
RF-2400 Series Module Application Guide

——AVR-controlled based RF communications

V 1.0

Content

1	Overview.....	- 1 -
2	Hardware configuration	- 2 -
2.1	RF-2400 wireless module.....	- 2 -
2.2	ATMEGA324PV-controlled typical circuit diagram	- 4 -
3	Software Structure.....	- 5 -
3.1	Overall software structure	- 5 -
3.2	System initiation configure module	- 5 -
3.2.1	System initiation flow chart	- 6 -
3.2.2	Interface,timer/counter Configuration.....	- 6 -
3.3	Bk2421 initialization process	- 7 -
3.3.1	BANK0 register configuration	- 7 -
3.3.2	BANK1 register configuration	- 9 -
3.4	SPI Communications.....	- 10 -
3.4.1	SPI timing.....	- 10 -
3.4.2	SPI software implementatio	- 12 -
3.5	RF communication module	- 12 -
3.5.1	RF data receiving process	- 12 -
3.5.3	Power measurement Module.....	- 18 -
3.6	References.....	- 21 -

1 Overview

This document is based on 2.4G RF wireless demo board, which is consisted by ATMEGA324PV chip, RF-2400 (BK2421 chip) module, and some peripheral devices , combined with the sample code to describe how to implement RF data communication process . This document gives some reference to 2.4G wireless applications project , and some introductory guide in wireless applications.

www.inhaos.com

2 Hardware configuration

The demo board hardware uses ATmega324pv chip plus independent BK2421 wireless modules and basic peripheral .As RF circuit design is demanding than the average, to simplify the design of the demo board, we using the wireless module. The demo board uses RF-2400 wireless module. RF-2400 module package both the BK2421 2.4G radio frequency chip and its peripheral circuits required. It's small and convenient (can be directly used on the circuit board).

2.1 RF-2400 wireless module

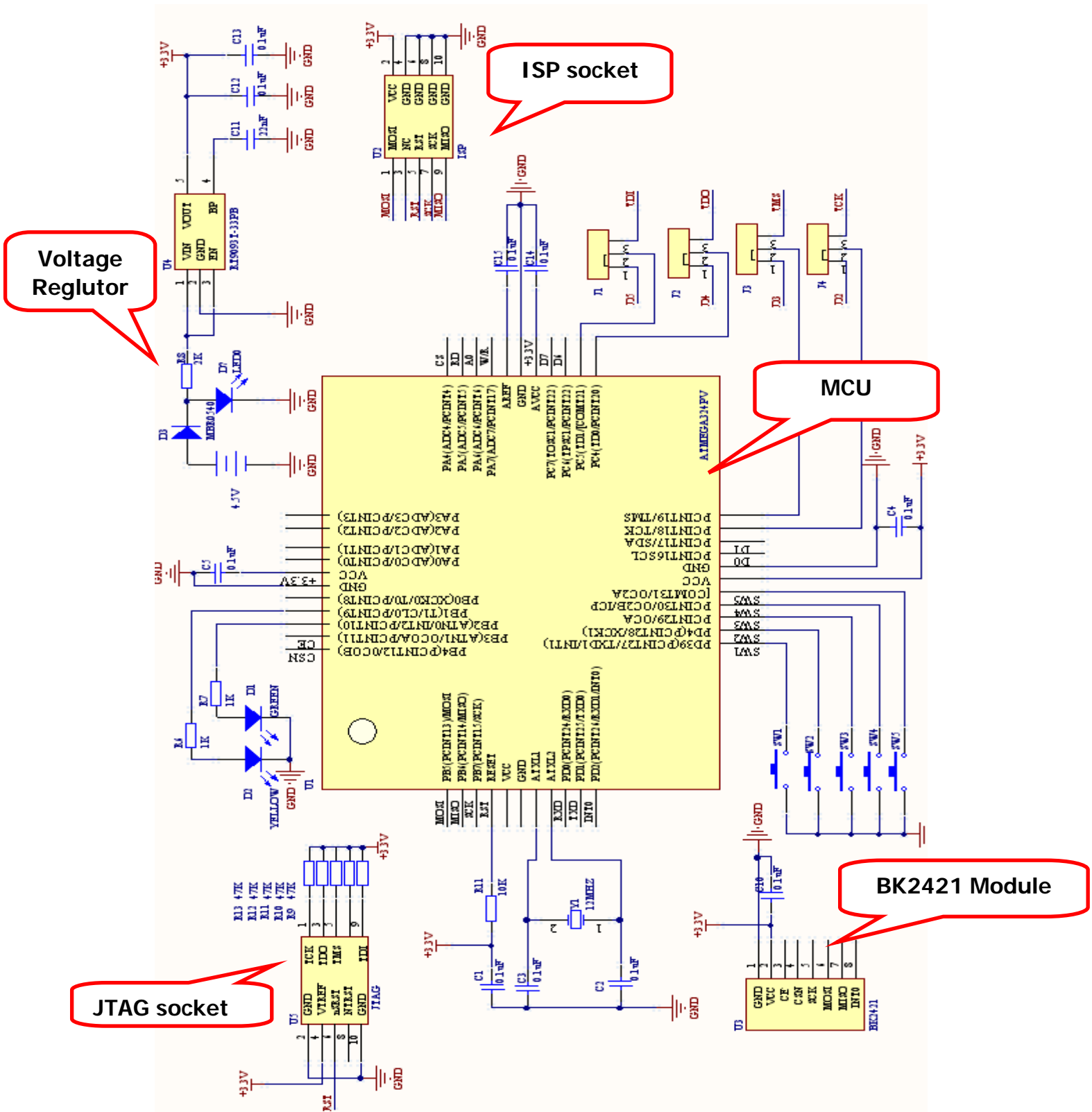
RF-2400 practicality pictures:



RF-2400 wireless module circuit diagram



2.2 ATMEGA324PV-controlled typical circuit diagram



3 Software Structure

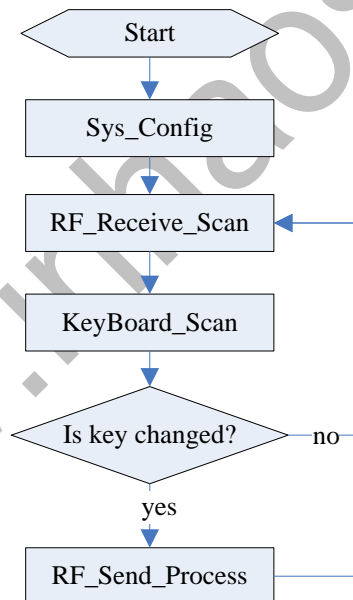
Software functions:

- When press SWn in A(B),transmitting start, yellow light would bright if transmitting fail , otherwise no any light.
- B Receive the other key values in B(A),and LED would bright (green)if receiving successfully , otherwise no any lights.

3.1 Overall software structure

Overall software structure includes: System initiation configure module , key scan module, RF receiving processing module , and RF transmitting processing module.

Software flow chart:



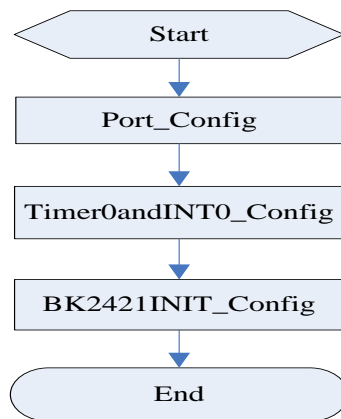
(Diagram 04)

3.2 System initiation configure

System initiation includes:

- ◆ Initial configuration of the IO ports;
- ◆ Open Timer0 and configure overflow period 1ms;
- ◆ Enable external interrupt, respond to IRQ signal to the RF chip;
- ◆ Initialize the Bank0 and Bank1 registers of Bk2421;

3.2.1 System initiation flow chart



(Diagram 05)

3.2.2 Interface, timer/counter Configuration

- Port configuration: configure PB1 ~ PB5, PB7 as Push-Pull One output, PB0, PB6 as an input (internal pull-high),; will, PORTD port configured as an input (internal pull-up resistor);and configure PORTD as input(internal pull-up resistor).

```
/******  
*Funcname:          void Sys_Config( void )  
*Remarks:          Port Configuration  
*****/  
void Sys_Config( void )  
{  
    PORTB = 0XFF;  
    DDRB  = 0XBE;    //Configuration data direction register;  
    PORTD = 0XFF;  
    DDRD  = 0X00;  
}
```

- Timer config:

Use timer0, mode1, timeout 1ms;

```
#define T0_1MS_OVF_INITIAL 0xA3 // = 256 - X
```

```
/******  
*Funcname:          void Timer0Init ( void )  
*Remarks:          timero Configuration  
*****/  
void Timer0Init( void )  
{  
    TCCR0A = 0;  
    TCCR0B = 0;  
    TIMSK0 = 0X01;    // enable T0 overflow interrupt  
    TCNT0 = T0_1MS_OVF_INITIAL;  
    TCCR0B = 0X03;    // Ft0 = CLKio/64    = 12MHz / 64  
}
```


- External interrupt configurations: external interrupt 0, falling edge trigger.

```
/******  
*Funcname:          void Extern intinit ( void )  
*Remarks:          Extern interrupt Configuration  
*****  
void ExternIntInit( void )  
{  
    EIFR = 0;           //Extern interrupt flag register  
    PCICR = 0;  
    PCIFR = 0;  
    PCMSK3 = 0;  
    PCMSK2 = 0;  
    PCMSK1 = 0;  
    EICRA = 0x02;       //INT0 falling edger active  
    EIMSK = 0X01;       //INT0 enable  
}
```

3.3 Bk2421 initialization process

This example will configure BK2421 data transfer rate 1Mbps, 0dbm transmitting, open the auto answer of channel 0 ,set the transmitting/receiving address width 5 bytes, open open the automatic repeat function(750us delay,5 re-transmitting) , 2 bytes check, RX mode initial configuration .

Note:

It's free to configure Bank0 and Bank1 , but you have to switching back Bank0 after configuration. Bank0 and Bank1 switching: first read status value of Bank0_Reg07_Rbank bit , if Rbank bit is 0 ,meaning Bank0 status currently , otherwise Bank1 status. Bank0 and Bank1 switching is achieved by SPI writing "ACTIVATE +0 x53" command.

3.3.1 BANK0 register configuration

Bank0 Register configuration description: from low byte to high byte. Each byte then from high to low reading and writing. Each of byte reads and writes for high to low.

Bank0 register configuration

Bank0 register	Configuration values	Remarks
CONFIG	0x0F	RX Mode, 2 bytes check
EN_AA	0x01	Auto-answer feature of enable data channels 0
EN_RXADDR	0x01	Channel 0 as the RX Address
SETUP_AW	0x03	RX / TX address width 5 bytes
SETUP_RETR	0x25	Automatic retransmitting delay 750us, 5 transmitting count

INHAOS RF-2400 Series Module Application Guide -AVR

RF_CH	0x20	Channel selection 32
RF_SETUP	0x15	Transfer rate of 1Mbps, transmitting power 0dbm
STATUS	0x70	Clear interrupt bit
OBSERVE_TX	0x00	Default
CD	0x00	Default
RX_ADD_P0	0x3A,0x3B,0x3C,0x3D,0x01	Receiving address of channel 0
RX_ADD_P2	0xC3	Default
RX_ADD_P3	0xC4	Default
RX_ADD_P4	0xC5	Default
RX_ADD_P5	0xC6	Default
TX_ADDR	0x3A,0x3B,0x3C,0x3D,0x01	Launch address
RX_PW_P0	0x20	Effective length of RX is 32 bytes in Channel 0
RX_PW_P1	0x20	Effective length of RX is 32 bytes in Channel 1
RX_PW_P2	0x20	Effective length of RX is 32 bytes in Channel 2
RX_PW_P3	0x20	Effective length of RX is 32 bytes in Channel 3
RX_PW_P4	0x20	Effective length of RX is 32 bytes in Channel 4
RX_PW_P5	0x20	Effective length of RX is 32 bytes in Channel 5
FIFO_STATUS	0x11	RX/TX FIFO Empty
DYNPD	0x01	Enable dynamic effective length of channel0
FEATURE	0x04	Enable dynamic effective length

(Diagram 06)

sample code is as follows:

```

Bank0 Registers Config=====
code volatile UINT8 Bank0_Reg_Init[21][2] = {
{ CONFIG,      0x0F }, { EN_AA,      0x01 }, { EN_RXADDR, 0x01 },
{ SETUP_AW,    0x03 }, { SETUP_RETR, 0x25 }, { RF_CH,     0x20 },
{ RF_SETUP,    0x15 }, { STATUS,     0x70 }, { OBSERVE_TX, 0x00 },
{ CD,          0x00 }, { RX_ADDR_P2, 0xc3 }, { RX_ADDR_P3, 0xc4 },
{ RX_ADDR_P4,  0xc5 }, { RX_ADDR_P5, 0xc6 }, { RX_PW_P0,   0x20 },
{ RX_PW_P1,    0x20 }, { RX_PW_P2,   0x20 }, { RX_PW_P3,   0x20 },
{ RX_PW_P4,    0x20 }, { RX_PW_P5,   0x20 }, { FIFO_STATUS, 0x11 }
};
RX/TX Address=====
const UINT8 RX_Address[5] = { 0x3a, 0x3b, 0x3c, 0x3d, 0x01 };
const UINT8 TX_Address[5] = { 0x3a, 0x3b, 0x3c, 0x3d, 0x01 };

const UINT8 Bank0_RegAct[2][2] = { { DYNPD, 0x01 }, { FEATURE, 0x04 } };
/*****
*Funcname:      void Bank0_Register_Init( void );
*Remarks:      Bank0 Registers Configuration,RX and TX Address Setting;
*****/
void Bank0_Register_Init( void )
{
    UINT8 i = 0;
    UINT8 k = 0;
    UINT8 Rt = 0;

```

INHAOS RF-2400 Series Module Application Guide -AVR

```
//Bank0 Register Configuration Operate=====
for( i = 0; i < 21; i++ )
{
    SPI_Write_Reg( W_REGISTER | Bank0_Reg_Init[i][0], Bank0_Reg_Init[i][1] );
    Rt = SPI_Read_Reg( Bank0_Reg_Init[i][0] );
}
//Write RX Pipes0 Address=====
SPI_Write_Buff( W_REGISTER | RX_ADDR_P0, &RX_Address[0], 5 );
//Write TX Address=====
SPI_Write_Buff( W_REGISTER | TX_ADDR, &TX_Address[0], 5 );
//Active Ship Operate=====
k = SPI_Read_Reg( FEATURE );
if( k == 0 )
{
    SPI_Write_Reg( ACTIVATE, 0X73 );
}
for( i = 0; i < 2; i++ )
{
    SPI_Write_Reg( W_REGISTER | Bank0_RegAct[i][0], Bank0_RegAct[i][1] );
    SPI_Read_Reg( Bank0_RegAct[i][0] );
}
}
```

3.3.2 BANK1 register configuration

Reg00 to Reg08 read-write order of Bank1: From high byte to low byte, and high bit to low bit of each byte. Reg09 to Reg0E read-write: low byte to high byte, high bit to low bit of each byte.

Bank1 Register configuration

Bank0 Register Number	Configuration values
Reg00	0xE2014B40
Reg01	0x00004BC0
Reg02	0x028CFCD0
Reg03	0x41390099
Reg04	0x0B869ED9
Reg05	0xA67F0624
Reg06	0x00000000
Reg07	0x00000000
Reg08	0x00000000
Reg09	0x00000000
Reg0A	0x00000000
Reg0B	0x00000000
Reg0C	0x00127300
Reg0D	0x36B480000
Reg0E	0x412008048120CFF7FEFFFF

(Diagram 07)

Sample code is as follows:

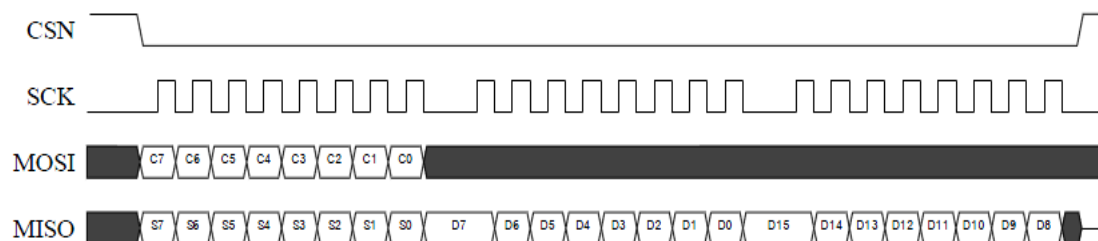
```
code volatile UINT32 Bank1_Reg0_Reg13[]={
0xE2014B40, 0x00004BC0, 0x028CFCD0, 0x41390099, 0x0B869ED9, 0xA67F0624,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00127300, 0x36B48000};
code volatile UINT8 Bank1_Reg14[11] = { 0X41, 0X20, 0X08, 0X04, 0X81, 0X20, 0XCF,
0XF7, 0XFE, 0XFF, 0XFF};
/*****
*Funcname: void Bank1_Register_Init( void );
*Remarks: Bank1_Register Initialize Configuration:
*****/
void Bank1_Register_Init( void )
{
    INT8 i = 0;
    UINT8 j = 0;
    UINT8 Buff[4] = {0};
    //Configuration Bank1 Register0 to Register8 =====
    for( i = 0; i < 9; i++ )
    {
        for( j = 0; j < 4; j++ )
        {
            Buff[j] = (UINT8)(( Bank1_Reg0_Reg13[i] >> ( 8 *(j) ) ) & 0xff);
        }
        SPI_Write_Buff( W_REGISTER | i, &(Buff[0]), 4 );
    }
    //Configuration Bank1 Register9 to Register13=====
    for( i = 9; i < 14; i++ )
    {
        for( j = 0; j < 4; j++ )
        {
            Buff[j] = (UINT8)( Bank1_Reg0_Reg13[i] >> 8 * (3-j) & 0xff );
        }
        SPI_Write_Buff( W_REGISTER | i, &(Buff[0]), 4 );
    }
    //Configuration Bank1 Register 14=====
    SPI_Write_Buff( W_REGISTER | 0x0e, &(Bank1_Reg14[0]), 11 );
    //toggle Reg4[25-26]=====
    for( i = 0; i < 4; i++ )
    {
        Buff[i] = (UINT8)(( Bank1_Reg0_Reg13[4] >> 8*(i)) & 0xff);
    }
    Buff[0] |= 0x06;
    SPI_Write_Buff( W_REGISTER | 0X04, &(Buff[0]), 4 );
    Buff[0] &= 0XF9;
    SPI_Write_Buff( W_REGISTER | 0X04, &(Buff[0]), 4 );
}
```

3.4 SPI Communications

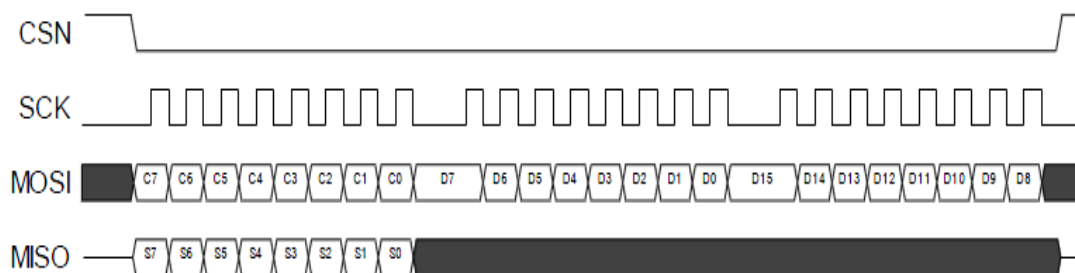
3.4.1 SPI timing

As AT8C51 without SPI, so the reader module is using software to simulate the SPI . SPI module includes the operation of read and write register.

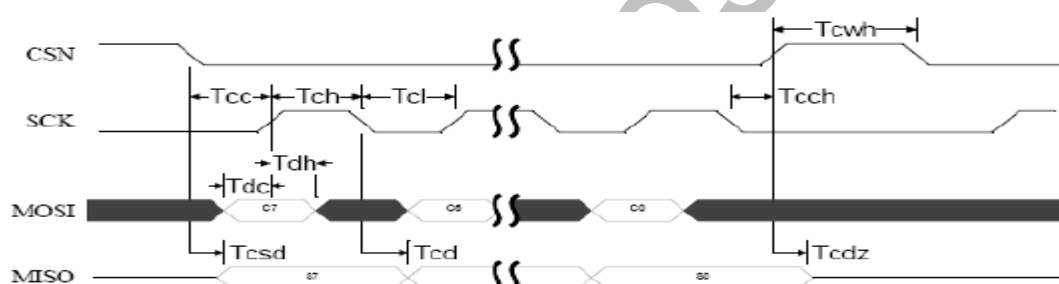
INHAOS RF-2400 Series Module Application Guide -AVR



SPI read operation (Diagram 08)



SPI write operation (Diagram 09)



SPI timing (Diagram 10)

Symbol	Parameters	Min	Max	Units
Tdc	Data to SCK Setup	10		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		38	ns
Tcd	SCK to Data Valid		55	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	8	MHZ
Tr, Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Hold	2		ns
Tcch	CSN to CSN Hold	2		ns
Tcwh	CSN Inactive Time	50		ns
Tcdz	CSN to Output High Z		38	ns

SPI timing parameter list (Diagram 11)

3.4.2 SPI software implementation

This module mainly includes: BK2421 register operation of reading one or multiple byte, writing one or multiple byte to BK2421 register (see the source code for program of reading/writing multiple byte).

Implement SPI byte write and read operations, sample code is as follows:

```
*****
*Funcname:   UINT8 SPI_Write_Bt( UINT8 pByte );
*Remarks:   Write a Byte to Register, Return Slave Register's Value:
*****/
UINT8 SPI_Write_Bt( UINT8 pByte )
{
    UINT8 i = 0x80;
    UINT8 Len = 8;
    UINT8 Rt = 0;
    CLR_SCK();
    while( Len-- )
    {
        if( pByte & i )
        {
            SET_MOSI();
        }
        else
        {
            CLR_MOSI();
        }
        SET_SCK();           //Rising Edge Send Data;
        SET_MIOS();
        if( GET_MIOS() )
        {
            Rt |= i;
        }
        CLR_SCK();
        i >>= 1;
    }
    return( Rt );
}
```

3.5 RF communication module

This module includes RF receiving and transmitting process. Button to start , successful launch will generate corresponding interrupt , TX_DS will be set high. If reaching max retransmitting, MAX_RT will be set to high , show yellow light and launch failed. Successful receiving and received length is of 5 bytes , RX_DR will be set to high, and show green light.

3.5.1 RF data receiving process

RF data receiving mechanisms:

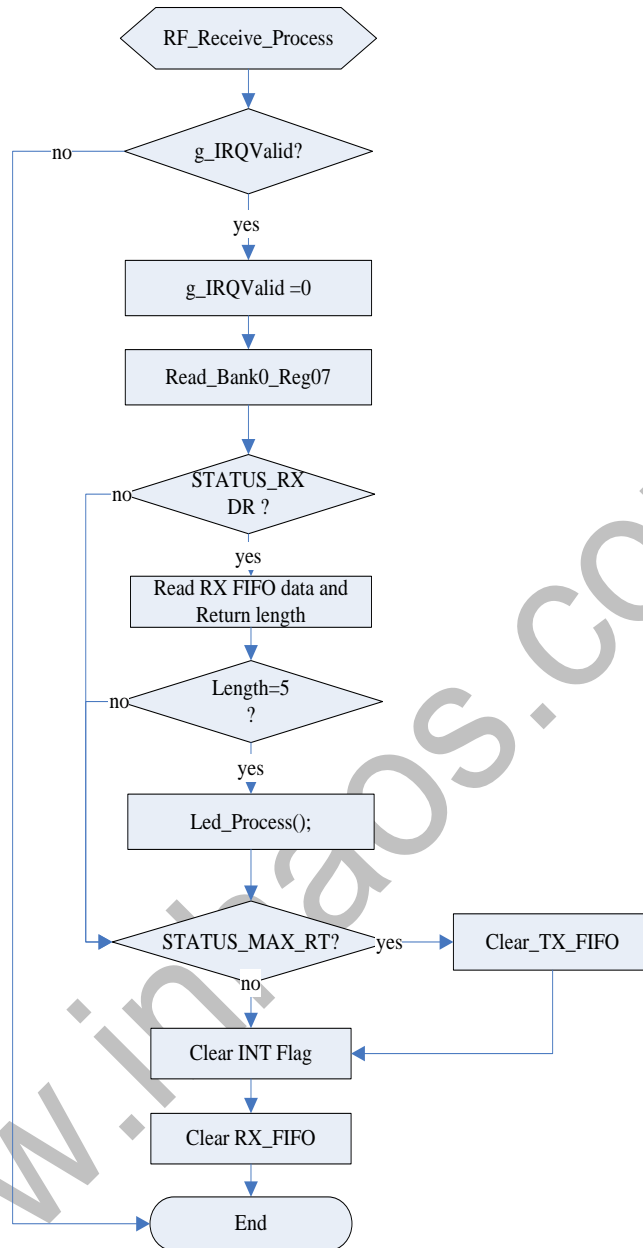
- 1、Receive mode configuration:
 - a) Set PRIM_RX to 1 in Bank0_STATUS register
 - b) Enable data receiving channel(by setting EA_RXADDR register)
 - c) set the data length (set by the RX_ADDR_Pn Register)
 - d) set RX address to the corresponding channel (through RX_ADDR_Pn register)
 - e) set automatic answer mode (by EN_AA register)
- 2、Raised the CE pin to start data receiving.
- 3、When receiving valid data (address match, CRC checksum correctly), BK2421 stored data in RX FIFO, RX_DR bit is high, and low IRQ pin.
- 4、When enable the automatic answer feature,BK2421 hardware will automatically switch to transmitting mode, and launch ACK response packet (Note: the transmitting address is the same to the receiving).
- 5、Standby mode after CE is low.

Data receiving process of the sample code.

Scan MCU to detect whether the IRQ pin is low. If low, then the interrupt occurs. Read the RX_DR, TX_DS, MAX_RT bit status of Bank0 the Reg0 (STATUS register):

- A、If RX_DR=1,receiving interrupt happens. Read the received data and its length, if the received data is equal to 5 bytes, receiving succeed. Press SWn(n=1,2,3,4,5) to light corresponding LEDn (n = 1,2,3,4,5),then clear the RX FIFO register and the interrupt flag. If the received data is not equal to 5 bytes, then clear the interrupt flag bit and RX FIFO registers.。
- B、If MAX_RT=1 means the max retransmitting. It requires manual addition to TX FIFO registers and the interrupt flag.

RF data receiving flow chart:



(Diagram 12)

Sample code is as follows

```

#define STATUS_RX_DR    0x40
/*****
*Funcname:      void RFReceiveProcess( void )
*Remarks:      If receive flag is set, then read out the received data from RF RX fifo, then
process the read out data
*****/
void RFReceiveProcess( void )
{
    UINT8 sta;
    UINT8 rlen;

```



```
if( g_RFIRQValid )
{
    g_RFIRQValid = FALSE;
    sta = RF_GET_STATUS();           //Get the RF status
    if( sta & STATUS_RX_DR )         //Receive OK?
    {
        //Readout the received data from RX FIFO
        rlen = RF_ReadRxPayload( (UINT8 *)&g_RFRecvBuff, RFPKT_LEN );
        if ( rlen == RFPKT_LEN )
        {
            switch( g_RFRecvBuff[0] )
            {
                case SW_1:
                    GLED_FLASH();
                    break;
                case SW_2:
                    GLED_FLASH();
                    break;
                case SW_3:
                    GLED_FLASH();
                    break;
                case SW_4:
                    GLED_FLASH();
                    break;
                case SW_5:
                    GLED_FLASH();
                    break;
            }
        }
    }
    if( sta & STATUS_MAX_RT )         //Send fail?
    {
        RF_FLUSH_TX();               //Flush the TX FIFO
    }
    RF_CLR_IRQ( sta );               //Clear the IRQ flag
}
```

3.5.2 RF data launch mechanisms:

1. Configuration Bank0_STATUS register PRIM_RX low, into the launch mode.
2. Before transmitting data, MCU would write the address to the TX_ADDR register, and the data to the written TX FIFO register. Note that the

receiving address is the same to the receiver.

3. By raising CE, to start BK2421 to send the data in the TX FIFO. CE continued high for at least 10us.
4. After transmitting BK2421 data in Auto Answer mode (auto retransmitting count is not 0), it will immediately enter the RX mode and wait to receive ACK packets. A valid ACK packet received within the time frame means data is received successfully by receiving party. At this point, TX_DS in Bank0_STATUS register will be set to 1, while data is removed from the TX FIFO registers. If still no ACK packet in max retransmitting, BK2421 will automatically set Bank0_STATUS register MAX_RT bit to 1, and transmitting failed. The outgoing data won't be removed until software clear it.
5. CE low would enter the Standby-I mode, otherwise the system will send the next packet data in TX FIFO registers. If the register is empty and the CE is high, then enter the Standby-II model.
6. Set CE low will change Standby-II mode to Standby-I mode.

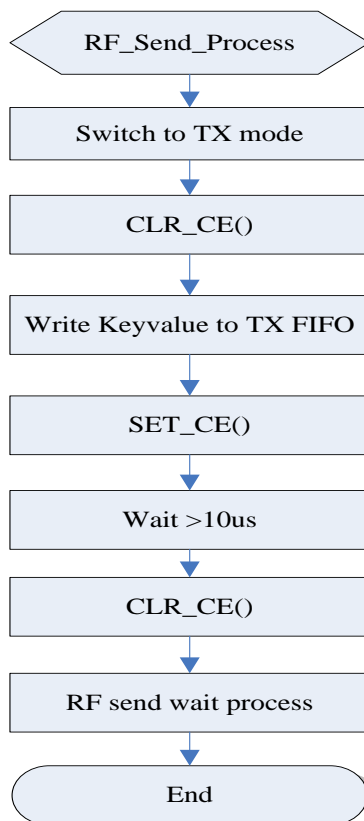
The data transmitting process of the sample code:

RF transmitting module is started by button-press. First detect the button, switching to TX mode if it's pressed. Set CE (RX/TX Prohibition mode) low, write the key to TX FIFO register, and set CE (Enable RX/TX mode) high and last for at least 10us. Then start transmitting timeout counting, constantly test whether the IRQ pin is pulled low and whether the timeout until launch complete. If IRQ low, interrupt, then read TX_DR, MAX_RT bit status of Reg07 (STATUS register) in Bank0:

- 1、 If MAX_RT=1, means max retransmitting, clear the TX FIFO registers and the interrupt flag bit, the yellow light bright (means transmitting failed). End the process.
- 2、 If TX_DS=1, means transmitting succeed, the yellow light is off, clear the interrupt flag. End the transmitting process.

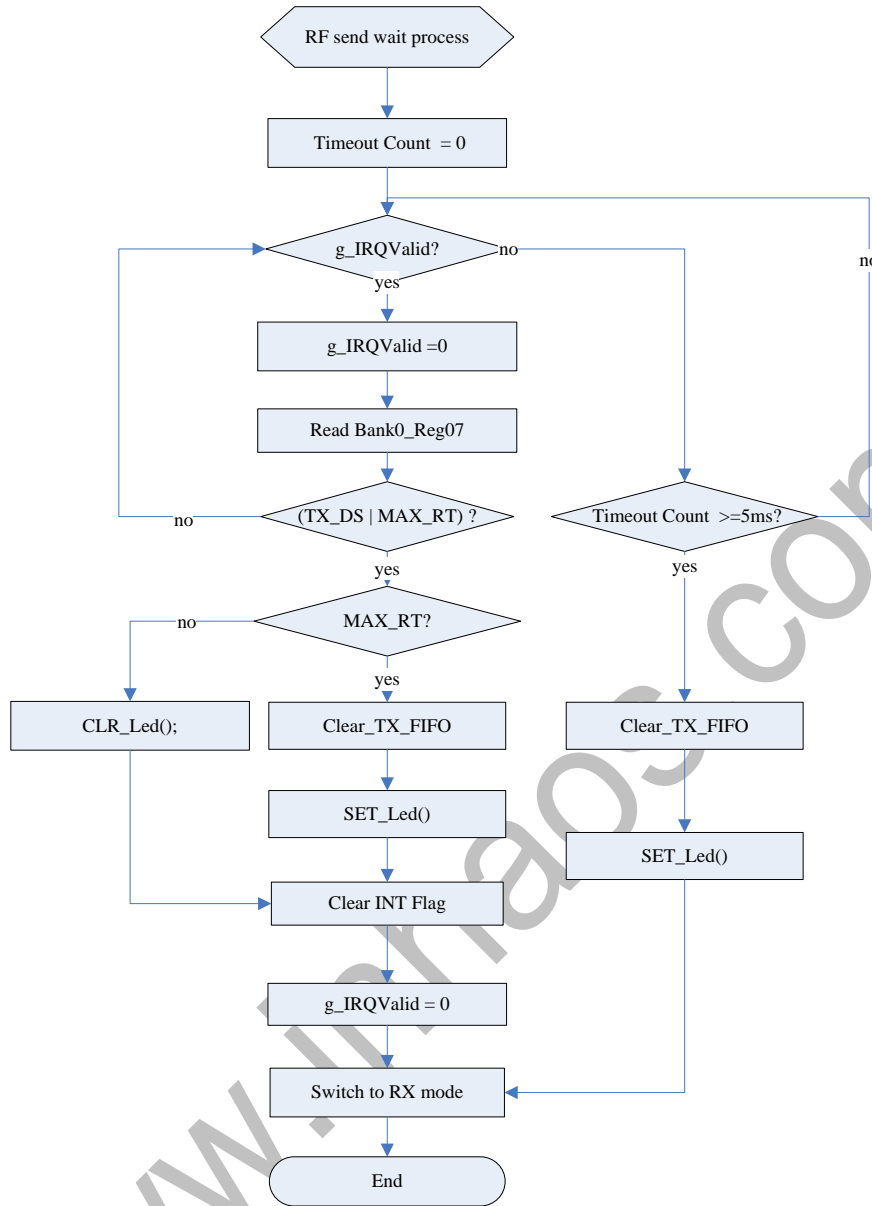
Transmitting over 5ms means over time and discard packet, clear TX FIFO registers, the yellow light bright. End the transmitting process. Neither time-out and nor interrupt, wait until transmitting complete.

RF data transmitting flow chart:



(Diagram 13)

RF ready-to-launch flow chart:



(Diagram 14)

Sample code is as follows:

```

#define STATUS_RX_DR      0x40
#define STATUS_TX_DS      0x20
#define STATUS_MAX_RT     0x10
/*****
*Funcname:      void RFSendPacket( UINT8 Key )
*Remarks:      Switch to TX Mode, Writes Data to TX FIFO ,After That, Into Interrupt
Detect Send Result, Fail or Timeout, Success. Switch to RX Mode
*****/
void RFSendPacket( UINT8 Key )
{
    UINT8 sta;
    UINT16 temp = 100;
    g_RFSendBuff[0] = Key;

```

```
SwitchToTxMode();           //Set RF to TX mode
CLR_CE();
SPI_Write_Buf(WR_TX_PLOAD, (UINT8 *)&g_RFSendBuff, RFPKT_LEN);
g_RFIRQValid = FALSE;
SET_CE();
while( temp-- );           //Wait for Time > 10us
CLR_CE();
//Wait for send over
g_DelayTick = 0;
while( 1 )
{
    if( g_RFIRQValid )
    {
        sta = SPI_Read_Reg( STATUS );           // read register STATUS's value
        if( (sta & STATUS_TX_DS) || (sta & STATUS_MAX_RT) )
        {
            if( sta & STATUS_MAX_RT )           //if send fail
            {
                RF_FLUSH_TX();
            }
            RF_CLR_IRQ( sta );
            RLED_OFF();
            g_RFIRQValid = FALSE;
            break;
        }
    }
    else if( g_DelayTick >= 5 )           //if timeout
    {
        RF_FLUSH_TX();
        break;
    }
}
SwitchToRxMode();
}
```

3.5.3 Power measurement Module

RF It usually required carrier power detection in RF test, that means BK2421 needs output single carrier signal in the specified channel. The implementation process is as follows:

- Set the chip in launch mode: Write Bank0_REG [0] _BIT0 = 0, Bank0_REG0_BIT1 = 1; pull CE high.
- Set the channel and the frequency: Write Bank0_REG [05] = 0X20; the corresponding frequency $F = (2400 + 32)$ MHZ.

INHAOS RF-2400 Series Module Application Guide -AVR

- Set the chip in a single carrier launch mode: Bank1_REG[04]=0XD99E8621;
- Set the chip in normal launch mode: Bank1_REG[04]= 0x0B869ED9;

Sample code is as follows:

```
/******  
*Fancname: void RF_SetCarrierOut( UINT8 bEnable )  
*Remarks: Set RF for serial single carrier mode  
*****/  
void RF_SetCarrierOut( UINT8 bEnable )  
{  
    UINT8 j = 0;  
    UINT8 Temp = 0;  
    UINT8 WriteArr[4];  
    SwitchToTXMode();  
    CLR_CE();  
    SPI_Write_Reg(W_REGISTER | RF_CH, 0X20 );  
    //Convert to Bank1 Modle=====   
    Temp = SPI_Read_Reg( STATUS );  
    Temp = Temp & 0X80;  
    if( !Temp )  
    {  
  
        SPI_Write_Reg( ACTIVATE, 0x53 );  
    }  
    if( bEnable ) //Enable single carrier mode;  
    {  
        for( j = 0; j < 4; j++ ) //Configuration single carrier  
        {  
            WriteArr[j] = ( Bank1_Reg04 >> ( 8*(j) ) ) & 0xff;  
        }  
    }  
    else //Enable normal modle  
    {  
        for( j = 0; j < 4; j++ ) // Configuration normal modle  
        {  
            WriteArr[j] = ( Bank1_Reg0_Reg13[4] >> ( 8*(j) ) ) & 0xff;  
        }  
    }  
  
    SPI_Write_Buff( (W_REGISTER | 4), &(WriteArr[0]), 4 );  
    //Convert to Bank0 Modle=====   
    Temp = SPI_Read_Reg( STATUS );  
    Temp = Temp & 0X80; //Bank0_Register[07] = 1:Bank1:Otherwise Bank0;  
    if( Temp )  
    {  
        SPI_Write_Reg( ACTIVATE, 0x53 );  
    }  
    SET_CE();  
}
```

3.6 References

1. BK2421/BK2401 Datasheetv2.0
2. BK2401_BK2421Hardware Reference Designv3.0
3. nRF24L01_Product_Specification_v2_0

www.inhaos.com

Declare

Due to technical limitations and the reader's understanding , this document is for reference only. Our company makes no legal commitment or guarantee of the document. If you have any doubt, please feel free to contact our company or authorized service provider, thank you! (The source code of the example can be download form www.inhaos.com.See the website for more technical support

Copyright

All the devices mentioned in this document are all cited from the information of the company copyright reserved. The rights to modify and distribute belong to the company, we do not make any guarantees of the information. When in application, please confirm the information updated through the appropriate channels ,and adjust accordingly.