文档编号：AN1015

上海东软载波微电子有限公司

应用笔记

HW3000 Programming Guide

修订历史

版本	修订日期	修改概要
V1.0	2018-5-8	初版公开发布

地　址：中国上海市龙漕路 299 号天华信息科技园 2A 楼 5 层
邮　编：200235
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com

目 录

内容目录

第 1 章 SPI接口

1.1 SPI帧格式和通讯时序

SPI 通讯帧格式和时序，参见 HW3000 芯片数据手册的“SPI 通信接口”章节。

1.2 SPI读写函数原型

通用 MCU 端需要根据 SPI 读写时序，实现寄存器读写和 FIFO 读写功能函数，函数原型如下：

```
/*****************************************************************************
* 函数名称：hw3000_write_reg
* 功能描述：写 HW3000 寄存器
* 输入参数：addr   寄存器地址
            value  寄存器值
* 返回参数：无
*****************************************************************************/
void hw3000_write_reg(uint8_t addr, uint16_t value)

/*****************************************************************************
* 函数名称：hw3000_read_reg
* 功能描述：读 HW3000 寄存器
* 输入参数：addr   寄存器地址
* 返回参数：value  寄存器值
*****************************************************************************/
uint16_t hw3000_read_reg(uint8_t addr)

/*****************************************************************************
* 函数名称：hw3000_write_fifo
* 功能描述：写 HW3000 FIFO
* 输入参数：addr    FIFO 地址
            data    数据地址
            length  数据长度
*****************************************************************************/
void hw3000_write_fifo(uint8_t addr, uint8_t *data, uint8_t length)
```

```
/*****************************************************************************
* 函数名称：hw3000_read_fifo
* 功能描述：写 HW3000 FIFO
* 输入参数：addr    FIFO 地址
            data    数据地址
            length  数据长度
* 返回参数：无
*****************************************************************************/
void hw3000_read_fifo(uint8_t addr, uint8_t *data, uint8_t length)
```

1.3 HR7P系列MCU SPI汇编驱动

用户若采用上海东软载波微电子有限公司 HR7P 系列 MCU 作为主控，推荐采用以下 SPI 汇编驱动程序。

```
#define  PCSN    PB     //MCU 端口选择，用户需要初始化输入/输出属性
#define  PSCK    PA
#define  PMOSI   PB
#define  PMISO   PA

#define BCSN      0x00   //MCU 端口比特位选择，用户需要初始化输入/输出属性
#define BSCK      0x02
#define BMOSI     0x01
#define BMISO     0x03

#define TCSN      PBT0   //方向寄存器
#define TSCK      PAT2
#define TMOSI     PBT1
#define TMISO     PAT3

#define CSN       PB0
#define SCK       PA2
#define MOSI      PB1
#define MISO      PA3
```

```
void hw3000_write_reg(uint8_t addr, uint16_t value)
{
uint8_t i;
__asm
{
    MOVI    0x80
    IOR     (&addr&) % 0x80, 1
    BCC     PCSN, BCSN

    CLR     (&i&)   % 0x80

    JBC     (&addr&)   % 0x80, 7 ;write     addr
    BSS     PMOSI, BMOSI
    JBS     (&addr&)   % 0x80, 7
    BCC     PMOSI, BMOSI
    RL      (&addr&)   % 0x80, 1
    BSS     PSCK, BSCK
    INC     (&i&)   % 0x80, 1
    BCC     PSCK, BSCK
    MOVI    0x08
    XOR     (&i&)   % 0x80, 0
    JBS     PSW, Z
    GOTO    $-0x0B;

    CLR     (&i&)   % 0x80

    JBC     (&value&)   % 0x80 + 1, 7 ;write     value_h
    BSS     PMOSI, BMOSI
    JBS     (&value&)   % 0x80 + 1, 7
```

```
        BCC     PMOSI, BMOSI
        RL      (&value&)   % 0x80 + 1, 1
        BSS     PSCK, BSCK
        INC     (&i&)   % 0x80, 1
        BCC     PSCK, BSCK
        MOVI    0x08
        XOR     (&i&)   % 0x80, 0
        JBS     PSW, Z
        GOTO    $-0x0B

        CLR     (&i&)   % 0x80

        JBC     (&value&)   % 0x80, 7;write     value_l
        BSS     PMOSI, BMOSI
        JBS     (&value&)   % 0x80, 7
        BCC     PMOSI, BMOSI
        RL      (&value&)   % 0x80, 1
        BSS     PSCK, BSCK
        INC     (&i&)   % 0x80, 1
        BCC     PSCK,    BSCK
        MOVI    0x08
        XOR     (&i&)   % 0x80, 0
        JBS     PSW, Z
        GOTO    $-0x0B

        BSS     PCSN, BCSN
    }
    }

    uint16_t hw3000_read_reg(uint8_t addr)
```

```
{
    uint8_t i;
    uint16_t value;

__asm
{
    BCC     PCSN, BCSN

    CLR     (&i&)   % 0x80

    JBC     (&addr&)   % 0x80, 7 ;write     addr
    BSS     PMOSI, BMOSI
    JBS     (&addr&)   % 0x80, 7
    BCC     PMOSI, BMOSI
    RL      (&addr&)   % 0x80, 1
    BSS     PSCK, BSCK
    INC     (&i&)   % 0x80, 1
    BCC     PSCK, BSCK
    MOVI    0x08
    XOR     (&i&)   % 0x80, 0
    JBS     PSW, Z
    GOTO    $-0x0B;

    CLR     (&i&)   % 0x80

    BSS     PSCK, BSCK        ;read value
    BCC     PSW, C
    RL      (&value&)   % 0x80
    RL      (&value&)   % 0x80 + 1
```

```
        BCC     PSCK, BSCK
        JBC     PMISO, BMISO
        BSS     (&value&)   % 0x80, 0
        INC     (&i&)   % 0x80, 1
        MOVI    0x10
        XOR     (&i&)   % 0x80, 0
        JBS     PSW, Z
        GOTO    $-0x0B;

        BSS     PCSN, BCSN
    }

    return value;
}

void hw3000_write_fifo(uint8_t addr, uint8_t *data, uint8_t length)
{
uint8_t i, j;
uint8_t   value;

__asm
{
        MOVI    0x80
        IOR     (&addr&) % 0x80, 1

        BCC     PCSN, BCSN

        CLR     (&i&)   % 0x80

        JBC     (&addr&)   % 0x80, 7 ;write     addr
```

```
        BSS     PMOSI, BMOSI
        JBS     (&addr&)  % 0x80, 7
        BCC     PMOSI, BMOSI
        RL      (&addr&)  % 0x80, 1
        BSS     PSCK, BSCK
        INC     (&i&)  % 0x80, 1
        BCC     PSCK, BSCK
        MOVI    0x08
        XOR     (&i&)  % 0x80, 0
        JBS     PSW, Z
        GOTO    $-0x0B;
    }

    for (j = 0; j < length; j++) {
        value = data[j];

        __asm
        {
            CLR     (&i&)  % 0x80

            JBC     (&value&)  % 0x80, 7;write data[j]
            BSS     PMOSI, BMOSI
            JBS     (&value&)  % 0x80, 7
            BCC     PMOSI, BMOSI
            RL      (&value&)  % 0x80, 1
            BSS     PSCK, BSCK
            INC     (&i&)  % 0x80, 1
            BCC     PSCK, BSCK
            MOVI    0x08
```

```
            XOR     (&i&)   % 0x80, 0
            JBS     PSW, Z
            GOTO    $-0x0B
        }
    }

    __asm
    {
        BSS     PCSN, BCSN
    }
    }

    void hw3000_read_fifo(uint8_t addr, uint8_t *data, uint8_t length)
    {
    uint8_t i, j;
    uint8_t   value;

    __asm
    {
        BCC     PCSN, BCSN

        CLR     (&i&)   % 0x80

        JBC     (&addr&)   % 0x80, 7 ;write     addr
        BSS     PMOSI, BMOSI
        JBS     (&addr&)   % 0x80, 7
        BCC     PMOSI, BMOSI
        RL      (&addr&)   % 0x80
        BSS     PSCK, BSCK
        INC     (&i&)   % 0x80, 1
```

```
        BCC     PSCK, BSCK
        MOVI    0x08
        XOR     (&i&)   % 0x80, 0
        JBS     PSW, Z
        GOTO    $-0x0B;
    }

    for (j = 0; j < length; j++) {
        __asm
        {
            CLR     (&i&)   % 0x80

            BSS     PSCK, BSCK              ;read data
            BCC     PSW, C
            RL      (&value&)   % 0x80, 1
            BCC     PSCK, BSCK
            JBC     PMISO, BMISO
            BSS     (&value&)   % 0x80, 0
            INC     (&i&)   % 0x80, 1
            MOVI    0x08
            XOR     (&i&)   % 0x80, 0
            JBS     PSW, Z
            GOTO    $-0x0A;
        }
        data[j] = value;
    }

    __asm
    {
```

```
        BSS     PCSN, BCSN

    }
    }
```

第 2 章　　程序设计示例

以下为 HW3000 芯片常用收发程序示例。

2.1　HW3000 复位功能区别

芯片从 POWER DOWN 状态恢复至 IDLE 状态过程中内部 POR 将复位全芯片(通过设置 PDN 输入引脚为高电平进入 POWER DOWN 状态)。

芯片从 DEEP SLEEP/SLEEP 恢复至 IDLE 状态，芯片除配置寄存器外将全部被复位（注意寄存器中的状态指示位将被复位）。

芯片共提供两种软复位方式，分别为 SFT_RST0（0x60）和 SFT_RST1（0x61），其中：

1) SFT_RST0 复位硬件电路与 FIFO 而保留原有的寄存器设置值

2) SFT_RST1 进行全芯片复位，寄存器同时也会被复位成默认值

2.2　HW3000 默认寄存器初始化

HW3000 内部寄存器分为 Bank0 和 Bank1 两部分，其中 Bank0 是面向用户开放功能寄存器，Bank0 寄存器由用户根据功能进行设置；Bank1 是内部校准寄存器，Bank1 寄存器仅需用户在寄存器复位后按照建议值初始化即可。Bank0 和 Bank1 切换方式详见如下 HW3000 初始化流程，带宏定义寄存器均为 Bank0 寄存器，hw3000_mode_t 结构体用于演示各种用户功能模式设置过程。

```
typedef struct {
    osc_mode_t          osc;        //osc selection
    frequency_band_t    band;       //freq band setting
    symbol_rate_t       rate;       //data rate setting
    frequency_mode_t    freq_mode;  //0 only for 433MHz 1 direct mode for all
    int8_t              power;      //tx power set
    frame_mode_t        frame_mode; //0 fifo mode, 1 frame mode
    uint8_t             ack_mode;   //0 disable, 1 enable
    uint8_t             tx_mode;    //0 tx length by LEN0_PKLEN, 1 by fifo pointer
    uint8_t             rx_mode;    //0 automatic 1 by LEN0_PKLEN
    uint8_t             lp_enable;  //0 low power mode disable, 1 enable
} hw3000_mode_t;
```

```
uint8_t i;
uint16_t agc_table[16] = { 0x1371,0x1351,0x0B51,0x2F71,0x2F51,0x2E51,
                           0x2E31,0x4B31,0x4731,0x4631,0x4531,0x4431,
                           0x6131,0x6031,0x6011,0x6009
```

```
                                    };
    uint16_t reg_val;
    hw3000_write_reg(0x4C, 0x5555);     //Bank 0 open by default
    while(!(hw3000_read_reg(INTFLAG) & 0x4000)); //wait for chip ready

    hw3000_write_reg(0x4C, 0x55AA);   //Bank 1 open
    for (i = 0; i < 16; i++) {
        hw3000_write_reg(0x1B + i, agc_table[i]);
    }
    hw3000_write_reg(0x03, 0x0508);
    hw3000_write_reg(0x11, 0xC630);
    switch (hw3000_mode.rate) {
    case SYMBOL_RATE_1K:
        hw3000_write_reg(0x14, 0x1935);
        hw3000_write_reg(0x40, 0x0008);
        hw3000_write_reg(0x41, 0x0010);
        hw3000_write_reg(0x42, 0x82D8);
        hw3000_write_reg(0x43, 0x3D38);
        break;
    case SYMBOL_RATE_10K:
        hw3000_write_reg(0x14, 0x1935);
        hw3000_write_reg(0x40, 0x0008);
        hw3000_write_reg(0x41, 0x0010);
        hw3000_write_reg(0x42, 0x82D8);
        hw3000_write_reg(0x43, 0x3D38);
        break;
    case SYMBOL_RATE_19K2:
        hw3000_write_reg(0x14, 0x1915);
        break;
```

```
    case SYMBOL_RATE_38K4:
        hw3000_write_reg(0x14, 0x1915);
        break;
    case SYMBOL_RATE_50K:
        hw3000_write_reg(0x14, 0x1915);
        break;
    case SYMBOL_RATE_100K:
        hw3000_write_reg(0x14, 0x1915);
        break;
    default:
        break;
    }
    switch (hw3000_mode.band) {
    case FREQ_BAND_315MHZ:
        hw3000_write_reg(0x17, 0xF223);
        break;
    case FREQ_BAND_433MHZ:
        hw3000_write_reg(0x17, 0xF6C2);
        break;
    case FREQ_BAND_779MHZ:
        hw3000_write_reg(0x17, 0xFB61);
        break;
    case FREQ_BAND_868MHZ:
        hw3000_write_reg(0x17, 0xFB61);
        break;
    case FREQ_BAND_915MHZ:
        hw3000_write_reg(0x17, 0xFB61);
        break;
    default:
```

```
            break;
        }
        hw3000_write_reg(0x51, 0x001B);
        hw3000_write_reg(0x55, 0x8003);
        hw3000_write_reg(0x56, 0x4155);
        hw3000_write_reg(0x62, 0x70ED);

        hw3000_write_reg(0x4C, 0x5555);     //Bank 0 open
        hw3000_write_reg(INTIC, 0x8000);    //clear por int
        hw3000_write_reg(MIXFW, 0x2E35);       //must modify this value
        hw3000_write_reg(MODECTRL, 0x100F);       //must close GPIO clock for noise reason
        /*osc set*/
        if (hw3000_mode.osc == OSC20MHZ) {
            hw3000_write_reg(MODEMCTRL, 0x5201); //20MHz osc select
        } else {
            hw3000_write_reg(MODEMCTRL, 0x1201); //26MHz osc select
        }
        /*frequency set*/
        if (hw3000_mode.freq_mode == DEFAULT && hw3000_mode.band ==
FREQ_BAND_433MHZ) {
            hw3000_write_reg(RFCFG, 0x3312);
            hw3000_write_reg(FREQCFG0, 0x30EA); //default mode only for 433MHz
        } else {
            hw3000_freq_set(hw3000_mode); //from deep sleep/sleep need reconfigure this
reg
        }
        /*data rate*/
        hw3000_rate_set(hw3000_mode);
        /*tx power set*/
```

```
hw3000_power_set(hw3000_mode);
/*rx power set*/
if (hw3000_mode.lp_enable == ENABLE) { //HOP_TIMER/(HOP_TIMER+LP_TIMER)
    reg_val = hw3000_read_reg(MODEMCTRL);
    reg_val &= 0xF0FF;
    reg_val |= 0x0200;      //LP_TIMER set
    reg_val |= 0x0004;      //LP_ENABLE
    hw3000_write_reg(MODEMCTRL, reg_val);
    reg_val = hw3000_read_reg(HOPCFG);
    reg_val &= 0xFF0F;
    reg_val |= 0x0060;      //HOP_TIMER set
    hw3000_write_reg(HOPCFG, reg_val);
}
/*frame set*/
hw3000_frame_set(hw3000_mode);
```

2.3 数据载荷说明

HW3000 收发数据 FIFO 深度为 256，若用户采用增强型帧结构，用户可用 FIFO 深度为 252 字节，若用户采用直接 FIFO 模式，通过半空半满中断可支持用户超过 256 字节数据传输。

2.4 HW3000 TX测试波形

HW3000 提供单载波、PN9、0101 等数据发送模式，以方便频点测试与发送功率测试。

```
void hw3000_tx_sw(void)
{
hw3000_write_reg(PKTCTRL, 0xE000); //enable single carrior mode
hw3000_write_reg(DEVIATION, 0x0000); //deviation 0 Hz
hw3000_write_reg(TRCTRL, 0x0100); //tx enable
hw3000_write_reg(FIFODATA, 0x5555);
hw3000_write_reg(FIFOCTRL, 0x0001); //ocpy = 1
}
```

```
void hw3000_cancel_sw(hw3000_mode_t hw3000_mode)
{
uint16_t reg_val;
hw3000_write_reg(FIFOCTRL, 0x0000); //ocpy = 0
hw3000_write_reg(TRCTRL, 0x0000); //tx disable
if (hw3000_mode.frame_mode == FRAME) {  //增强型帧结构
    hw3000_write_reg(PKTCTRL, 0xC000);
} else {   //直接 FIFO 模式帧结构
    if (hw3000_mode.tx_mode == 0) {
        reg_val = 0x0000;
    } else {
        reg_val = 0x4000;
    }
    if (hw3000_mode.rx_mode == 1) {
        reg_val |= 0x0100;
    }
    hw3000_write_reg(PKTCTRL, reg_val);
}
hw3000_rate_set(hw3000_mode);
}

void hw3000_tx_pn9(void)
{
hw3000_write_reg(MODECTRL, 0x150F); //DERECT_MOD REP_MOD
hw3000_write_reg(TRCTRL, 0x0100); //tx_enable
delay_ms(1);
hw3000_write_reg(MODECTRL, 0x170F);
}
```

```
void hw3000_cancel_pn9(void)
{
 hw3000_write_reg(TRCTRL, 0x0000); //tx_disable
 hw3000_write_reg(MODECTRL, 0x100F);
}

void hw3000_tx_0101(void)
{
 hw3000_write_reg(MODECTRL, 0x190F); //DERECT_MOD REP_MOD
 hw3000_write_reg(TRCTRL, 0x0100); //tx_enable
 delay_ms(1);
 hw3000_write_reg(MODECTRL, 0x1B0F);
}

void hw3000_cancel_0101(void)
{
 hw3000_write_reg(TRCTRL, 0x0000); //tx_disable
 hw3000_write_reg(MODECTRL, 0x100F);
}
```

2.5 HW3000 TX 数据

Hw3000 支持增强型帧结构发射数据和直接 FIFO 模式发射数据，其中直接 FIFO 模式需要用户自行完成数据校验工作，直接 FIFO 模式可实现与市面同类无线芯片的互联互通。

```
 int8_t hw3000_frame_tx(hw3000_mode_t hw3000_mode, hw3000_data_t *txbuf)
 {
     uint16_t reg_val;
     if (txbuf->len > 252) {
         return -1;
```

```
    }
    if (_hw3000_state != TX) {
        _hw3000_state = TX;

        hw3000_write_reg(TRCTRL, 0x0100);      //tx_enable
        hw3000_write_reg(FIFOSTA, 0x0100); //flush fifo
        hw3000_write_fifo(FIFODATA, txbuf->data, txbuf->len); //write fifo
        hw3000_write_reg(FIFOCTRL, 0x0001); //ocpy = 1

        _hw3000_irq_request = 0;
        while (!_hw3000_irq_request); //wait for send finish
        _hw3000_irq_request = 0;
        _hw3000_state = IDLE;

        if (hw3000_mode.ack_mode == ENABLE) {
            reg_val = hw3000_read_reg(INTFLAG);
            if (reg_val & 0x0002) {
                hw3000_write_reg(FIFOCTRL, 0x0000);   //ocpy = 0
                hw3000_write_reg(INTIC, 0x0001);        //clr_int
                hw3000_write_reg(TRCTRL, 0x0000);      //send disable
                return -1;
            }
        }
        hw3000_write_reg(FIFOCTRL, 0x0000); //ocpy = 0
        hw3000_write_reg(INTIC, 0x0001);  //clr_int
        hw3000_write_reg(TRCTRL, 0x0000);      //send disable

        return 0;
    }
```

```
        return -1;
    }

    int8_t hw3000_fifo_tx(hw3000_mode_t hw3000_mode, hw3000_data_t *txbuf)
    {
        if (_hw3000_state != TX) {
            _hw3000_state = TX;
            hw3000_write_reg(TRCTRL, 0x0100);      //tx_enable
            hw3000_write_reg(FIFOSTA, 0x0108);    //flush fifo
            if (hw3000_mode.tx_mode == 0) {
                hw3000_write_reg(LEN0PKLEN, txbuf->len);
            }
            if (txbuf->len > 256) {
                hw3000_write_fifo(FIFODATA, txbuf->data, 256); //write fifo
                txbuf->len -= 256;
            } else {
                hw3000_write_fifo(FIFODATA, txbuf->data, txbuf->len); //write fifo
                txbuf->len = 0;
            }
            txbuf->idx = 0;
            hw3000_write_reg(FIFOCTRL, 0x0001);   //ocpy = 1
            _hw3000_irq_request = 0;
            _hw3000_gpio2_request = 0;
            while (1) {
                if (_hw3000_gpio2_request == 1 && txbuf->len != 0) {
                    _hw3000_gpio2_request = 0;
                    if (txbuf->len > 128) {
                        hw3000_write_fifo(FIFODATA,
    &txbuf->data[256+128*txbuf->idx++], 128);      //write fifo
                        txbuf->len -= 128;
```

```
                    } else {
                        hw3000_write_fifo(FIFODATA,
&txbuf->data[256+128*txbuf->idx], txbuf->len); //write fifo
                        txbuf->len = 0;
                    }
                }
                if (_hw3000_irq_request == 1) {      //wait for send finish
                    _hw3000_irq_request = 0;
                    _hw3000_state = IDLE;
                    hw3000_write_reg(FIFOCTRL, 0x0000); //ocpy = 0
                    hw3000_write_reg(INTIC, 0x0001); //clr_int
                    hw3000_write_reg(TRCTRL, 0x0000); //send disable
                    return 0;
                }
            }
        }
        return -1;
    }
```

2.6 HW3000 RX 数据

Hw3000 支持增强型帧结构发射数据和直接 FIFO 模式接收数据，其中直接 FIFO 模式需要用户自行完成数据校验工作，直接 FIFO 模式可实现与市面同类无线芯片的互联互通。

```
int8_t hw3000_rx_enable(void)
{
    if (_hw3000_state != TX) {
        hw3000_write_reg(TRCTRL, 0x0080); //enable rx
        _hw3000_state = RX;
        _hw3000_irq_request = 0;
        _hw3000_gpio1_request = 0;
        return 0;
```

```
        }
        return -1;
    }

    void hw3000_rx_disable(void)
    {
        hw3000_write_reg(TRCTRL, 0x0000); //rx_disable
        hw3000_write_reg(INTIC, 0x0001); //clr_int
        _hw3000_state = IDLE;
        _hw3000_irq_request = 0;
        _hw3000_gpio1_request = 0;
    }

    int8_t hw3000_rx_data(hw3000_mode_t hw3000_mode, hw3000_data_t *rxbuf)
    {
        uint16_t reg;
        if (_hw3000_state == RX) {
            if (hw3000_mode.frame_mode == FRAME) {
                reg = hw3000_read_reg(FIFOCTRL);
                if (!(reg & 0xC000)) { //PHR CRC check
                    reg = hw3000_read_reg(RXPHR0);
                    rxbuf->len = ((reg >> 8) - 3) & 0x00FF;
                    hw3000_read_fifo(FIFODATA, rxbuf->data, rxbuf->len);
                    hw3000_rx_disable();
                    hw3000_rx_enable();
                    return 0;
                }
            } else {
                hw3000_read_fifo(FIFODATA, rxbuf->data, 2);
```

```
                rxbuf->len = rxbuf->data[0] * 256 + rxbuf->data[1];
                if (rxbuf->len > 2048) {
                    rxbuf->len = 2048;
                }
                rxbuf->idx = 0;
                if (hw3000_mode.rx_mode == 1) {
                    hw3000_write_reg(LEN0PKLEN, rxbuf->len);
                }
                rxbuf->len -= 2;
                while (1) {
                    if (_hw3000_gpio1_request == 1) {
                        _hw3000_gpio1_request = 0;
                        hw3000_read_fifo(FIFODATA, &rxbuf->data[2 + 128 *
rxbuf->idx++], 128);
                        rxbuf->len -= 128;
                    }
                    if (_hw3000_irq_request == 1) {
                        _hw3000_irq_request = 0;
                        hw3000_read_fifo(FIFODATA, &rxbuf->data[2 + 128 *
rxbuf->idx], rxbuf->len);
                        rxbuf->len = 0;
                        break;
                    }
                }
            }
            hw3000_rx_disable();
            hw3000_rx_enable();
            return 0;
        } else {
```

```
            return -1;
        }
    }
```

2.7 HW3000 RSSI说明

在接收模式时，芯片会评估天线端接收信号能量的大小，该数值会保存在寄存器 RSSI（0x23）中。RSSI 的读数单位为 dBm，数据的格式为二进制补码形式的符号数。在 RSSI 寄存器里提供两个 RSSI 读数值，其中 RSSI1 保存的是上一个有效数据包（SFD 正确同步）的 RSSI 计算值，而 RSSI2 中保存的是实时的 RSSI 计算值，可用于 CSMA/CA 工作。

若需检测环境能量 RSSI2，建议在 RX 接收使能后至少延时 350us，再进行读取。另外如果用户系统 SPI 速率较快，大于 1Mbps 以上，在读取环境能量前，如有接收使能的关闭、打开操作，在接收使能切换中间必须有一定的延时操作，否则读取的 RSSI2 值会有跳变现象；同理，如果在读取 RSSI2 操作前，有发射、接收使能的切换操作，在状态切换中间必须加一定的延时操作。

直接 FIFO 模式下，若读取上一个有效数据包的 RSSI1 值，必须在 SFD_INT 与 FIFO_INT 之间读取，即在同步之后，数据接收完成之前读取，数据接收完成之后， RSSI1 被复位成 0x81。

增强型模式下，RSSI1 值会被锁定，直到下一个有效数据包之后才会被更新，且只有在芯片复位之后，RSSI1 值才会被复位成 0x81。

2.8 HW3000 状态查询模式特别说明

当通过 SPI 轮询方式检测 HW3000 收发状态是否完成时，在轮询相关状态寄存器时请加入适当延时，以避免 SPI 通信所产生的高次谐波可能对收发过程产生影响。

采用 IRQ 中断方式实现收发状态检测则无此问题产生。

2.9 HW3000 FIFO数据在发射完成后将驻留

若发射相同数据，可仅写入 FIFO 一次数据通过清除 INT 中断方式实现循环发送。

利用该方式可更好实现有源 RFID 功能。

若存在接收数据，发射端 FIFO 将不再保留。

2.10 HW3000 AFC及晶振的校准功能

HW3000 在接收端提供载波频偏自动补偿功能（AFC），可通过 AFC_EN（0x25）寄存器使能

晶振校准寄存器为 XOSC_CAL(0x37)，设置值支持 0x00 至 0xFF，步长约 15Hz。