CC1100

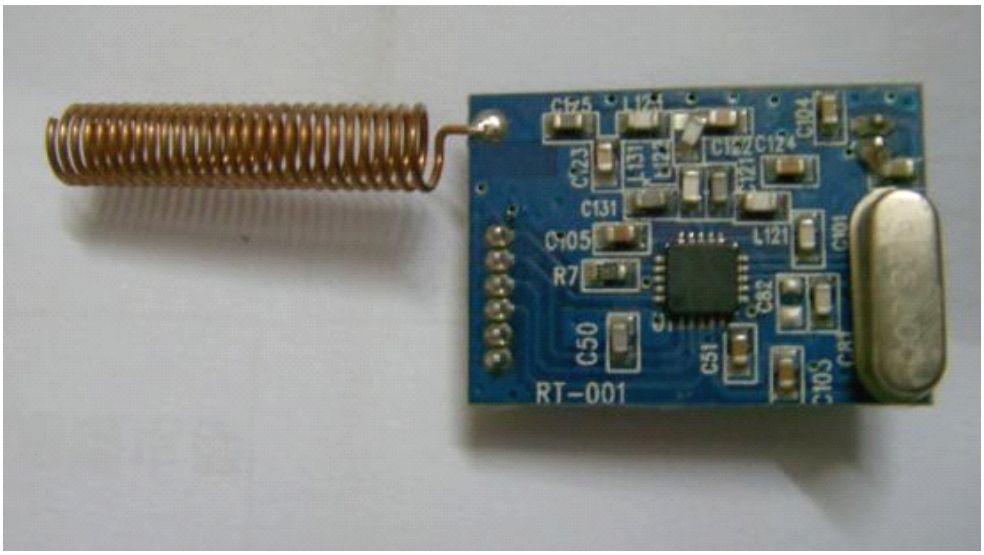
说

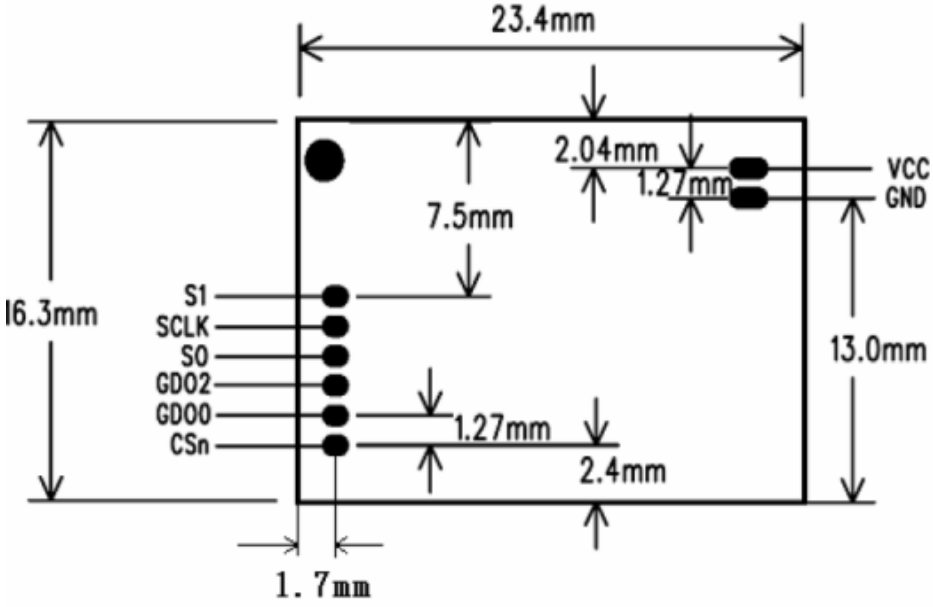
明

书

2008 年 12 月 20 日

一、模块介绍





(1) 315、433、868、915Mh 的ISM 和SRD 频段

(2) 最高工作速率500kbps，支持2-FSK、GFSK 和MSK 调制方式

(3) 高灵敏度（1.2kbps 下-110dDm，1％数据包误码率）

(4) 内置硬件CRC 检错和点对多点通信地址控制

(5) 较低的电流消耗（RX 中，15.6mA，2.4kbps,433MHz）

(6) 可编程控制的输出功率，对所有的支持频率可达+10dBm

(7) 支持低功率电磁波激活功能

(8) 支持传输前自动清理信道访问（CCA），即载波侦听系统

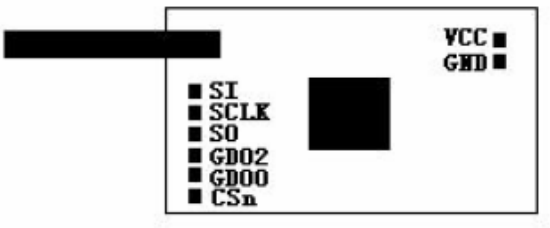
(9) 快速频率变动合成器带来的合适的频率跳跃系统

(10) 模块可软件设地址，软件编程非常方便

(11) 1.27MMDIP 间距接口，便于嵌入式应用

(12) 单独的 64 字节 RX 和 TX 数据 FIFO

二、接口电路



说明:

(1) VCC 脚接电压范围为 1.9V-3.6V 之间，不能在这个区间之外，超过3.6V 将会烧毁模块。推荐电压3.3V 左右。

(2) 除电源 VCC 和接地端，其余脚都可以直接和普通的5V 单片机 IO 口直接相连，无需电平转换。当然对3V 左右的单片机更加适用了。

(3) 硬件上面没有SPI 的单片机也可以控制本模块，用普通单片机

IO 口模拟SPI 不需要单片机真正的串口介入，只需要普通的单片机IO 口就可以了，当然用串口也可以了。

(4) 排针间距为1.27mm,DIP 插针，如果需要其他封装接口，比如密脚插针，或者其他形式的接口，可以联系我们定做。

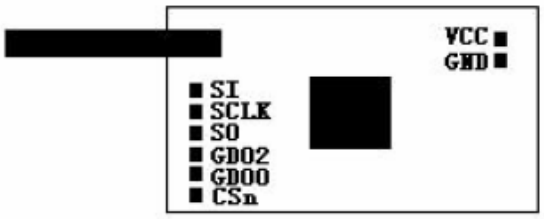
(5) 与 51 系列单片机P0 口连接时候，需要加10K 的上拉电阻,与其余口连接不需要。

(6) 其他系列的单片机，如果是5V 的，请参考该系列单片机IO口输出电流大小，如果超过10mA，需要串联电阻分压，否则容易烧毁模块!如果是 3.3V 的，可以直接和 RF1100 模块的 IO 口线连接。

三、模块结构

NRF1100 模块使用Chipcon 公司的CC1100 芯片开发而成。

NRF1100 单片无线收发器工作在433/868/915MHZ 的ISM 频段由一个完全集成的频率调制器一个带解调器的接收器一个功率放大器一个晶体震荡器和一个调节器组成。工作特点是自动产生前导码 和 CRC 可以很容易通过 SPI 接口进行编程配置，电流消耗低。

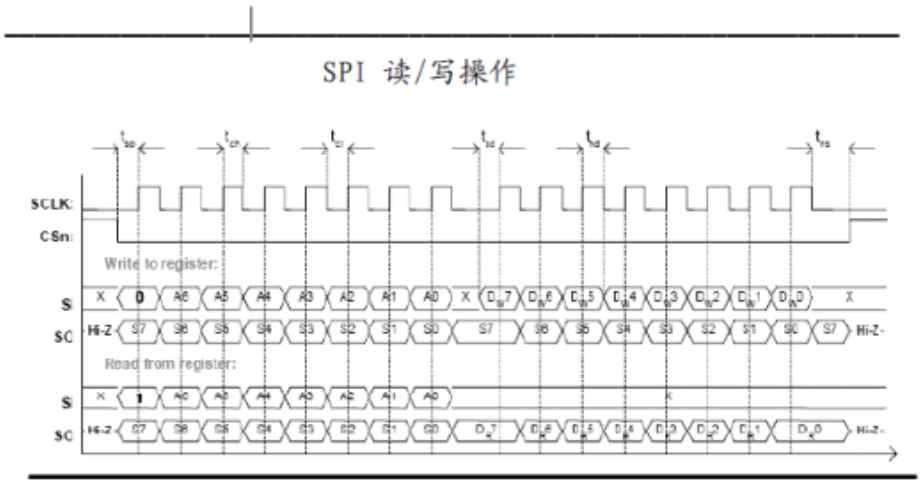


四、配置NRF1100 模块

所有配置字都是通过SPI 接口送给CC1100。SIP 接口的工作方式

可通过SPI 指令进行设置

4.3 SPI 时序



4.4 SPI 接口定时要求

参数	描述	最小值	最大值
FSCLK	SCLK频率	0	10MHz
tsp,pd	CSn低到SCLK的正边缘，功率降低模式下	TBDus	-
tsp	CSn低到SCLK的正边缘，活动模式下	TBDns	-
tch	时钟高	50ns	-
tcl	时钟低	50ns	-
trise	时钟上升时间	-	TBDns
tfall	时钟上升时间	-	TBDns
tsd	向SCLK的正边缘建立数据	TBDns	-
thd	在SCLK的正边缘之后保持数据	TBDns	-
tns	SCLK到CSn高时的负边缘	TBDns	-

5.4 状态字节概述

<table>
<tr><th>比特</th><th>名称</th><th>描述</th></tr>
<tr><td>7</td><td>CHIP_RDYn</td><td>保持高，直到功率和晶体已稳定。当使用SPI接口时应始终为低。</td></tr>
<tr><td>6:4</td><td>STATE[2:0]</td><td>表明当前主状态机模式
<table>
<tr><th>值</th><th>状态</th><th>描述</th></tr>
<tr><td>000</td><td>空闲</td><td>空闲状态</td></tr>
<tr><td>001</td><td>RX</td><td>接收模式</td></tr>
<tr><td>010</td><td>TX</td><td>发送模式</td></tr>
<tr><td>011</td><td>FSTXON</td><td>快速TX准备</td></tr>
<tr><td>100</td><td>校准</td><td>频率合成器校准正运行</td></tr>
<tr><td>101</td><td>迁移</td><td>PLL正迁移</td></tr>
<tr><td>110</td><td>RXFIFO_OVERFLOW</td><td>RX FIFO已经溢出。读出任何有用数据，然后用SFRX冲洗FIFO。</td></tr>
<tr><td>111</td><td>TXFIFO_OVERFLOW</td><td>TX FIFO已经下溢。同SFTX应答</td></tr>
</table></td></tr>
<tr><td>3:0</td><td>FIFO_BYTES_AVAILABLE[3:0]</td><td>TX FIFO中的自由比特数。若FIFO_BYTES_AVAILABLE=15，它表明有15或更多个比特是可用/自由的。</td></tr>
</table>

4.5 命令滤波

地址	滤波名	描述
0x30	SRES	重启芯片
0x31	SFSTXON	开启和校准频率合成器（若MCSM0.FSAUTOCAL=1）
0x32	SXOFF	关闭晶体振荡器
0x33	SCAL	校准频率合成器并关断（开启快速启动）。在不设置手动校准模式（MCSM0.FS_AUTOCAL=0）的情况下，SCAL能从空闲模式滤波。
0x34	SRX	启用RX。若上一状态为空闲且MCSM0.FS_AUTOCAL=1则首先运行校准。

0x35	STX	空闲状态：启用TX。若MCSM0.FS_AUTOCAL=1首先运行校准。若在RX状态且CCA启用：若信道为空则进入TX
0x36	SIDLE	离开RX/TX,关断频率合成器并离开电磁波激活模式若可用
0x37	SAFC	运行22.1节列出的频率合成器的AFC调节
0x38	SWOR	运行27.5节描述的自动RX选举序列（电磁波激活）
0x39	SPWD	当CSn为高时进入功率降低模式。
0x3A	SFRX	冲洗RX FIFO缓冲
0x3B	SFTX	冲洗TX FIFO缓冲
0x3C	SWORRST	重新设置真实时间时钟
0x3D	SNOP	无操作。可能用来为更简单的软件将滤波命令变为2字节。

4.6 配置寄存器概述

地址	寄存器	描述	保存在休眠状态中
0x00	IOCFG2	GDO2输出脚配置	Yes
0x01	IOCFG1	GDO1输出脚配置	Yes
0x02	IOCFG0	GDO0输出脚配置	Yes
0x03	FIFOTHR	RX FIFO和TX FIFO门限	Yes
0x04	SYNC1	同步词汇，高字节	Yes
0x05	SYNC0	同步词汇，低字节	Yes
0x06	PKTLEN	数据包长度	Yes
0x07	PKTCTRL1	数据包自动控制	Yes
0x08	PKTCTRL0	数据包自动控制	Yes
0x09	ADDR	设备地址	Yes
0x0A	CHANNR	信道数	Yes
0x0B	FSCTRL1	频率合成器控制	Yes
0x0C	FSCTRL0	频率控制词汇，高字节	Yes
0x0D	FREQ2	频率控制词汇，中间字节	Yes
0x0E	FREQ1	频率控制词汇，低字节	Yes
0x0F	FREQ0	调制器配置	Yes
0x10	MDMCFG4	调制器配置	Yes
0x11	MDMCFG3	调制器配置	Yes
0x12	MDMCFG2	调制器配置	Yes
0x13	MDMCFG1	调制器配置	Yes

0x14	MDMCFG0	调制器背离设置	Yes
0x15	DEVIATN	主通信控制状态机配置	Yes
0x16	MCSM2	主通信控制状态机配置	Yes
0x17	MCSM1	主通信控制状态机配置	Yes
0x18	MCSM0	频率偏移补偿配置	Yes
0x19	FOCCFG	位同步配置	Yes
0x1A	BSCFG	AGC控制	Yes
0x1B	AGCTRL2	AGC控制	Yes
0x1C	AGCTRL1	AGC控制	Yes
0x1D	AGCTRL0	高字节时间0暂停	Yes
0x1E	WOREVT1	低字节时间0暂停	Yes
0x1F	WOREVT0	电磁波激活控制	Yes
0x20	WORCTRL	前末端RX配置	Yes
0x21	FREND1	前末端TX配置	Yes
0x22	FREND0	频率合成器校准	Yes
0x23	FSCAL3	频率合成器校准	Yes
0x24	FSCAL2	频率合成器校准	Yes
0x25	FSCAL1	频率合成器校准	Yes
0x26	FSCAL0	RC振荡器配置	Yes
0x27	RCCTRL1	RC振荡器配置	Yes
0x28	RCCTRL0	频率合成器校准控制	Yes
0x29	FSTEST	产品测试	No
0x2A	PTEST	AGC测试	No
0x2B	AGCTEST	不同的测试设置	No
0x2C	TEST2	不同的测试设置	No
0x2D	TEST1	不同的测试设置	No
0x2E	TEST0		No

4.7 状态寄存器概述

地址	寄存器	描述
0x30 (0xF0)	PARTNUM	CC2550的组成部分数目
0x31 (0xF1)	VERSION	当前版本数
0x32 (0xF2)	FREQEST	频率偏移估计
0x33 (0xF3)	LQI	连接质量的解调器估计
0x34 (0xF4)	RSSI	接收信号强度指示
0x35 (0xF5)	MARCSTATE	控制状态机状态
0x36 (0xF6)	WORTIME1	WOR计时器高字节
0x37 (0xF7)	WORTIME0	WOR计时器低字节
0x38 (0xF8)	PKTSTATUS	当前GDOx状态和数据包状态

0x39 (0xF9)	VCOVCDAC	PLL校准模块的当前设定
0x3A (0xFA)	TXBYTES	TX FIFO中的下溢和比特数
0x3B (0xFB)	RXBYTES	RX FIFO中的下溢和比特数

五、NRF1100 编程指南

使用NRF1100 模块无需掌握任何专业无线或高频方面的理论，读

者只需要具备一定的C 语言程序基础即可。本文档没有涉及到的问题，

读者可以参考CC1100 官方手册.

范例程序中的部分相关代码段（完整代码请查看Demo 程序）：

```
// CC2500/CC1100 STROBE, CONTROL AND STATUS REGSITER
#define CCxxx0_IOCFG2 0x00 // GDO2 output pin configuration
#define CCxxx0_IOCFG1 0x01 // GDO1 output pin configuration
#define CCxxx0_IOCFG0 0x02 // GDO0 output pin configuration
#define CCxxx0_FIFOTHR 0x03 // RX FIFO and TX FIFO thresholds
#define CCxxx0_SYNC1 0x04 // Sync word, high INT8U
#define CCxxx0_SYNC0 0x05 // Sync word, low INT8U
#define CCxxx0_PKTLEN 0x06 // Packet length
#define CCxxx0_PKTCTRL1 0x07 // Packet automation control
#define CCxxx0_PKTCTRL0 0x08 // Packet automation control
#define CCxxx0_ADDR 0x09 // Device address
#define CCxxx0_CHANNR 0x0A // Channel number
#define CCxxx0_FSCTRL1 0x0B // Frequency synthesizer control
#define CCxxx0_FSCTRL0 0x0C // Frequency synthesizer control
#define CCxxx0_FREQ2 0x0D // Frequency control word, high
INT8U
#define CCxxx0_FREQ1 0x0E // Frequency control word, middle
INT8U
#define CCxxx0_FREQ0 0x0F // Frequency control word, low
INT8U
#define CCxxx0_MDMCFG4 0x10 // Modem configuration
#define CCxxx0_MDMCFG3 0x11 // Modem configuration
#define CCxxx0_MDMCFG2 0x12 // Modem configuration
#define CCxxx0_MDMCFG1 0x13 // Modem configuration
#define CCxxx0_MDMCFG0 0x14 // Modem configuration
#define CCxxx0_DEVIATN 0x15 // Modem deviation setting
#define CCxxx0_MCSM2 0x16 // Main Radio Control State
Machine configuration
#define CCxxx0_MCSM1 0x17 // Main Radio Control State
```

```
Machine configuration
#define CCxxx0_MCSM0 0x18 // Main Radio Control State
Machine configuration
#define CCxxx0_FOCCFG 0x19 // Frequency Offset Compensation
configuration
#define CCxxx0_BSCFG 0x1A // Bit Synchronization configuration
#define CCxxx0_AGCCTRL2 0x1B // AGC control
#define CCxxx0_AGCCTRL1 0x1C // AGC control
#define CCxxx0_AGCCTRL0 0x1D // AGC control
#define CCxxx0_WOREVT1 0x1E // High INT8U Event 0 timeout
#define CCxxx0_WOREVT0 0x1F // Low INT8U Event 0 timeout
#define CCxxx0_WORCTRL 0x20 // Wake On Radio control
#define CCxxx0_FREND1 0x21 // Front end RX configuration
#define CCxxx0_FREND0 0x22 // Front end TX configuration
#define CCxxx0_FSCAL3 0x23//Frequency synthesizer calibration
#define CCxxx0_FSCAL2 0x24 // Frequency synthesizer calibration
#define CCxxx0_FSCAL1 0x25 // Frequency synthesizer calibration
#define CCxxx0_FSCAL0 0x26 // Frequency synthesizer Calibration
#define CCxxx0_RCCTRL1 0x27 // RC oscillator configuration
#define CCxxx0_RCCTRL0 0x28 // RC oscillator configuration
#define CCxxx0_FSTEST 0x29 // Frequency synthesizer
calibration control
#define CCxxx0_PTEST 0x2A // Production test
#define CCxxx0_AGCTEST 0x2B // AGC test
#define CCxxx0_TEST2 0x2C // Various test settings
#define CCxxx0_TEST1 0x2D // Various test settings
#define CCxxx0_TEST0 0x2E // Various test settings

// Strobe commands
#define CCxxx0_SRES 0x30 // Reset chip.
#define CCxxx0_SFSTXON 0x31 // Enable and calibrate
frequency synthesizer (if MCSM0.FS_AUTOCAL=1).
// If in RX/TX: Go to
a wait state where only the synthesizer is
// running (for
quick RX / TX turnaround).
#define CCxxx0_SXOFF 0x32 // Turn off crystal oscillator.
#define CCxxx0_SCAL 0x33 // Calibrate frequency
synthesizer and turn it off
// (enables quick start).
#define CCxxx0_SRX 0x34 // Enable RX. Perform
calibration first if coming from IDLE and
// MCSM0.FS_AUTOCAL=1.
```

```
#define CCxxx0_STX 0x35 // In IDLE state: Enable TX.
Perform calibration first if
// MCSM0.FS_AUTOCAL=1. If in RX state and CCA is enabled:
// Only go to TX if channel is clear.
#define CCxxx0_SIDLE 0x36 // Exit RX / TX, turn off frequency
synthesizer and exit
// Wake-On-Radio mode if applicable.
#define CCxxx0_SAFC 0x37 // Perform AFC adjustment of the
frequency synthesizer
#define CCxxx0_SWOR 0x38 // Start automatic RX polling sequence
(Wake-on-Radio)
#define CCxxx0_SPWD 0x39 // Enter power down
mode when CSn goes high.
#define CCxxx0_SFRX 0x3A // Flush the RX FIFO buffer.
#define CCxxx0_SFTX 0x3B // Flush the TX FIFO buffer.
#define CCxxx0_SWORRST 0x3C // Reset real time clock.
#define CCxxx0_SNOP 0x3D // No operation. May
be used to pad strobe commands to two
// INT8Us for simpler software.
#define CCxxx0_PARTNUM 0x30
#define CCxxx0_VERSION 0x31
#define CCxxx0_FREQEST 0x32
#define CCxxx0_LQI 0x33
#define CCxxx0_RSSI 0x34
#define CCxxx0_MARCSTATE 0x35
#define CCxxx0_WORTIME1 0x36
#define CCxxx0_WORTIME0 0x37
#define CCxxx0_PKTSTATUS 0x38
#define CCxxx0_VCO_VC_DAC 0x39
#define CCxxx0_TXBYTES 0x3A
#define CCxxx0_RXBYTES 0x3B
#define CCxxx0_PATABLE 0x3E
#define CCxxx0_TXFIFO 0x3F
#define CCxxx0_RXFIFO 0x3F

// RF_SETTINGS is a data structure which contains all relevant
CCxxx0 registers
typedef struct S_RF_SETTINGS{
INT8U FSCTRL2; //
INT8U FSCTRL1; // Frequency synthesizer control.
INT8U FSCTRL0; // Frequency synthesizer control.
INT8U FREQ2; // Frequency control word, high INT8U.
INT8U FREQ1; // Frequency control word, middle INT8U.
```

```
INT8U FREQ0; // Frequency control word, low INT8U.
INT8U MDMCFG4; // Modem configuration.
INT8U MDMCFG3; // Modem configuration.
INT8U MDMCFG2; // Modem configuration.
INT8U MDMCFG1; // Modem configuration.
INT8U MDMCFG0; // Modem configuration.
INT8U CHANNR; // Channel number.
INT8U DEVIATN; // Modem deviation setting (when FSK
modulation is enabled).
INT8U FREND1; // Front end RX configuration.
INT8U FREND0; // Front end RX configuration.
INT8U MCSM0; // Main Radio Control State Machine configuration.
INT8U FOCCFG; // Frequency Offset Compensation Configuration.
INT8U BSCFG; // Bit synchronization Configuration.
INT8U AGCCTRL2; // AGC control.
INT8U AGCCTRL1; // AGC control.
INT8U AGCCTRL0; // AGC control.
INT8U FSCAL3; // Frequency synthesizer calibration.
INT8U FSCAL2; // Frequency synthesizer calibration.
INT8U FSCAL1; // Frequency synthesizer calibration.
INT8U FSCAL0; // Frequency synthesizer calibration.
INT8U FSTEST; // Frequency synthesizer calibration control
INT8U TEST2; // Various test settings.
INT8U TEST1; // Various test settings.
INT8U TEST0; // Various test settings.
INT8U IOCFG2; // GDO2 output pin configuration
INT8U IOCFG0; // GDO0 output pin configuration
INT8U PKTCTRL1; // Packet automation control.
INT8U PKTCTRL0; // Packet automation control.
INT8U ADDR; // Device address.
INT8U PKTLEN; // Packet length.
} RF_SETTINGS;
```

6.2 [通过SPI 接口向RF1100 配置寄存器读写配置信息]

RF1100 通过SPI 接口与单片机通讯，因此必须首先了解SPI 接口。

[SPI 概念] SPI 外围串行接口由四条线构成:

MOSI 主机输出从机输入（主机写操作）

MISO 主机输入从机输出（主机读操作）

SCK 串行时钟信号，由主机控制

CSN 片选信号，低电平有效

//<SPI 读写操作代码>

```
INT8U SpiTxRxByte(INT8U dat)
{
INT8U i,temp;
temp = 0;
SCK = 0;
for(i=0; i<8; i++)
{
if(dat & 0x80)
{
MOSI = 1;
}
else MOSI = 0;
dat <<= 1;
SCK = 1;
_nop_();
_nop_();
temp <<= 1;
if(MISO)temp++;
SCK = 0;
_nop_();
_nop_();
}
return temp;
}
```

需要注意的是：数据的传输时，高位在前，低位在后。

//<主机通过SPI 接口向CC1100 配置寄存器读写入信息>

```
INT8U halSpiReadReg(INT8U addr)
{
INT8U temp, value;
temp = addr|READ_SINGLE;//读寄存器命令
```

```
CSN = 0;
while (MISO);
SpiTxRxByte(temp);
value = SpiTxRxByte(0);
CSN = 1;
return value;
}
void halSpiWriteReg(INT8U addr, INT8U value)
{
CSN = 0;
while (MISO);
SpiTxRxByte(addr); //写地址

SpiTxRxByte(value); //写入配置

CSN = 1;
}

// 配置RF1100

void halRfWriteRfSettings(void)
{
halSpiWriteReg(CCxxx0_FSCTRL0, rfSettings.FSCTRL2);
// Write register settings
halSpiWriteReg(CCxxx0_FSCTRL1, rfSettings.FSCTRL1);
halSpiWriteReg(CCxxx0_FSCTRL0, rfSettings.FSCTRL0);
halSpiWriteReg(CCxxx0_FREQ2, rfSettings.FREQ2);
halSpiWriteReg(CCxxx0_FREQ1, rfSettings.FREQ1);
halSpiWriteReg(CCxxx0_FREQ0, rfSettings.FREQ0);
halSpiWriteReg(CCxxx0_MDMCFG4, rfSettings.MDMCFG4);
halSpiWriteReg(CCxxx0_MDMCFG3, rfSettings.MDMCFG3);
halSpiWriteReg(CCxxx0_MDMCFG2, rfSettings.MDMCFG2);
halSpiWriteReg(CCxxx0_MDMCFG1, rfSettings.MDMCFG1);
halSpiWriteReg(CCxxx0_MDMCFG0, rfSettings.MDMCFG0);
halSpiWriteReg(CCxxx0_CHANNR, rfSettings.CHANNR);
halSpiWriteReg(CCxxx0_DEVIATN, rfSettings.DEVIATN);
halSpiWriteReg(CCxxx0_FREND1, rfSettings.FREND1);
halSpiWriteReg(CCxxx0_FREND0, rfSettings.FREND0);
halSpiWriteReg(CCxxx0_MCSM0 , rfSettings.MCSM0 );
halSpiWriteReg(CCxxx0_FOCCFG, rfSettings.FOCCFG);
halSpiWriteReg(CCxxx0_BSCFG, rfSettings.BSCFG);
halSpiWriteReg(CCxxx0_AGCCTRL2, rfSettings.AGCCTRL2);
halSpiWriteReg(CCxxx0_AGCCTRL1, rfSettings.AGCCTRL1);
halSpiWriteReg(CCxxx0_AGCCTRL0, rfSettings.AGCCTRL0);
```

```
halSpiWriteReg(CCxxx0_FSCAL3, rfSettings.FSCAL3);
halSpiWriteReg(CCxxx0_FSCAL2, rfSettings.FSCAL2);
halSpiWriteReg(CCxxx0_FSCAL1, rfSettings.FSCAL1);
halSpiWriteReg(CCxxx0_FSCAL0, rfSettings.FSCAL0);
halSpiWriteReg(CCxxx0_FSTEST, rfSettings.FSTEST);
halSpiWriteReg(CCxxx0_TEST2, rfSettings.TEST2);
halSpiWriteReg(CCxxx0_TEST1, rfSettings.TEST1);
halSpiWriteReg(CCxxx0_TEST0, rfSettings.TEST0);
halSpiWriteReg(CCxxx0_IOCFG2, rfSettings.IOCFG2);
halSpiWriteReg(CCxxx0_IOCFG0, rfSettings.IOCFG0);
halSpiWriteReg(CCxxx0_PKTCTRL1, rfSettings.PKTCTRL1);
halSpiWriteReg(CCxxx0_PKTCTRL0, rfSettings.PKTCTRL0);
halSpiWriteReg(CCxxx0_ADDR, rfSettings.ADDR);
halSpiWriteReg(CCxxx0_PKTLEN, rfSettings.PKTLEN);
}
```

其中，rfSettings 需要定义并按照需要初始化适当的值。比如配置

如下：

```
// RF output power = 0 dBm
// RX filterbandwidth = 540.000000 kHz
// Deviation = 0.000000
// Datarate = 250.000000 kbps
// Modulation = (7) MSK
// Manchester enable = (0) Manchester disabled
// RF Frequency = 433.000000 MHz
// Channel spacing = 199.951172 kHz
// Channel number = 0
// Optimization = Sensitivity
// Sync mode = (3) 30/32 sync word bits detected
// Format of RX/TX data = (0) Normal mode, use FIFOs for RX and TX
// CRC operation = (1) CRC calculation in TX and CRC check in RX enabled
// Forward Error Correction = (0) FEC disabled
// Length configuration = (1) Variable length packets, packet length
configured by the first received byte after sync word.
// Packetlength = 255
// Preamble count = (2) 4 bytes
// Append status = 1
// Address check = (11) No address check
// FIFO autoflush = 0
// Device address = 0
// GDO0 signal selection = ( 6)
// GDO2 signal selection = (11) Serial Clock
```

```
const RF_SETTINGS rfSettings = {
0x00,
0x0B, // FSCTRL1 Frequency synthesizer control.
0x00, // FSCTRL0 Frequency synthesizer control.
0x10, // FREQ2 Frequency control word, high byte.
0xA7, // FREQ1 Frequency control word, middle byte.
0x62, // FREQ0 Frequency control word, low byte.
0x2D, // MDMCFG4 Modem configuration.
0x3B, // MDMCFG3 Modem configuration.
0x73, // MDMCFG2 Modem configuration.
0x22, // MDMCFG1 Modem configuration.
0xF8, // MDMCFG0 Modem configuration.
0x00, // CHANNR Channel number.
0x00, // DEVIATN Modem deviation setting (when FSKmodulation
is enabled).
0xB6, // FREND1 Front end RX configuration.
0x10, // FREND0 Front end RX configuration.
0x18, // MCSM0 Main Radio Control State Machine configuration.
0x1D, // FOCCFG Frequency Offset Compensation
Configuration.
0x1C, // BSCFG Bit synchronization Configuration.
0xC7, // AGCCTRL2 AGC control.
0x00, // AGCCTRL1 AGC control.
0xB2, // AGCCTRL0 AGC control.
0xEA, // FSCAL3 Frequency synthesizer calibration.
0x0A, // FSCAL2 Frequency synthesizer calibration.
0x00, // FSCAL1 Frequency synthesizer calibration.
0x11, // FSCAL0 Frequency synthesizer calibration.
0x59, // FSTEST Frequency synthesizer calibration.
0x88, // TEST2 Various test settings.
0x31, // TEST1 Various test settings.
0x0B, // TEST0 Various test settings.
0x0B, // IOCFG2 GDO2 output pin configuration.
0x06, // IOCFG0D GDO0 output pin configuration.
0x04, // PKTCTRL1 Packet automation control.
0x05, // PKTCTRL0 Packet automation control.
0x00, // ADDR Device address.
0x0c // PKTLEN Packet length.
};

//使用CC1100 发送数据

void halRfSendPacket(INT8U *txBuffer, INT8U size)
{
```

```
halSpiWriteReg(CCxxx0_TXFIFO, size);
halSpiWriteBurstReg(CCxxx0_TXFIFO, txBuffer, size); //
```

写入要发送的数据

```
halSpiStrobe(CCxxx0_STX); //进入发送模式发送数据
// Wait for GDO0 to be set -> sync transmitted while (!GDO0);
// Wait for GDO0 to be cleared -> end of packet while (GDO0);
halSpiStrobe(CCxxx0_SFTX);
}
```

//使用CC1100 接收数据

```
INT8U halRfReceivePacket(INT8U *rxBuffer, INT8U *length)
{
INT8U status[2];
INT8U packetLength;
halSpiStrobe(CCxxx0_SRX); //进入接收状态
while (!GDO1);
while (GDO1);
if ((halSpiReadStatus(CCxxx0_RXBYTES) & BYTES_IN_RXFIFO))
```

//如果接的字节数不为0

```
{
packetLength = halSpiReadReg(CCxxx0_RXFIFO);
if (packetLength <= *length) {
halSpiReadBurstReg(CCxxx0_RXFIFO, rxBuffer,
packetLength);
*length = packetLength; //把接收数据长度的修改为当
```

前数据的长度

```
// Read the 2 appended status bytes (status[0] = RSSI, status[1]
= LQI)
halSpiReadBurstReg(CCxxx0_RXFIFO, status, 2); // 读
```

出CRC 校验位

```
halSpiStrobe(CCxxx0_SFRX); //清洗接收缓冲区
return (status[1] & CRC_OK); //如果校验成功返
```

回接收成功

```
}
else
{
*length = packetLength;
halSpiStrobe(CCxxx0_SFRX); //清洗接收缓冲区
return 0;
}
}
else
return 0;
}
```