1. Introduction

RF-2410M module is a wireless communication module which integrate C8051F330 and BK2421 chip produced by INHAOS. Modular package can be used as target board or welded to stick in PCB board. Users can control BK2421 chip for wireless data transfer by programming software on C8051F330. As for this module, our firm provides detailed datasheet, application notes, after-sales service, etc. Please visit our website (www.inhaos.com) to get more information.

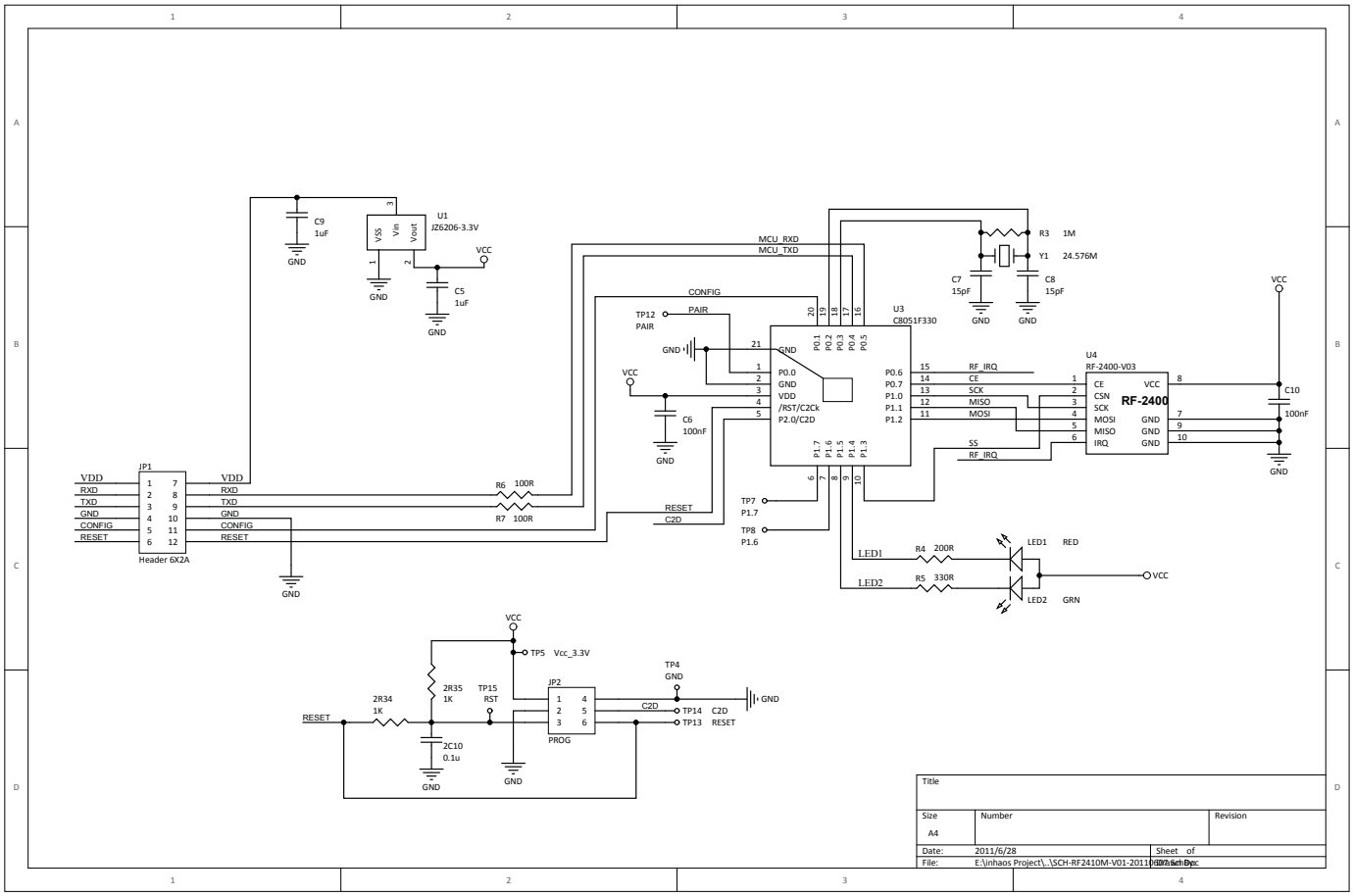
This document aims to illustrate the application of RF-2410M module by hardware platform and codes. This example only provides one-way transceiver operation, can transiceive multi-byte data at one time. The distance can reach about 10m for one-way. Communication distance depends on circumstance. Users can improve performance by changing baud rate, RF communication channels and output power, etc. You can get some revelation and reference to develop RF-2410M module.

2. Hardware architecture

This hardware platform is assembled by C8051F330 MCU, RF-2401-V03 module and its peripherals. The hardware platform is made into two pieces of small size, portability module which can be used as SMD or DIP. Research appears very convenient no matter design product or teaching.
Attention: RF-2410M module link to PC by USB to RS232 interface converter (here using INHAOS UC-2000). Besides, RF-2410M requires a 3.3V to 7V power supply separately.

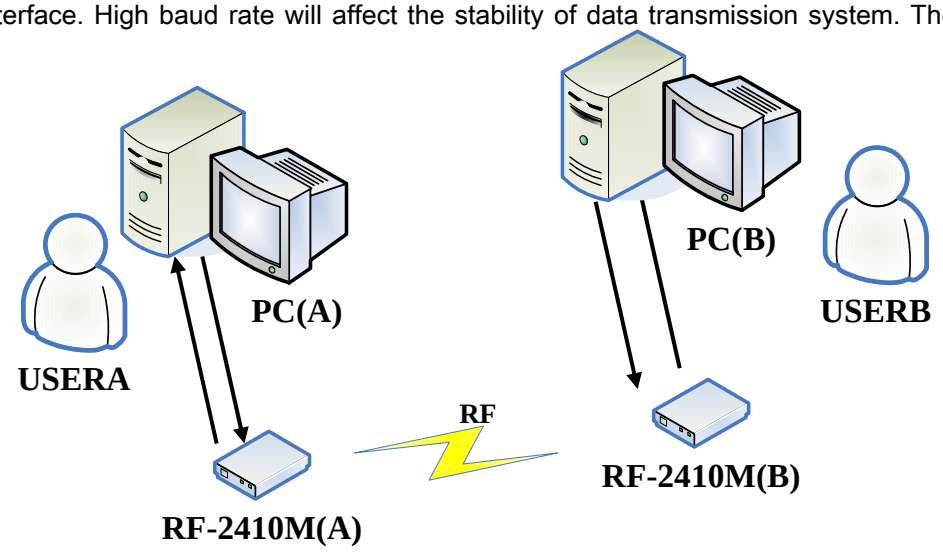
2.1. RF-2410M module physical picture

2.2. RF-2410M module Schematic



3. Software Architecture

The purpose of this system software is building a data transmission channel between PC (A) and PC (B). the sample code only provide half-duplex data transmission. You can send data from PC(A) to PC(B) or vice versa. RF-2410M port can't connected with the PC directly only after converting with UC-2000 (USB to RS232) . After transferring, RF-2410M transmits data with PC by UART interface. You need to consider how to set the baud rate as a result of adopting UART interface. High baud rate will affect the stability of data transmission system. The baud rate here is 57600bps.

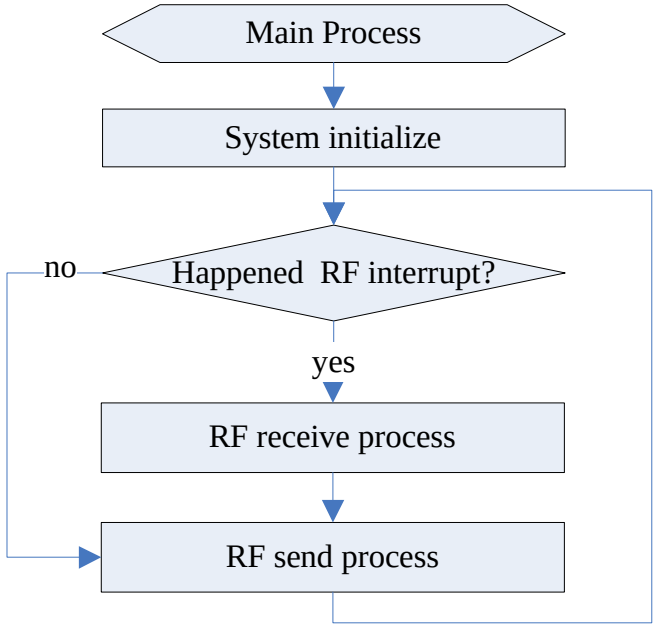


Software functions (Assume module A and B communicating)

- PC (A) sends multi-byte data to RF-2410M A by UART. UART will copy the data to RF transmitter buffer, start RF and send the data to RF-2410M B module. Flash the green light when successfully sending, flash red light indicate retransmission failure.
- RF-2410M B module verify the data it received. If correct, RX_DR will be interrupted. And then copy the data to UART buffer. After starting UART, the data reach PC (B). Flash the green light when successfully sending each time.

3.1. Overall Structure

Including: initialization part, RF receiving processing part and RF transmit processing part, etc.
Software flow chart as follows:



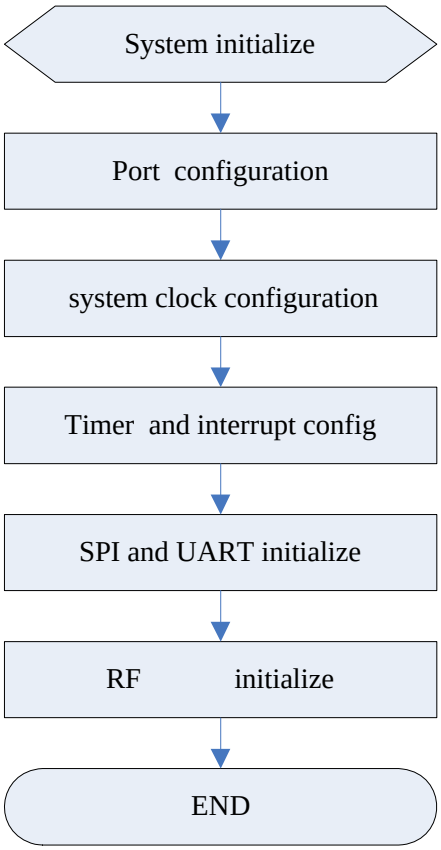
(Diagram 04)

3.2. Initialization part

Including:

- Port configuration
- Clock Configuration
- Timer initialization
- External interrupted configuration
- SPI and UART configuration
- RF Initialization

3.2.1 System Initialization Chart



(Diagram 05)

3.2.2 Timer、SPI and UART Configuration

1、Timer configuration: Timer:0; work mode by 1, time period is 1ms. External clock divided by 8 as clock source

```
//sample code as follow:
void Timer_Init( )
{
    TMOD        |= ( 1 << 0 );              //Timer0 workways 1;
    CKCON       = 0x03;                    //Colck External /8;
    TH0         = 0xF4;
    TL0                     = 0X00;
    TCON                    |= ( 1 << 4 );          //TR0 = 1;
}
```

2、SPI configuration: Working in 3-wire SPI, Master-Slave mode, and data rate is 6MHz,

```
//sample code as follow:
void SPI_Init()
{
    SPI0CFG     = 0x40;
    SPI0CN      = 0x03;                    // Three wire master mode ;
    SPI0CKR     = 0x01;                    // Clock rate = 6MHz;
}
```

3、UART configuration: Work with 8bit mode, timer1 as baud rate generator. Work mode by 2, external clock divided by 8 as clock source

```
//sample code as follow:
void UART_Init()
```

```
{
    SCON0   = 0x30;                         //8 bit UARAT;
}

void Change_BaudRate( UINT32 pBaud )
{
    UINT16    b   = 0;
    INT32     a   = 1536000L;
    while( a   >=   pBaud )
    {
        a   =   a -   pBaud;
        b++;
    }
    TH1       = 256 - b;
    TL1       = TH1;
    TMOD    |= ( 1 << 5 );                  //Timer1 workways 2;
    CKCON   = 0x03;                         //Colck External /8;
    TCON    |= ( 1 << 6 );                          //TR1 = 1;
}
```

3.3. RF Initialization Operation Part

RF-2410M module's data rate is configured to 2Mbps, transmit power is 5dBm and enable auto-ACK. Set receive / transmit address width 5 bytes. Start automatic retransmission function (timeout 250us, Retransmission time is five), parity bytes is 2 and initial configuration for RX mode.

Notes:

There is no order when config Bank0 and Bank1. But you must switch to the state Bank0 after the configuration.

Switch between Bank0 and Bank1: First read Bank0_Reg07_Rbank bit's status. If Rbank bit is 0, means the current state is in Bank0, otherwise is in Bank1. You can switch Bank0 and Bank1 by SPI writing "ACTIVATE +0 x53" command. You have to active register by ACTIVATE command before write DYNPD and FEATURE register, otherwise the operation is invalid.

3.3.1 . BANK0Register Configuration

Bank0 Register Configuration Description: Read low byte first, and then high byte. Read and write each byte from high to low.

Bank0 Register Configuration

Bank0 Mnemonic	Configuration values	Remarks
CONFIG	0x0F	RX mode, PWR_UP mode, 2 bytes CRC
EN_AA	0x01	Enable auto acknowledgement data pipe 0
EN_RXADDR	0x01	Enable data pipe 0.
SETUP_AW	0x03	5 bytes RX/TX address
SETUP_RETR	0x05	Auto Retransmission Delay 250uS and total 5times
RF_CH	0x20	Select RF channel 32 (2432MHz)
RF_SETUP	0x2d	Data rate is 2Mbps, Transmit power is 0dbm,

		high gain
STATUS	0x70	Clear interrupt bit
OBSERVE_TX	0x00	Defaults
CD	0x00	Defaults
RX_ADD_P0	0x2a, 0x2b, 0x2c, 0x2d, 0x02	Address of pipe0
RX_ADD_P2	0xC3	Defaults
RX_ADD_P3	0xC4	Defaults
RX_ADD_P4	0xC5	Defaults
RX_ADD_P5	0XC6	Defaults
TX_ADDR	0x2a, 0x2b, 0x2c, 0x2d, 0x02	Transmitter address
RX_PW_P0	0x20	Max 32bytes for pipe0
RX_PW_P1	0x20	Max 32bytes for pipe1
RX_PW_P2	0x20	Max 32bytes for pipe2
RX_PW_P3	0x20	Max 32bytes for pipe3
RX_PW_P4	0x20	Max 32bytes for pipe4
RX_PW_P5	0x20	Max 32bytes for pipe5
FIFO_STATUS	0x11	RX/TX FIFO Blank
DYNPD	0x01	Enable dynamic payload length data pipe 0
FEATURE	0x04	Enables Dynamic Payload Length

(Diagram 06)

```
//Sample codes as follow :
//Bank0 Registers Config==================================================
code volatile UINT8 Bank0_Reg_Init[21][2] = {
{CONFIG,        0x0F},   {EN_AA,        0x01},   {EN_RXADDR,    0x01},
{SETUP_AW,      0x03},   {SETUP_RETR,   0x05},   {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] = { 0x2a, 0x2b, 0x2c, 0x2d, 0x02 };
const UINT8 TX_Address[5] = { 0x3a, 0x3b, 0x3c, 0x3d, 0x01 };

const UINT8 Bank0_RegAct[2][2] = {    {DYNPD,    0x01},   {FEATURE, 0x04}   };

void Bank0_Register_Init( void )
{
    UINT8   i   = 0;
    UINT8   k   = 0;
    UINT8   Rt = 0;
//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(R_REGISTER |   Bank0_Reg_Init[i][0] );
    }
//Write RX /TX Address=====================================================
    RF_SET_RX_ADDR( &URX_Address[0] );
    RF_SET_TX_ADDR( &URX_Address[0] );
//Active Ship Operate configurationDYNPD and FEATURE register ========================
    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( R_REGISTER | Bank0_RegAct[i][0] );
    }
}
```

3.3.2 . BANK1 Register Configuration

Bank1 Reg00 to Reg08's reading and writing order: high to low. Each byte is also from high to low;

Bank1 Reg09 to Reg0E's reading and writing order: low to high. Each byte is from high to low;

Bank1 Register Configuration

Bank0 Register No.	Configuration values
00	0xE2014B40
01	0x00004BC0
02	0x028CFCD0
03	0x41390099
04	0x0B869ED9
05	0xA67F0624
06	0x00000000
07	0x00000000
08	0x00000000
09	0x00000000
0A	0x00000000
0B	0x00000000
0C	0x00127300
0D	0x36B480000
0E	0x412008048120CFF7FEFFFF

(Diagram 07)

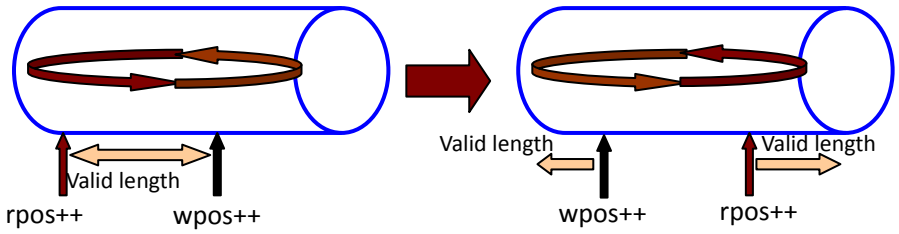
```
//Sample codes as follow:
code volatile   UINT32   Bank1_Reg0_Reg13[] ={
0xE2014B40,   0x00004BC0, 0x028CFCD0,0x41390099,   0x0B869ED9,   0xA67F0624,
0x00000000,    0x00000000,   0x00000000, 0x00000000,   0x00000000,   0x00000000,
0x00127300,    0x36B48000};
code volatile   UINT8    Bank1_Reg14[11] = { 0X41, 0X20, 0X08, 0X04, 0X81, 0X20, 0XCF, 0XF7,0XFE, 0XFF,
0XFF};

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. UART interrupt processing module

UART interrupt handling process description: By setting up two large buffers to store data. They are g_RFRcBuff[200] and g_UARTSnBuff[200]. This two buffer adopt shift register work mode. When the buffer accumulated to a maximum, it will return to the starting position. Data which has been written before will be covered.

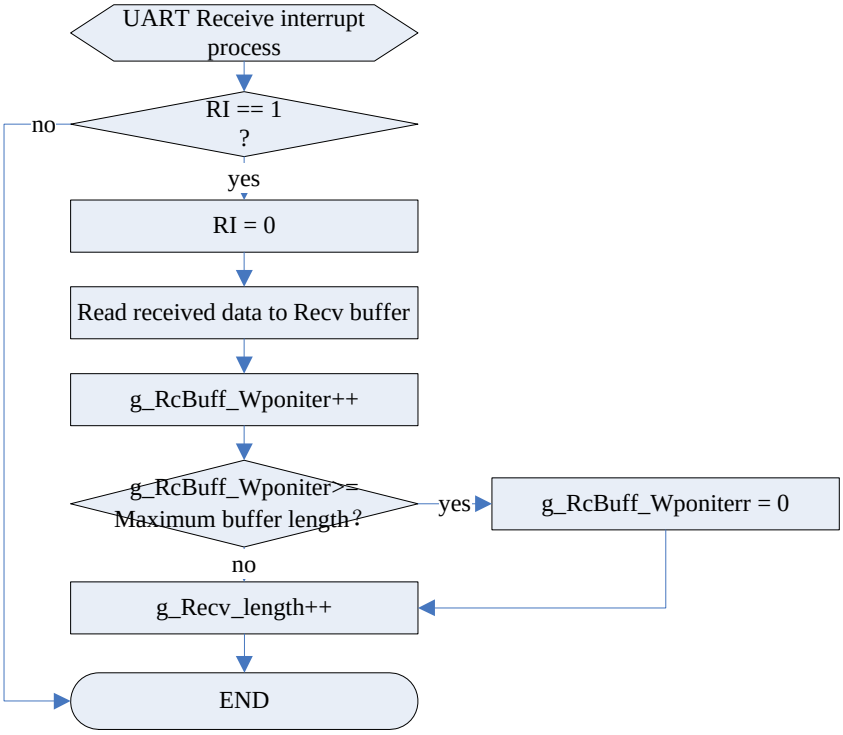
3.4.1 . UART receive interrupt process



(Diagram 08)

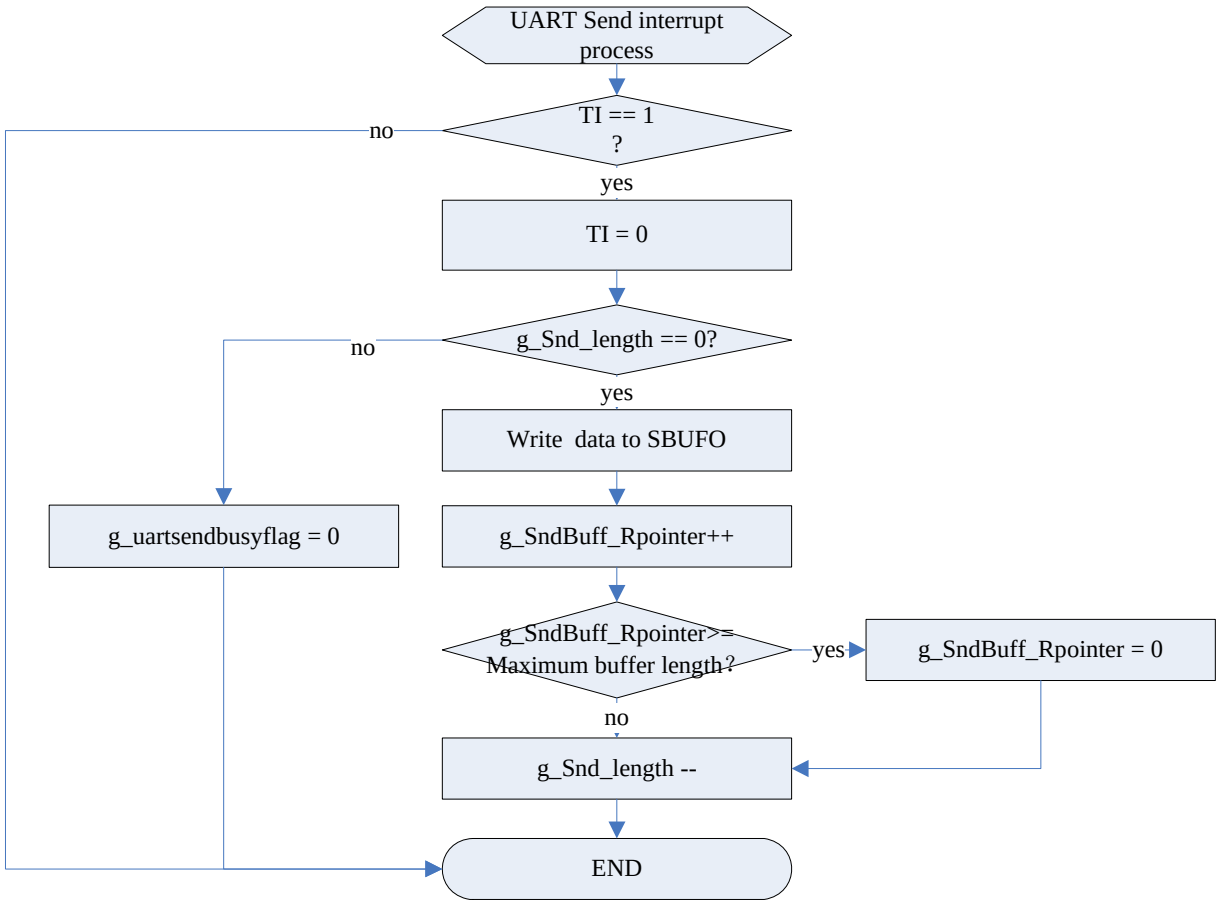
UART receive interrupt process include four parts: rpos (Read pointer) , wpos (Write pointer) , RcLen (Valid length of receiving buffer) , g_RFRcBuff[200](receiving buffer) . Every time serial ports received , the wpos and RcLen will increase by the receive lenght, if wpos reach to the max value it will point to the starting position again. Every time Read one byte, rpos++ and RcLen-- each time, if rpos reach to the max value it also will point to the starting position again. If RcLen's value is greater or equal than the maximum length of buffer, RcLen will remain the same. rpos = wpos will read the newest value.

Software flow chart



3.4.2 . UART send interrupt process

UART Transmit process is similar with receiving process. It consists rpos,wpos,SnLen and g_UARTSnBuff[200]. When RF receives valid data, it will copy the data to g_UARTSnBuff. If UART detected do not in sending busy position, it will force to start the transmission process. When UART detect the valid length of data (SnLen) is not 0, it starts to send data. Otherwise it will have to wait for the next packet data



```
//Sample codes as follow:
void UART_ISR( void )interrupt 4
{
    if( RI0 )
    {
        RI0   = 0;
        g_RFRcBuff[g_RF.wpos++] = SBUF0;                    //read SBUFO of data
        if( g_RF.wpos >= RFRCLEN )                  //g_RF.wpos is wirite pointer
        {
            g_RF.wpos = 0;
        }
        g_RF.RcLen++;                                   //UART received /valid data length
    }
    if( TI0 )
    {
        TI0 = 0;
        if( g_uart.SnLen )
        {
            SBUF0 = g_UARTSnBuff[g_uart.rpos++];
            if( g_uart.rpos >= UARTSNLEN )
            {
                g_uart.rpos = 0;
            }
            g_uart.SnLen--;                             //UART send /valid data length
        }
        else
        {
            g_uart.SendInProgress = FALSE;
        }
    }
}
```

3.5. RF2410M module communication

PC(A) send bytes to the module A by serial port. Module A copy the data from UART to RF packet and then start sending process. Module B have to wait for data from A, and then copy the data to UART sending buffer. Finally, send the data to PC(B) by serial port. RF data transmission adopt separate packet transmission mode. The packet format is defined as follows:

	SN	Len	SN_PKT	Parameter
Definition	RF serial number	Valid data length	RF Packet Serial Number	Data
Length	1 Byte	1 Byte	1 Byte	0~25 Bytes
Value	0x00~0xff	0x00~0x19	0x00~0xff	0x00~0xff

(Diagram 11)

The maximum packet length of this sample is 28Bytes (Must be less than RF TX / RX FIFO Length). You can regulate the length of parameter from 0 to 25. So the length of the whole packet is dynamic packet length. The range is 3 to 28 bytes.

3.5.1 . RF Data receiving process

RF data receiving mechanism

- Receive mode configuration

- Set Bank0_CONFIG register PRIM_RX by 1
- Set Bank0_CONFIG register PWR_RX by 1
- Enabled data pipe(by EA_RXADDR register)
- Set data length(By RX_ADDR_Pn register)
- Set the RX pipe address(By RX_ADDR_Pn register)
- Set automatic ACK mode (Set by EN_AA register)

➢ Set CE pin to start receiving data

➢ When receiving valid data (Address match, CRC correct), save the data to RX FIFO. Set STATUS_RX_DR bit and Reset the IRQ pin.

➢ the hardware will automatically switch to transmit mode and sends acknowledge (ACK) packet if using automatic ACK

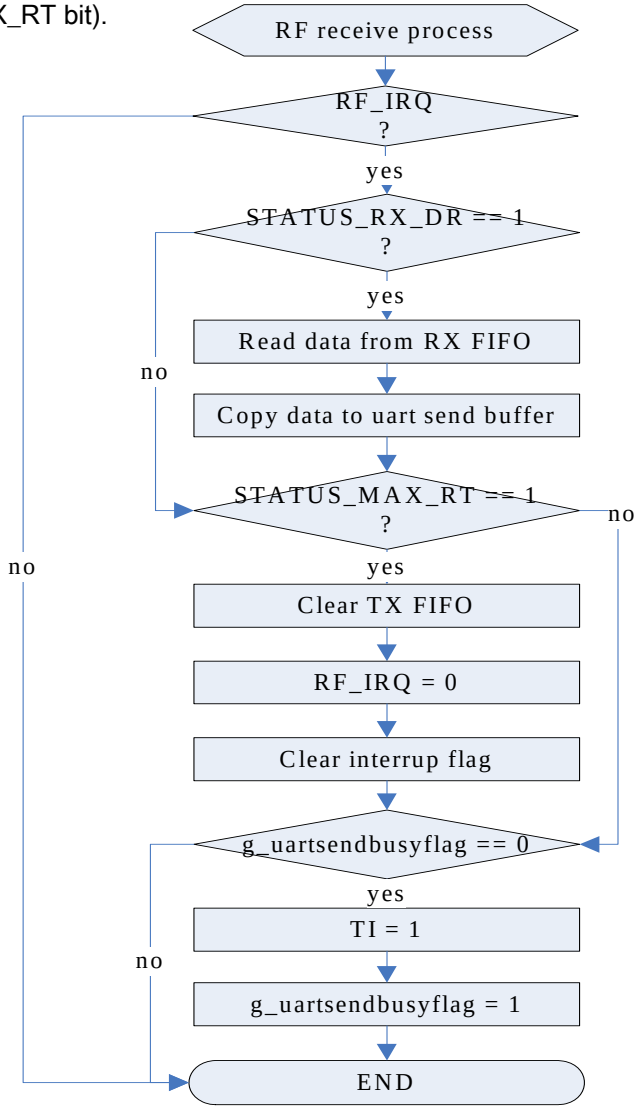
➢ Entering Standby mode after Reset CE pin.

Data receiving process

MCU is responsible for detecting whether the IRQ pin is low. If is low, means have been interrupted. Read Bank0 Reg0's RX_DR,MAX_RT state:

1、If RX_DR is 1, indicates that receiving have been interrupted. Read data from RX FIFO register and flash the green light, clear the interrupt flag(RX_DR bit). Then start to copy data to UART sending buffer. If the module is not in the busy position, force to start UART sending process.

2、If MAX_RT is 1, indicates the maximum number of retransmission. Then you need to manually clear TX FIFO register and interrupt flag(MAX_RT bit).



(Diagram 12)

```
//Sample codes as follow:
void RF_RcPackageProcess( void )
{
    UINT8 Rt = 0;
    UINT8 Len = 0;
    UINT8 temp = 0;
    UINT8 isValid = FALSE;

    if( RF.IRQValid )
    {
        RF.IRQValid = FALSE;
        Rt = SPI_Read_Reg( R_REGISTER | STATUS );                    //read interrupt flag;
        if( Rt & STATUS_RX_DR )                                      //Happen RX receive interrupt?
        {
            Len =   Read_RXPayload(( UINT8 *)&g_RFRecvData ,sizeof( RFDATAPKT ));
            FLUSH_LED2();

            isValid = TRUE;
        }
        if( Rt & STATUS_MAX_RT )                                     //Send Fail?
        {
            SPI_Write_Reg( FLUSH_TX, 0X00 );                         //Clear TX FIFO;
        }
        SPI_Write_Reg( FLUSH_RX, 0X00 );
        SPI_Write_Reg( W_REGISTER | STATUS, Rt );

        if( isValid )
        {
            CopyRF_RcDatatoUART_SnBuff( &g_RFRecvData , Len );   //copy data;

            CLR_EA();
            g_uart.SnLen += RF.R_Usalvelen;                          //new length;
            if( g_uart.SnLen > UARTSNLEN )
            {
                g_uart.SnLen = UARTSNLEN;
                g_uart.rpos   = g_uart.wpos;
            }
            SET_EA();
        }
        if( !g_uart.SendInProgress )
        {
            TI0 = 1;
            g_uart.SendInProgress = TRUE;
        }
    }
}
```

3.5.2 . RF Data transmission process

RF Data transmission mechanism

- deploy Bank0_CONFIG register's PRIM_RX low, enter into transmit mode
- Before sending data, MCU need to write down the address and outgoing data to TX_ADDR register and TX FIFO register.
- Start to send TX FIFO's data by Set CE pin and keep high level at least 10uS.
- If use auto-ACK mode (automatic retransmission is not 0), it will immediately enter into RX mode and wait to receive ACK packet after finish sending data. When you receive ACK packet in valid time means data has been

successfully accepted by counterpart, Bank0_STATUS register's TX_DS will be 1. Meanwhile the data will be cleared from TX FIFO register. If the retransmission reaches the maximum value but still not receive ACK packet, the hardware will automatically set Bank0_STATUS register's MAX_RT bit to 1, indication transmit failed. You need to remove the outgoing data from TX FIFO registers by executive Software.

- If CE set low, then enter the standby-I mode. Otherwise the system will send the next packet data from TX FIFO register. If CE set high and register is empty, then enter the standby-II mode.
- If system enter standby-II mode, then Reset CE will enter standby-I mode.

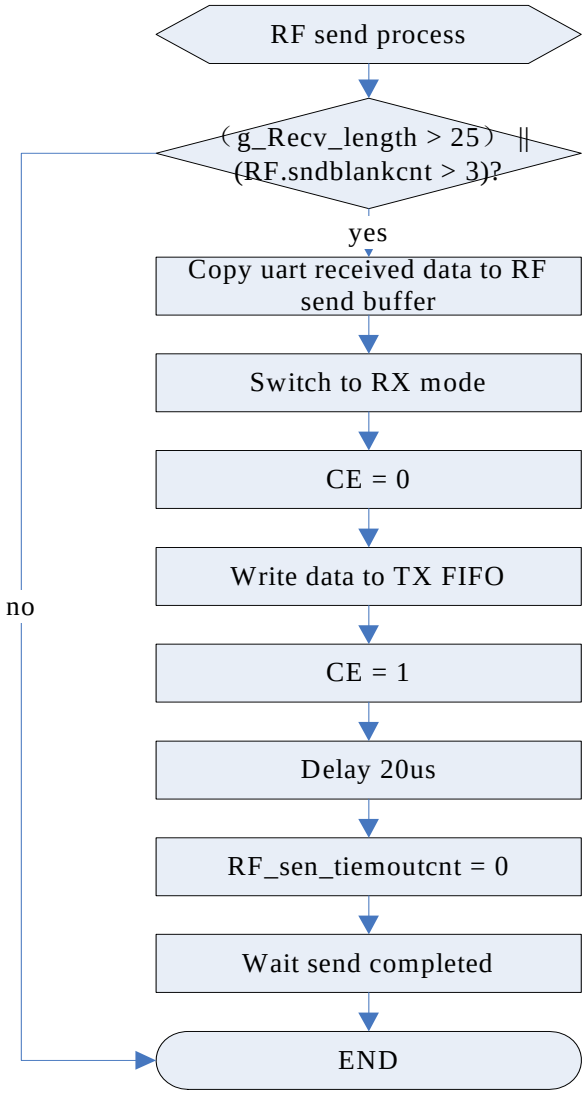
Data transmission process

UART MCU continuously detects the received data packet length whether is greater or equal than PKTLENGTH or whether interval reaches 3ms or not. If meet above conditions, it will start to copy UART's data to RF sending packet. Then write the packet to the TX FIFO register and start counting timeout. Detect IRQ pin whether has been pulled down or timeout. If pull down means interrupting, and then start read Reg07 in BANK0 (STATUS register) TX_DR, MAX_RT bit 's status:

1、If MAX_RT is 1, retransmission reach maximum, then clear TX FIFO register and its interrupt flag. Flash red light once (fail to transmit)

2、If TX_DS is 1, means successfully transmit. Flash green light once and clear interrupt flag. Ending the whole transmit process.

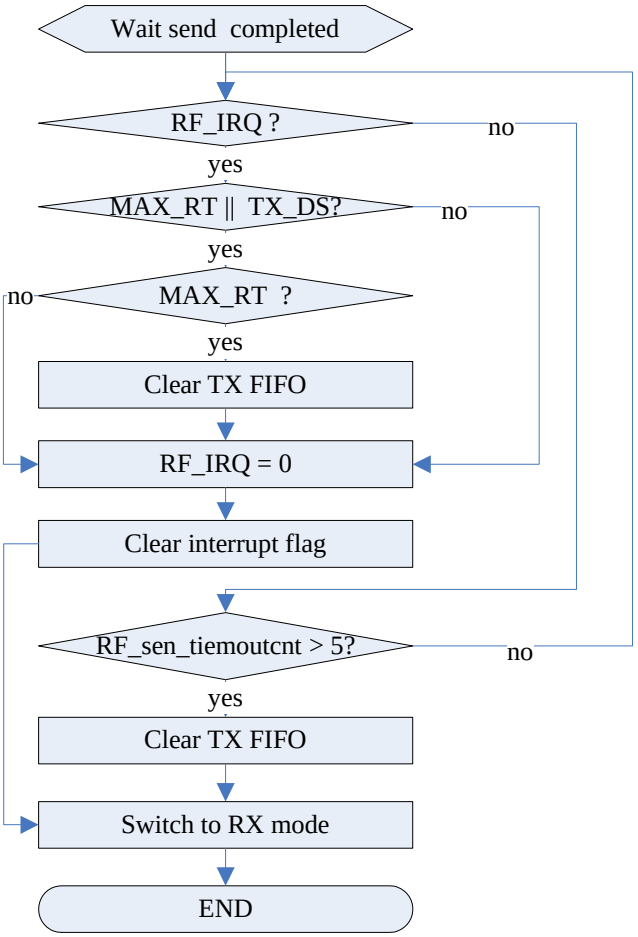
3、If transmit process is more than 5ms, consider timeout, and drop the packets, clear TX FIFO register, flash red light. End of the whole launch process



(Diagram 13)

```
//Sample codes as follow:
void RF_SnPackageProcess( void )
{

    UINT8 Len = 0;
    if( ( (g_RF.RcLen >= RF_PKT_LEN    ) || ( RF.snblankcnt >= 3 ) )    )
    {
        CLR_EA();
        RF.R_Usalvewpos = g_RF.wpos;                                    //save current wpos position
        if( g_RF.RcLen > RFRCLEN )
        {
            g_RF.RcLen = RFRCLEN;
            g_RF.rpos = RF.R_Usalvewpos;
        }
        RF.U_Rsalvelen = g_RF.RcLen;                                    //save current received length
        SET_EA();

        Len = CopyUART_RcDatatoRF_SnBuff( &g_RFRcBuff );                //copy data;

        CLR_EA();
        g_RF.RcLen -= g_RFSenData.Len;
        SET_EA();

        RF_SendData( (UINT8 *)&g_RFSenData , Len );
        RF.snblankcnt = 0;
    }

}
```

Wait send completed chart

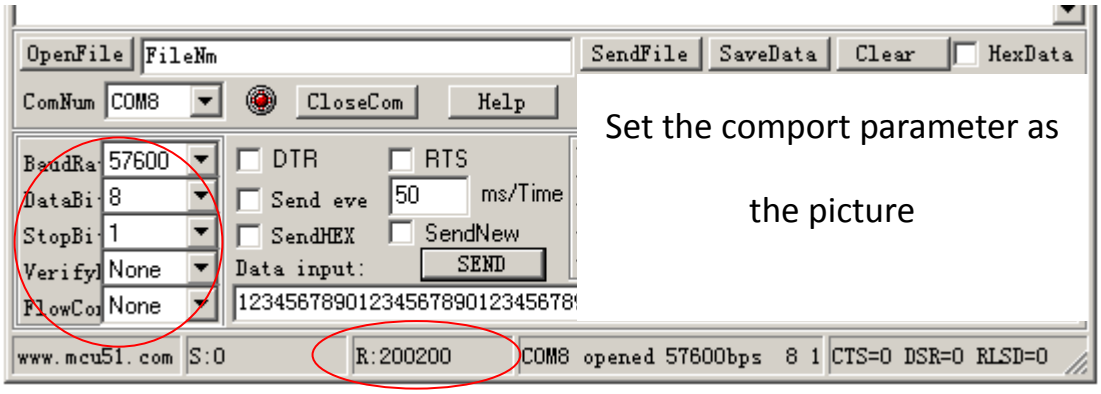


(Diagram14)

```
//Sample codes as follow:
void RF_SendData( UINT8 *pSnd, UINT8 Len )
{
    UINT8 i   = 100;
    UINT8 Rt = 0;
    SwitchtoTXMode();
    CLR_CE();
    RF.IRQValid = FALSE;
    Write_TXPayload( pSnd, Len );          //Writes Data to TX FIFO;
    SET_CE();
    while( i-- );                          //Wait Send;Require Maintain Time > 10us;
    CLR_CE();
    RF.sntimeoutcnt = 0;
    //Wait Send complete
    while( 1 )
    {
        if( RF.IRQValid )
        {

            Rt = SPI_Read_Reg( R_REGISTER | STATUS );

            if( (Rt & STATUS_TX_DS) || (Rt & STATUS_MAX_RT) )
            {
                if( Rt & STATUS_MAX_RT )
                {
                    SPI_Write_Reg( FLUSH_TX, 0X00 );          //Clear TX FIFO;
                    FLUSH_LED1();
                }
                else
                {
                    FLUSH_LED2();
                }
                SPI_Write_Reg(W_REGISTER | STATUS,Rt);
                RF.IRQValid = FALSE;
                break;
            }
        }
        else if( RF.sntimeoutcnt >= 5 )                  //Send Timeout >= 5ms;
        {
            SPI_Write_Reg( FLUSH_TX, 0X00 );            //Clear TX FIFO;
            CLR_LED1();
            break;
        }
    }
    SwitchtoRXMode( );
}
```

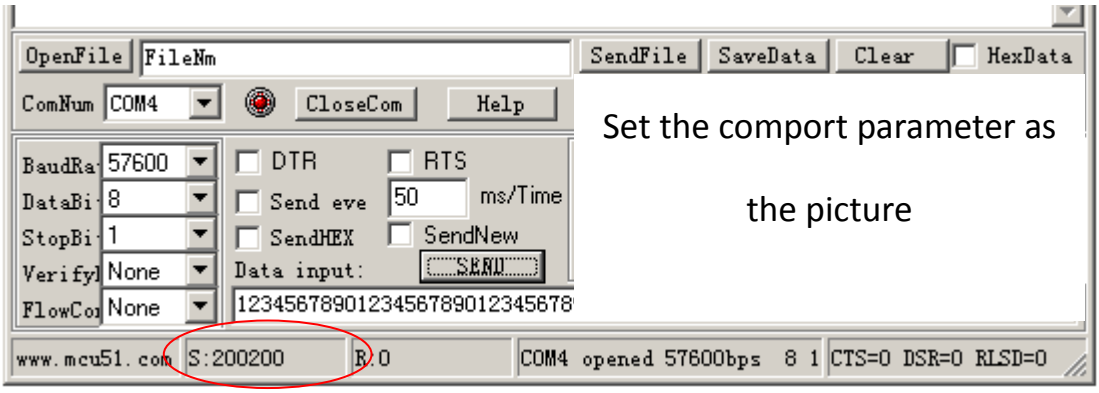
3.6. RF2410M Module performance test

This sample adopt half-duplex communication. RF Configuration: Transmission rate: 2Mbps, Output Power: 5dBm, Baud rate: 57600bps, Transmission distance is 10m, send 200 bytes every 50ms by serial port.



(diagram 15)

PC(A) send data to RF-2410M module A by UART. As diagram 16 shows, send 202000 data packets in total.



(diagram 16)

Module B received data from module A. Display in PC(B) by serial port. Shown in the above chart, receive 200200 data packets in total. The same as receiving data packets.

If the numbers of data packets sent and received are not equal, indicate some data is missing. The following method can be used to debug:

A、Begin debugging, select close distance, about 2m will be better. Set the RF output power of 0dBm, transmission rate of 1Mbps or 2Mbps. When RF transmission rate is high, its sensitivity will be lower. (Special Note: 2Mbps high bandwidth rates will shorten the transmission distance between modules) But if RF transmission rate is too low, serial ports baud rate will be inclined. Otherwise it will lead to UART buffer overflow. There are also another ways to debug. Firstly, you need to debug one-way. Disconnect module A (B) 's TX pin and module B(A)'s RX pin. Secondly, connect module A and B's ground. Finally, connect module B (A) 'sTX pin to the module A (B)'s TX pin. This data can be sent by a serial debug window.

B、Followed debug a little far distance and select environment distance for about 6m. You can test data transmission effect by changing RF communication channels、baud rate、hardware retransmission and RF module orientation etc. During debugging, it would be better to take notes for different parameters' sending packets, receiving packets, losing packets, module orientation towards and environmental conditions. Do a few more tests. So it's easy to find where the problem is.

C、Finally, debug far distance and select environment distance for about 10m. If communication effect is not stable, then increase the output power to 5dBm. Reduce baud rate moderately. Debugging method as above.

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

About Us

INHAOS is a high-tech private limited company combined with electronic products, telecommunications equipment, computer peripheral equipment development and sales. Aiming to promote domestic IT technological progress, we develop a series of embedded product development kit. This kit comes from large quantities of commercial product. The user can use it directly for design and verification, also can quickly convert the design to production and collect new product design ideas .

We also can undertake the following services:

Electronic product design

Brand components acting

Embedded development kit，Circuit module

Contact Us: http://www.inhaos.com/about.php?aID=7