20;
    HS6220_wr_buffer(HS6220_BANK1_CAL_CTL, temp, 3);

    HS6220_Bank_Switch(HS6220_Bank0);
    temp[0] = 0x46;
    temp[1] = 0x0b;
    temp[2] = 0xaf; // 设置TX RX的通信地址, 这里属于用户应用配置的内容了。
    temp[3] = 0x43;
    temp[4] = 0x98;
    HS6220_wr_buffer(HS6220_BANK0_RX_ADDR_P0, temp, 5); // set address
    HS6220_wr_buffer(HS6220_BANK0_TX_ADDR, temp, 5); // set address

    HS6220_Clear_All_Irq(); // 这里可以在使用的时候去写, , 直到这里, 目前CE还是处于拉低的状态
    HS6220_Flush_Tx();
}
```

图表 24 HS6220 初始化配置

```
HS6220_write_byte(RF_BANK0_FEATURE, 0x00);
HS6220_write_byte(RF_BANK0_EN_AA, 0x00); // HS6220做接收, 没有ACK的应用配置
HS6220_write_byte(RF_BANK0_CONFIG, 0xf3);
HS6220_write_byte(HS6220_BANK0_RX_PW_P0, 0x20);
HS6220_write_byte(RF_BANK0_RF_CH, 0x02);
HS6220_write_byte(HS6220_BANK0_EN_RXADDR, 0x03); //scramble_en = 0
HS6220_CE_Low();
```

```
HS6200_write_byte(RF_BANK0_FEATURE, 0x00);
HS6200_write_byte(RF_BANK0_CONFIG, 0x02); HS6200做发射, 没有ACK的配置
HS6200_write_byte(RF_BANK0_EN_AA, 0x00);
HS6200_write_byte(RF_BANK0_SETUP_AW, 0x03);
HS6200_write_byte(RF_BANK0_RF_CH, 0x02);
```

图表 25 6220 收, 6200 发, 没有 ACK 的应用配置

```
HS6220_write_byte(RF_BANK0_FEATURE, 0x00);
HS6220_write_byte(RF_BANK0_EN_AA, 0x00); HS6220做发送, 没有ACK
HS6220_write_byte(RF_BANK0_CONFIG, 0xf2);
HS6220_write_byte(HS6220_BANK0_RX_PW_P0, 0x20);
HS6220_write_byte(RF_BANK0_RF_CH, 0x02);
HS6220_write_byte(HS6220_BANK0_EN_RXADDR, 0x03); //scramble_en = 0
HS6220_Flush_Tx();
HS6220_Flush_Rx();
HS6220_Clear_All_Irq();

HS6200_write_byte(RF_BANK0_FEATURE, 0x00);
HS6200_write_byte(RF_BANK0_CONFIG, 0x03); HS6200做接收, 没有ACK
HS6200_write_byte(RF_BANK0_EN_AA, 0x00);
HS6200_write_byte(RF_BANK0_SETUP_AW, 0x03);
HS6200_write_byte(RF_BANK0_RF_CH, 0x02);
HS6200_Flush_Tx();
HS6200_Flush_Rx();
HS6200_Clear_All_Irq();
```

图表 26 HS6220 发送, HS6200 接收, 没有 ACK 的应用配置

```
HS6220_write_byte(RF_BANK0_FEATURE, HS6220_read_byte(RF_BANK0_FEATURE) | 0x07);
HS6220_write_byte(RF_BANK0_DYNPD, HS6220_read_byte(RF_BANK0_DYNPD) | 0x07);
HS6220_write_byte(RF_BANK0_EN_AA, 0x07);
HS6220_write_byte(RF_BANK0_SETUP_RETR, 0x08);
HS6220_write_byte(RF_BANK0_CONFIG, 0xfa); HS6220做发送, 有ACK

HS6220_CE_Low();
HS6220_ModeSwitch(Rf_PTX_Mode);

HS6200_write_byte(RF_BANK0_FEATURE, 0x07);
HS6200_write_byte(RF_BANK0_EN_AA, 0x07);
HS6200_write_byte(RF_BANK0_CONFIG, 0x0f); HS6200做接收, 有ACK
HS6200_write_byte(RF_BANK0_SETUP_RETR, 0x08);
HS6200_write_byte(RF_BANK0_DYNPD, 0x3f);

HS6200_CE_Low();
HS6200_ModeSwitch(Rf_PRX_Mode);
```

图表 27 HS6220 发送, HS6200 接收, 有 ACK 的应用配置

```
HS6220_CE_Low();
HS6220_write_byte(RF_BANK0_FEATURE, HS6220_read_byte(RF_BANK0_FEATURE) | 0x07);
HS6220_write_byte(RF_BANK0_DYNPD, HS6220_read_byte(RF_BANK0_DYNPD) | 0x07);
HS6220_write_byte(RF_BANK0_EN_AA, 0x07);
HS6220_write_byte(RF_BANK0_SETUP_RETR, 0x08); HS6220接收, 有ACK
HS6220_write_byte(RF_BANK0_CONFIG, 0xfb);
```

```
HS6200_CE_Low();
HS6200_write_byte(RF_BANK0_FEATURE, 0x07);
HS6200_write_byte(RF_BANK0_EN_AA, 0x07);
HS6200_write_byte(RF_BANK0_CONFIG, 0x0f);    HS6200做发送, 有ACK
HS6200_write_byte(RF_BANK0_SETUP_RETR, 0x08);
HS6200_write_byte(RF_BANK0_DYNPD, 0x3f);
HS6200_ModeSwitch(Rf_PTX_Mode);
```

图表 28 HS6220 接收, HS6200 发送, 有 ACK 功能的应用配置

4.HS6220 使用注意事项

4.1 天线端防止烙铁漏电烧坏处理

为了防止在生产的时候, 由于烙铁没有做防漏电处理, 导致芯片的天线脚被烧坏, 需要在芯片脚和天线焊接点中间串一个 5pf 左右的电容。

4.2 晶体负载电容

晶体两端的负载电容可以不用焊接。

4.3 SPI 接口电平

SPI 接口不支持 5V 电平, 只能支持 3.3V 的电平。

4.4 通信地址要求

通信地址的设置, 不能是所有的字节都一样, 也不能出现连续的 6 个“0”或者连续的 6 个“1”。也不能是全部是 01010101.....这样的。