RF2410U USB RF Programming

Introduction

RF-2410U module is an application module which integrates C8051F321 MCU chip and 2.4G wireless chip BK2421 launched by INHAOS. C8051F321 is a high-speed MCU which aims to USB applications compatible 8051 architecture. Its processing speed can reach 25MIPS and has a wealth of external resources; BK2421 is a high data transmission chip designed by BEKEN Company for the purpose of 2.4GRF application. The maximum transfer rate can reaches 2Mbps. Using RF-2410U module can achieve USB communication and RF data transmission at the same time.

RF-2410M integrates C8051F330 and BK2421chip. Modular package can be used as target board or welded to stick in PCB board. Users can control BK2421 chip for wireless data transmission by programming software on C8051F330.

This sample demo achieved by directly using RF-2410U and RF-2410M as hardware platform combine with codes. It shows how to use RF-2410U and RF-2410M source achieve data transmission from one PC to another PC. It covers USB device application development, RF data transmission, SPI protocol, UART data transmission and PC port USB application software, etc. Hope you can get some guidance and reference for USB and RF application development by this sample demo.

.

1 System hardware architecture

To form a complete circuit of data transmission, this demo requires the following hardware and its connection shows below:

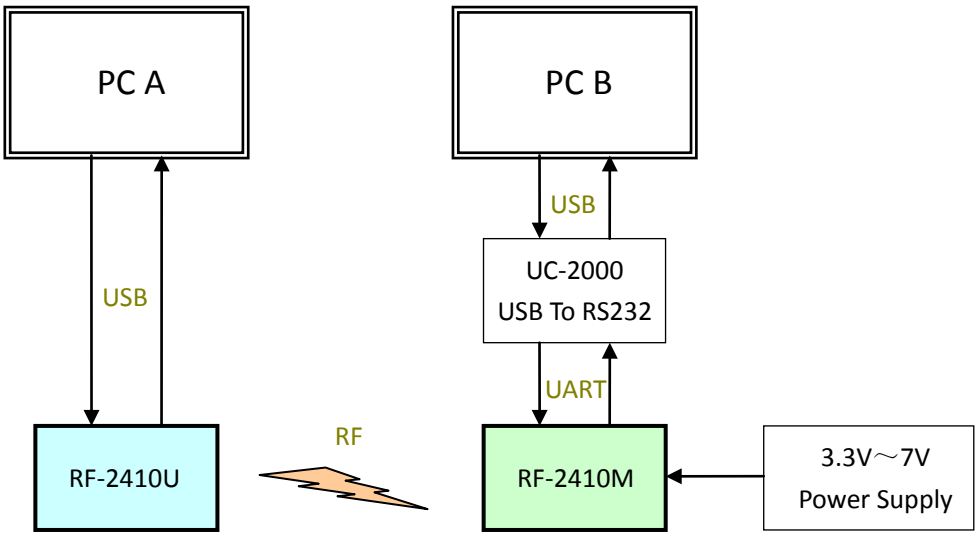

Figure 1 System Hardware Architecture Diagram

RF-2410U connects to PC A by USB port and using USB port to directly supply power.
Due to UART interface, RF-2410M needs to connect PC B by USB to RS232 interface converter (here using INHAOS UC-2000). Besides, RF-2410M requires a 3.3V to 7V power supply separately.
Specifications: RF-2410M has already integrated 3.3V LDO chip. It can directly inherit power input range is 3.3V to 7V.

1.1 RF-2410U module Schematic

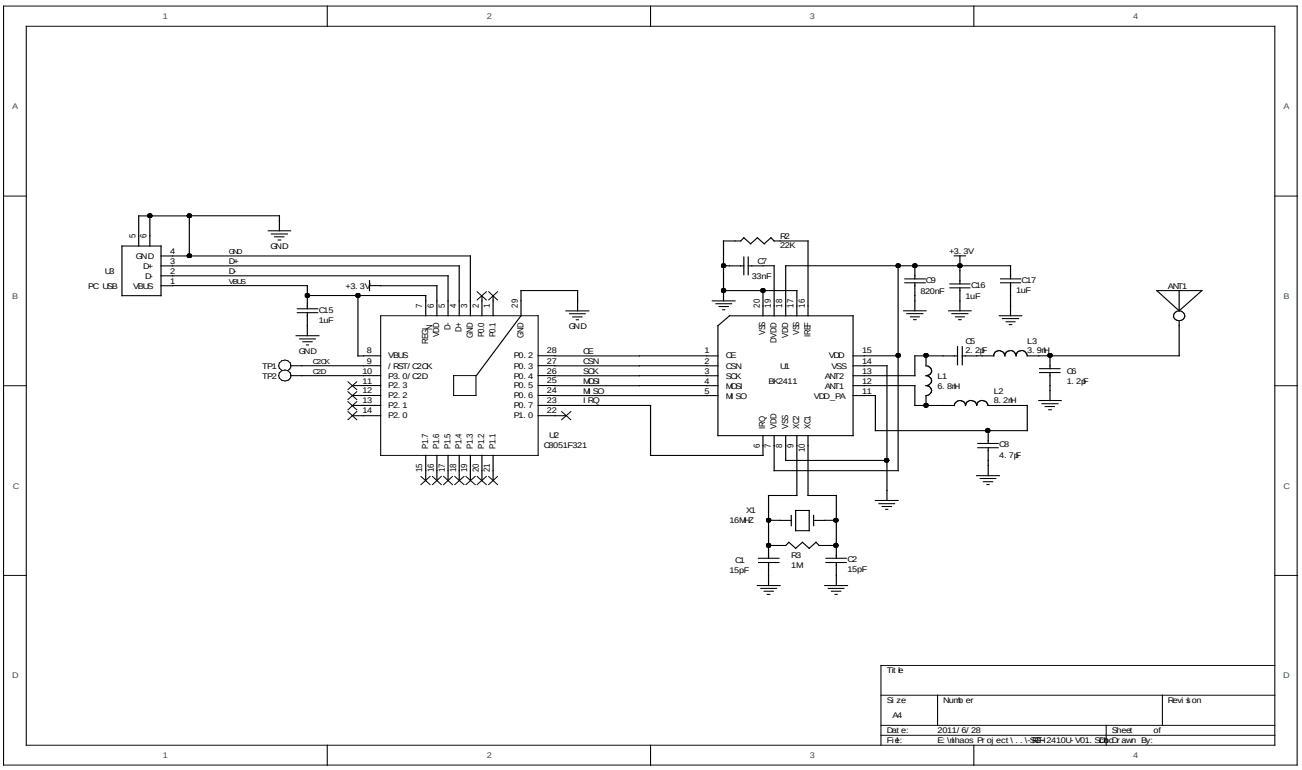

Figure 2 RF-2410U module hardware circuit

1.2 RF-2410M Schematic

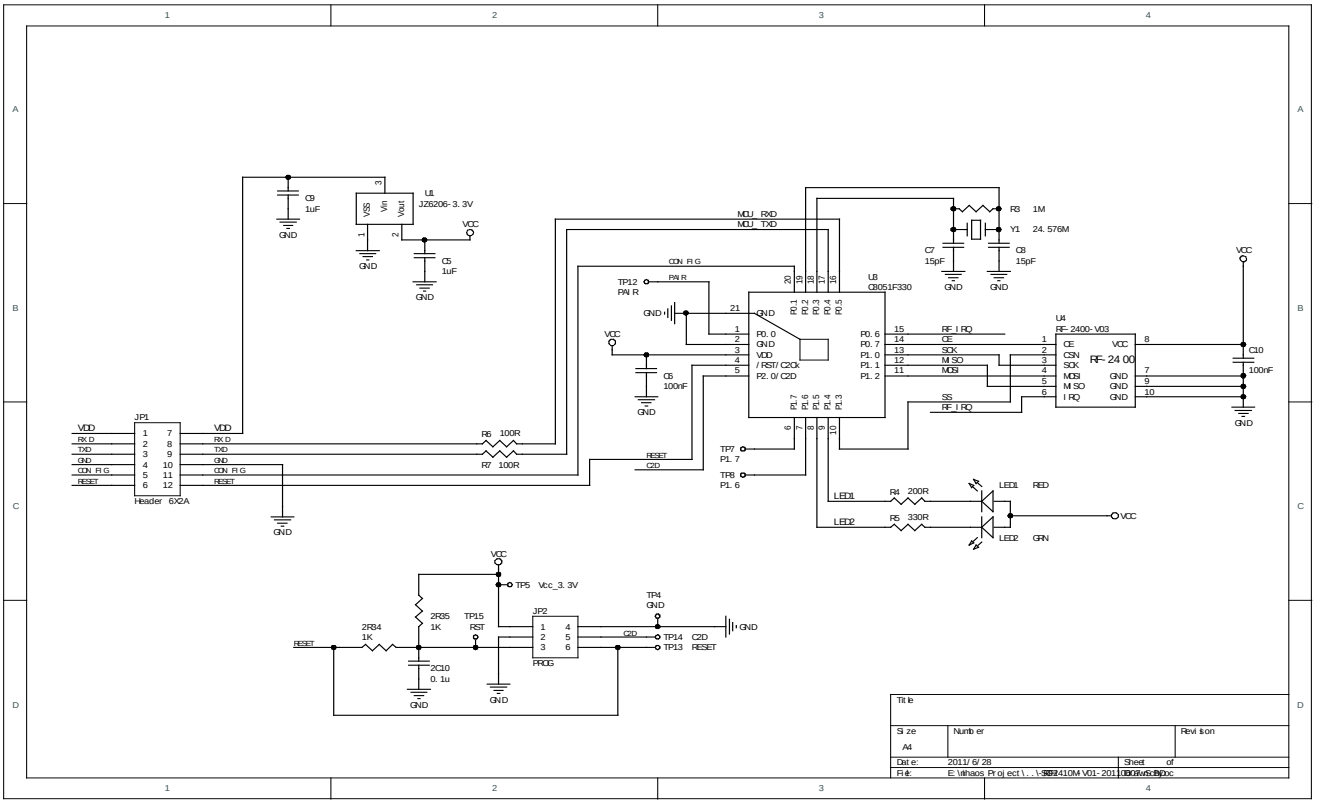

Figure 3 RF 2410M module hardware circuit scheme

2 System software architecture

This system demo aims to build a data transmission channel between PC A and PC B. we only use half-duplex data transmission channel here which means we can send data from PC A to PC B or vice versa. But both ends of data transmission cannot be carried out at the same time.

RF-2410U connects to PC A by USB interface. RF-2410U will be recognized as USBXpress Device after inserting PC A USB port. PC A can transfer data with RF-2410U directly by USB Bulk.

RF-2410U and RF-2410M transfer data by RF interface. To improve the efficiency of transmission system, configure RF air rate of 2Mbps. (Specifications: high bandwidth rates of 2Mbps will shorten the transmission distance between RF-2410U and RF-2410M)

RF-2410M cannot directly connect to PC B instead of via UC-2000 (USB to RS232) to transform. After transforming, RF-2410M and PC B transfer data by UART interface.

Due to adopting UART interface, RF-2410M needs to consider deploying baud rate. Because of the effect of MCU processing speed, RF actual transmission efficiency and space interference, etc, high baud rate will affect the stability of data transmission system. Here deploy baud rate of 57600bps considering MCU processing speed and leave larger tolerance margin for RF data transmission.

To achieve the purpose of visual presentation, this sample system software requires the following element:

- RF-2410U： RF-2410U Demo Firmware
- RF-2410M： RF-2410M Demo Firmware
- PC A port： USB To RF UART Demo
- PC B port： UART serial debugging tools

RF-2410U Demo Firmware、RF-2410M Demo Firmware and USB To RFUART Demo are application implementation codes for this sample; UART serial debugging tools directly use ready-made tools like SSCOM32.exe. The connection of the software is as follows:

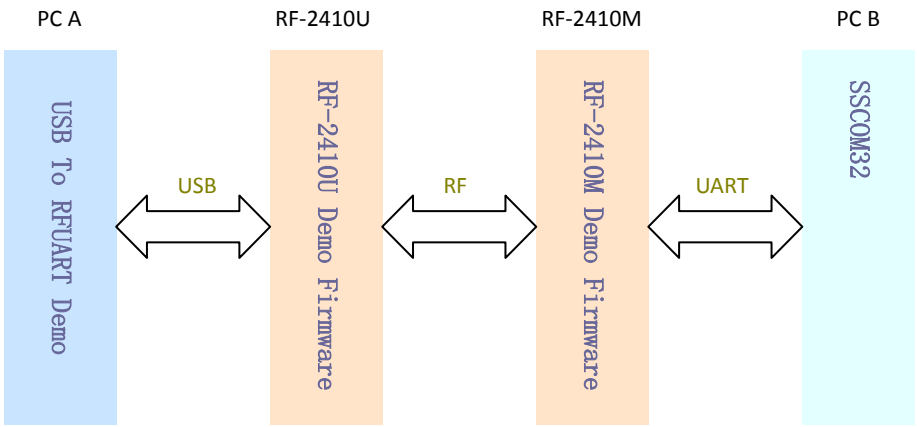

Figure 4 Software system data flow chart

2.1 RF-2410U Demo Firmware software architecture

2.1.1 Software features and main structure description

RF-2410U Demo Firmware finishes the following function:

- Enumerate USB CDC class device, achieve USB Bulk data transmission process
- Achieve RF data transmission, RF transmission air rate is 2Mbps
- Receive data from USB and send via RF
- Receive data from RF and send via USB

RF-2410U Demo Firmware main task is divided into System Initialize、USB Receive Process、USB Send Process、RF Send Process and RF Receive Process, etc. System Initialize includes C8051F320 related initial configuration (timers, ports, system clock, interrupts and other hardware configuration) 、USB initial configuration and BK2421 chip initial configuration; USB Receive Process and USB Send Process achieve USB data transmission process; RF Send Process and RF Receive Process achieve RF data transmission process. And then achieve complete data transmission process by USB and RF data transmission process.

Specifications: to improve RF data receive promptly, RF Receive Process directly put on IRQ Interrupt Service Routine. So RF reception process is high priority than USB.

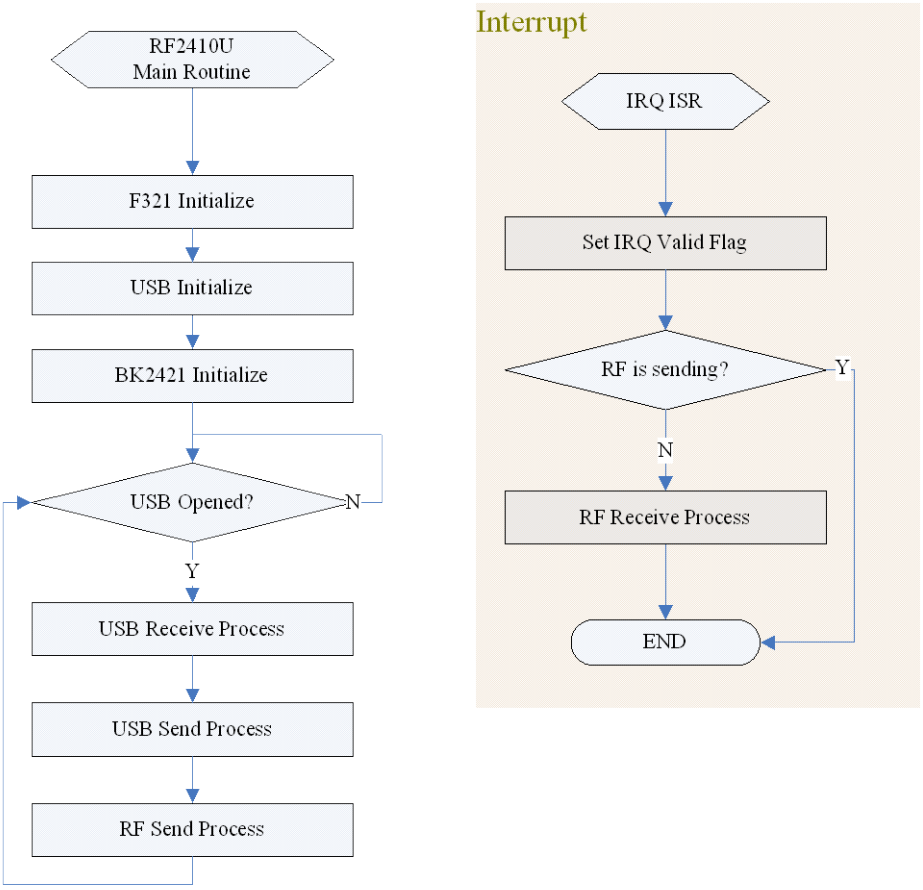

Figure 5 RF-2410U Demo Firmware Main Routine

2.1.2 USB CDC device implement

RF-2410U Demo Firmware's USB device directly adopt USBXpress development kit which provided by Silicon Labs to achieve.

2.1.2.1 USBXpress introduction

The Silicon Laboratories USBXpress® Development Kit provides a complete host and device software solution for interfacing Silicon Laboratories C8051F32x, C8051F34x, and CP210x devices to the Universal Serial Bus (USB). No USB protocol or host device driver expertise is required. Instead, a simple, high-level Application Program Interface (API) for both the host software and device firmware is used to provide complete USB connectivity.

The USBXpress Development Kit includes Windows device drivers, Windows device driver installer, host interface function library (host API) provided in the form of a Windows Dynamic Link Library (DLL), and device firmware interface function library.

Please visit"AN169_USBXpress_Programmers_Guide.pdf" for detail descriptions.

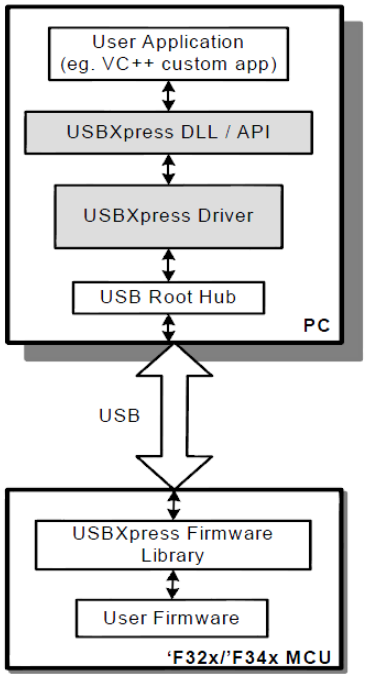

Figure 6 USBXpress data flow chart

USBXpress Firmware Library provides the following USB device interface functions:

- USB_Clock_Start() - Initializes the USB clock
- USB_Init() - Enables the USB interface
- Block_Write() - Writes a buffer of data to the host via the USB
- Block_Read() - Reads a buffer of data from the host via the USB
- Get_Interrupt_Source() - Indicates the reason for an API interrupt
- USB_Int_Enable() - Enables the API interrupts
- USB_Int_Disable() –Disables API interrupts
- USB_Disable() –Disables the USB interface
- USB_Suspend() –Suspend the USB interrupts
- USB_Get_Library_Version() –Returns the USBXpress firmware library version

Besides, it also provides an API virtual interrupt source. The USB API interrupt (interrupt 16 for 'F320/1/6/7 devices and interrupt 17 for 'F34x devices) is a virtual interrupt generated by the USBXpress firmware library whenever user code needs to be notified of a USBXpress event. The events are defined in the description of the *Get_Interrupt_Source* function.

The USBXpress firmware library operates the MCU's USB controller at USB Full Speed, and uses the Bulk Transfer type with a data payload of 64 bytes per packet.

2.1.2.2 USB device initialization

Before initialization, you need to define USB device's VID、PID、Manufacturer String、Product String and SerialNumber String. RF-2410U Demo Firmware defined as follows:

```
//Vendor ID
#define USB_VID 0x2156
//Product ID
#define USB_PID 0xC002

//Manufacture String defination
#define STR1LEN sizeof ("INHAOS Technology") * 2
code const unsigned char USB_MfrStr [] =
{
      STR1LEN, 0x03,
      'I', 0, 'N', 0, 'H', 0, 'A', 0, 'O', 0, 'S', 0, ' ', 0,
      'T', 0, 'e', 0, 'c', 0, 'h', 0, 'n', 0, 'o', 0, 'l', 0, 'o', 0, 'g', 0, 'y', 0,
};

//Product String defination
#define STR2LEN sizeof ("USB To RF UART Demo") * 2
code const unsigned char USB_ProductStr [] =
{
    STR2LEN, 0x03,
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'T', 0, 'o', 0, ' ', 0,
    'R', 0, 'F', 0, ' ', 0, 'U', 0, 'A', 0, 'R', 0, 'T', 0, ' ', 0,
    'D', 0, 'e', 0, 'm', 0, 'o', 0
};

//Serial number defination
unsigned char data HIDSNBuffer[] ={
      10, 3,
      '0', 0, '0', 0, '0', 0, '3', 0,
};
```

Call USB_Initialize()function for USB device initialization. It will automatically complete all the listed action required by USB CDC device, you don't have to do any actions in Firmware. After completing calling USB_Initialize() function, host will indicates succeeded in finding USBXpress device and indicates to install its driver. USB_Initialize() function achieve codes are as follows:

```
/*****************************************************************************
Function:        void USB_Initialize( void )
Parameter:
                    None
Return:
                    None
```

```
Description:
                    Initialize USB
                    Set USB vid,pid,manufacture strings,product strings,serial number strings
                    USB power consume 100mA
                    USB power supply
***********************************************************************************/
void USB_Initialize( void )
{
    //Enables the internal oscillator,
    //initializes the clock multiplier,
    //and sets the USB clock to 48 MHz for USB full speed operation
    USB_Clock_Start();

    //Enables the USB interface,
    //the USB clock recovery feature,
    //and the use of Device Interface Functions
    USB_Init(USB_VID, USB_PID, USB_MfrStr, USB_ProductStr, HIDSNBuffer,0x32,0x80,0x0001);

    //Enables the USB API to generate interrupts
    USB_Int_Enable();
}
```

2.1.2.3 USB device API interrupt

To use USBXpress Firmware Library for data transmission, need to achieve USB API interrupt service functions. In this function, you need to call Get_Interrupt_Source function first to get the interrupt source which is current generated. And then do some relevant handling base on the current interrupt source received.

Get_Interrupt_Source function return back the following interrupt source information:

■ 0x00		No USB API Interrupts have occurred
■ 0x01	USB_RESET	USB Reset Interrupt has occurred
■ 0x02	TX_COMPLETE	Transmit Complete Interrupt has occurred
■ 0x04	RX_COMPLETE	Receive Complete Interrupt has occurred
■ 0x08	FIFO_PURGE	Command received (and serviced) from the host to purge the USB buffers
■ 0x10	DEVICE_OPEN	Device Instance Opened on host side
■ 0x20	DEVICE_CLOSE	Device Instance Closed on host side
■ 0x40	DEV_CONFIGURED	Device has entered configured state
■ 0x80	DEV_SUSPEND	USB suspend signaling present on bus

RF-2410U Demo Firmwar USB data transfer operations need to use DEVICE_OPEN、DEVICE_CLOSE、TX_COMPLETE、RX_COMPLETE interrupt source. They are used to indicate open the device, Close the device, complete transferring and receive.

Related codes are as follows:

```
/***********************************************************************************
Function:       void USB_API_TEST_ISR(void) interrupt 16
Parameter:
                    None
Return:
                    None
Description:
```

```
                    USB API interrupt service
*******************************************************************************/
void USB_API_TEST_ISR(void) interrupt 16
{
    UINT8 INTVAL = Get_Interrupt_Source(); //Determine type of API interrupts

    if(INTVAL & USB_RESET)      //Bus Reset Event
    {
    }

    if(INTVAL & DEVICE_OPEN)    //Device opened on host
    {
        g_usbopen = TRUE;                   //Set flag to indicate usb device opened
    }

    if(INTVAL & TX_COMPLETE)    //TX Complete
    {
        g_usb.sendirq = TRUE;   //Set the send over irq
    }
    if(INTVAL & RX_COMPLETE)    //RX Complete
    {
        g_usb.recvirq = TRUE;   //Set the received complete irq
    }

    if(INTVAL & FIFO_PURGE)     //Fifo purged
    {
    }

    if(INTVAL & DEVICE_CLOSE)   //Device closed
    {
        g_usbopen = FALSE;      //Clear the device open flag
    }
}
```

2.1.3 Data cache planning and management

To improve the efficiency of data transmission, RF-2410U Demo Firmware's USB and RF data transmission are separated. They transfer data by two FIFO buffer:

- The data received from USB will save in RF SEND FIFO. RF send tasks read data from the buffer and send away.
- The data received from RF will save in USB SEND FIFO . USB send tasks read data from the buffer and send away.

C8051F321 RAM size is 2304 bytes (1K +256 +1 K USB FIFO), except 1K bytes occupied by USB FIFO. The available RAM size which can be absolutely used is 1K+256. So RF-2410U Demo Firmware put RF Send FIFO and USB Send FIFO into external RAM area and set the size of 400 bytes. It occupies XRAM 800bytes in total. Internal RAM's 256 bytes and the remaining bytes will for other use.

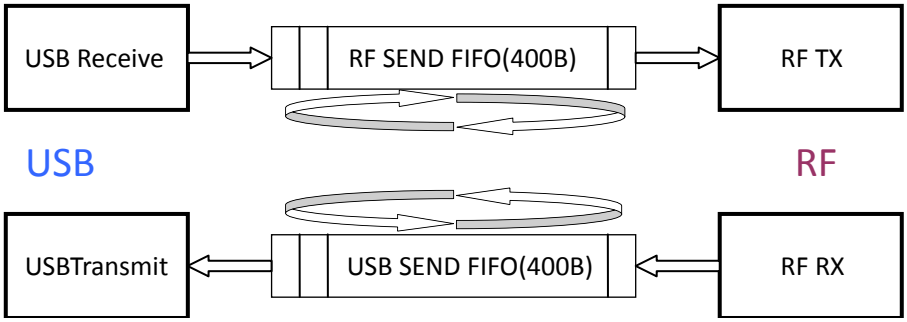

Figure 7 USB to UARTDemo data flow chart

Both writing and reading of RF SEND FIFO and USB SEND adopt cycling mode and manage by a unified structure. This structure involves buffer's writing location, reading location and buffer's data length. The data structure is defined as follows:

```
/*A struct defination for USB transfer*/
typedef struct
{
    UINT16 pos_w;          //write pointer for usb receive fifo
    UINT16 pos_r;          //read pointer for usb receive fifo
    UINT16 length;         //data length for usb receive fifo
      …
}S_USB;
/*A struct defination for RF transfer*/
typedef struct
{
    UINT16 pos_w;          //write pointer for RF send fifo
    UINT16 pos_r;          //read pointer for RF send fifo
    UINT16 length;         //data length for RF send fifo
      …
}S_RF;
```

When writing data to FIFO, it will increase pos_w position and length. When pos_w reaches the end of FIFO, we need to set pos_w of 0 again and point to FIFO's first position, write to the buffer by circulating. When reading data from FIFO, it increases pos_r position but decrease length. When pos_r reaches the end of FIFO, it sets pos_r of 0 again and point to FIFO first position, read the buffer by circulating.

2.1.4 USB data transmission process

2.1.4.1 USB data receive process

RF-2410U Demo Firmware USB data receive process mainly responsible for reading data received from USB hardware FIFO. And then write the data to RF SEND FIFO. Please refer to "3.1.3.1 Data cache planning and management" for RF SEND FIFO writing operation.

```
/*****************************************************************************
Function:       void USB_Receive_Process( void )
Parameter:
                None
Return:
                None
Description:
```

```
                Query usb data receive flag periodically, and read out the received data
                and fill it into RF send fifo, when usb receive flag is valid
*******************************************************************************************/
void USB_Receive_Process ( void )
{
    UINT8 readlen;
    UINT8 *pPkt;

    if( g_usb.recvirq ) //data received from usb
    {
        g_usb.recvirq = FALSE;
        //Read data from usb hard's receive fifo
        readlen = Block_Read( (UINT8 *)g_usb_packet, PACKET_LEN_USB );

        //Copy read data to RF send fifo, use loop mode
        pPkt = g_usb_packet;
        while( readlen-- )
        {
            g_rf_fifo[g_rf.pos_w++] = *pPkt++;
            if( g_rf.pos_w >= FIFO_LEN_RF )
            {
                g_rf.pos_w = 0;
            }
            g_rf.length++;
        }

        g_rf.IntervalTick = 1;
    }
}
```

2.1.4.2 USB data transmission process

USB data transmission process mainly responsible for detecting USB SEND FIFO valid length. When the length is greater than 0, start sending process: read data from USB SEND FIFO and fill in the current package. And then call Block_Write() function to write the packet to USB interface. Please refer to "3.1.3.1 Data cache planning and management" for RF SEND FIFO reading operation. USB data transmission process main workflow shows below:

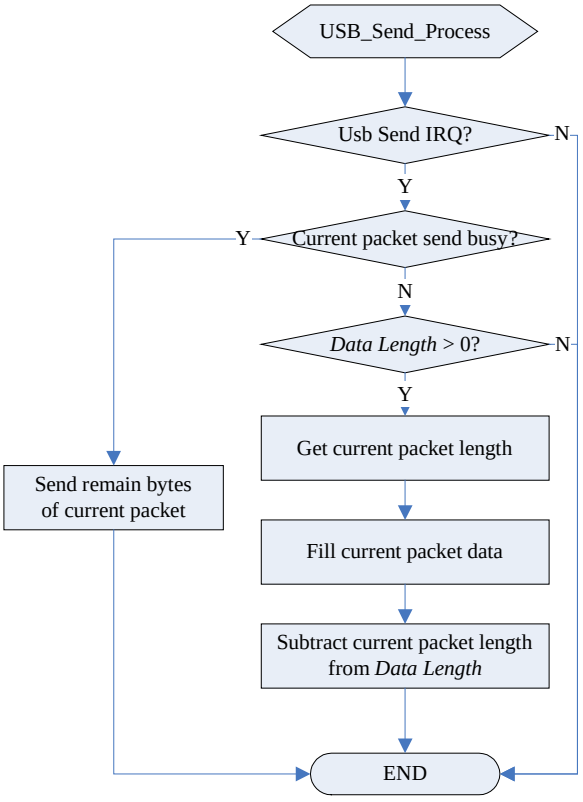

Figure 8 RF-2410U USB Send Process Flow

RF-2410U Demo Firmware USB data transmission process achieved by Usb_SendToPC() function, the codes are as follows:

```
/*******************************************************************************
Function:       void USB_Send_Process( void )
Parameter:
                None
Return:
                None
Description:
                Read data from usb send fifo once it's data length <> 0, and send it to PC
                according the usb
*******************************************************************************/
void USB_Send_Process( void )
{
    UINT16 sendlen;
    UINT8 bytessent;
    UINT8 *pPkt;

    //Check wether or not hardware operation for usb packet send is finished?
    if( !g_usb.sendirq )
    {
        return;
    }

    //packet send is in progress?
    if( g_usb.sendbusy )
    {
        //Is there data need to be send out?
        if( g_usb.bytestosend )
        {
            //write data to usb's hard fifo
            sendlen = Block_Write( (UINT8 *)g_usb.sendptr, g_usb.bytestosend );

            //Calculate remain data bytes
            if( g_usb.bytestosend > sendlen )
            {
                g_usb.bytestosend -= sendlen;
            }
            else
            {
                g_usb.bytestosend = 0;
                g_usb.sendbusy = FALSE;
            }
        }
    }
    else
    {
        //usb send fifo is not empty?
        if( g_usb.length )
        {
            ENTER_CRITICAL();
            //Check whether or not the usb send fifo overflow
            //And force the read pointer equal to the write pointer when overflow occurs
```

```
                if( g_usb.length > FIFO_LEN_USB )
                {
                    g_usb.length = FIFO_LEN_USB;
                    g_usb.pos_r = g_usb.pos_w;
                }
                sendlen = g_usb.length;
                EXIT_CRITICAL();

                //Limit send length: make it less than PACKET_LEN_USB, and also less than the actual
                //data length in the USB send fifo
                if( sendlen > PACKET_LEN_USB )
                {
                    sendlen = PACKET_LEN_USB;
                }

                //Prepare for packet sending
                g_usb.bytestosend = sendlen;
                g_usb.sendbusy = TRUE;
                g_usb.sendptr = g_usb_packet;

                //Fill data to usb packet buffer for send from usb send fifo
                pPkt = g_usb_packet;
                bytessent = sendlen;
                while( sendlen-- )
                {
                    *pPkt++ = g_usb_fifo[g_usb.pos_r++];
                    if( g_usb.pos_r >= FIFO_LEN_USB )
                    {
                        g_usb.pos_r = 0;
                    }
                }

                //Subtract the buffer valid data length
                ENTER_CRITICAL();
                g_usb.length -= bytessent;
                EXIT_CRITICAL();
            }
        }
    }
```

2.1.5 BK2421 configuration and initialization

BK2421 is a RF chip which highly compatible with Nordic nRF24L01. Its register's configuration is basically consistent with nRF24L01. The biggest difference is the former has two BANK register (Bank0 , Bank1) while nRF24L01 just has one. So BK2421 power-on initialization not only needs to initialize Bank0, also need to initialize Bank1.

RF-2410U Demo Firmware require the following element for BK2421 initialization configuration

- Enable RX_DR, TX_DS, MAX_RT interrupt
- Open 2-byte CRC check
- Enable data pipe 0
- Enable auto acknowledgement data pipe 0

- RX/TX Address field width 5 bytes
- Auto Retransmission Delay:250us
- Auto Retransmission Count: 6
- Air Data Rate:2Mbps
- Set RF output power: 5dBm
- Enable dynamic payload length data pipe 0.
- Enables Dynamic Payload Length

2.1.5.1 SPI interface operation

BK2421 only provides SPI interface for register's writing and reading operation. So RF-2410U Demo Firmware needs to achieve SPI's writing and reading operation to smoothly access to BK2421 register.

Part of RF-2410U SPI hardware circuit shows below:

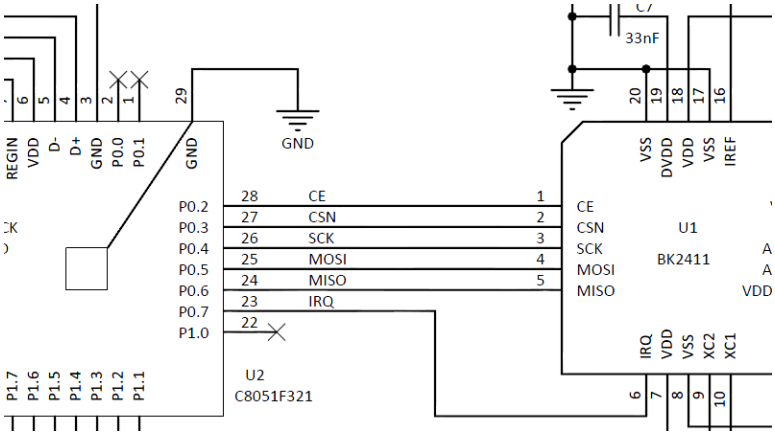

Figure 9 SPI Hardware Circuit

Due to C8051F321 chip's hardware SPI requires order for SCK\MISO\MOSI interface, just like the figure below:

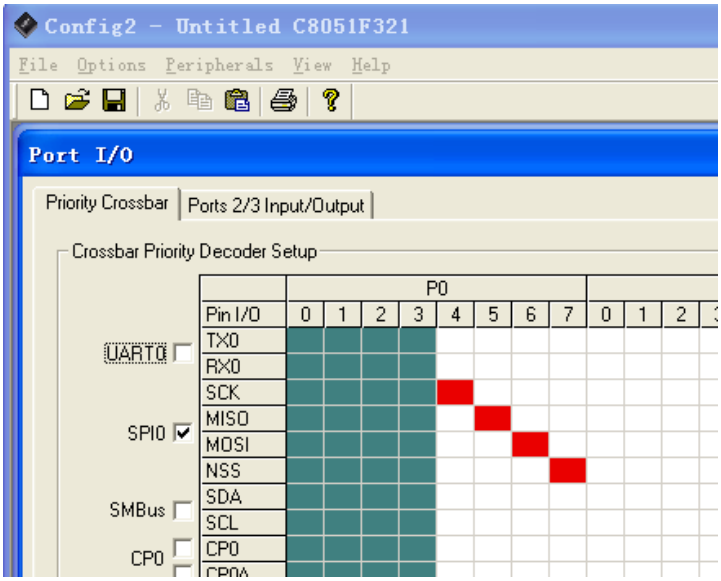

Figure 10 C8051F320 SPI Ports Config

As you can see from Figure 9 and Figure 10, SPI configuration require the order of MISO and MOSI is contrary with RF-2410U interface's order. So RF-2410U module cannot use hardware SPI, only can be achieved by software simulation SPI. The codes are as follows:

Firstly define SPI related pins

```
//Definations for SPI ports
sbit CE      =   P0^2;
sbit CSN     =   P0^3;
sbit SCK     =   P0^4;
sbit MOSI    =   P0^5;
sbit MISO    =   P0^6;
```

Function SPI_RW_Byte() simulate SPI sequential operation by software. Realize write one byte from SPI BUS and read one byte at the same time.

```
/**************************************************************************************
Function:          UINT8 SPI_RW_Byte( UINT8 Data )
Parameter:
                   Data       [IN]      Data byte will be write according spi bus
Return:
                   Return byte from spi bus
Description:
                   Write a byte to and Read a byte from spi bus
**************************************************************************************/
UINT8 SPI_RW_Byte( UINT8 Data )
{
    UINT8 rData = 0;
    UINT8 i = 0x80;

    while( i )
    {
        MOSI = ( i & Data );
        SCK = 1;
        if ( MISO )
        {
            rData |= i;
        }
        SCK = 0;
        i >>= 1;
    }

    return rData;
}
```

2.1.5.2 BANK0 Initialize

RF-2410U Demo Firmware adopts pre-defined array (in code storage area) for BANK0 register configuration. And then read array's register and the corresponding value by programming circulation. Deploy for BANK0 register.

RF-2410U Demo Firmware Bank0 initialization configuration process is as follows:

Firstly pre-define array Bank0_RegAct saved in DYNPD and FEATURE register's value and array Bank0_Reg_Init storage and other BANK0 register's value.

Specifications: Before writing to DYNPD and FEATURE register, you need to activation the register by ACTIVATE command. Otherwise writing operation will be invalid.

```
//Bank0_Register Configuration Operate==========================================
```

```
const UINT8 Bank0_RegAct[2][2] =
{
    {DYNPD,      0x01},      //Enable pipe 0, Dynamic payload length
    {FEATURE,    0x04}       //EN_DPL= 1, EN_ACK_PAY = 0, EN_DYN_ACK = 0
};

code UINT8 Bank0_Reg_Init[21][2] =
{
    {CONFIG,          0x0F},  //PRX,CRC=2,ENCRC,POWRUP;
    {EN_AA,           0x01},  //data pipe 0 ACK;
    {EN_RXADDR,       0x01},  //RX address data pipe 0;
    {SETUP_AW,        0x03},  //RX/TX address width 5B
    {SETUP_RETR,      0x06},  //auto retrasmit count 6,delay 250us;
    {RF_CH,           0x60},  //2400+0x60;
    {RF_SETUP,        0x1f},  //air rate = 2Mbps,high gain,output power = 5dBm;
    {STATUS,          0x70},  //Clear interrupt flag;
    {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 Payload Length = 32
    {RX_PW_P1,        0x20},
    {RX_PW_P2,        0x20},
    {RX_PW_P3,        0x20},
    {RX_PW_P4,        0x20},
    {RX_PW_P5,        0x20},
    {FIFO_STATUS,     0x11}
};
```

Then complete BANK0 initialization by calling BANK0_Init() function. The codes are as follows:

```
/**************************************************************************
Function:           void BANK0_Init( void )
Parameter:
                    None
Return:
                    None
Description:
                    BANK0 register initialize operation
**************************************************************************/
void BANK0_Init( void )
{
    UINT8   i   = 0;
    UINT8   k   = 0;
    UINT8   Rt = 0;

    //Config Bank0 Register
    for( i = 0; i < 21; i++ )
    {
        SPI_Write_Reg( W_REGISTER | Bank0_Reg_Init[i][0], Bank0_Reg_Init[i][1] );
        SPI_Read_Reg( Bank0_Reg_Init[i][0] );
    }
```

```
    //Write RX/TX Address
    RF_SET_RX_ADDR( &URX_Address[0] );
    RF_SET_TX_ADDR( &URX_Address[0] );

    //Before config DYNPD and FEATURE register, the ACTIVATE command need to be write
    k = SPI_Read_Reg( FEATURE );
    if( k == 0 )
    {
        SPI_Write_Reg( ACTIVATE,   0X73 );
    }

    //Now Config DYNPD and FEATURE register
    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] );
    }
}
```

2.1.5.3 BANK1 Initialize

BEKEN Co. has detail description for BANK1 register except register 4、5、7、8. So you just need to follow BEKEN Co. document to write the corresponding value for BANK1 initialization. RF-2410U Demo Firmware BANK1 initialization process is as follows:

Firstly, pre-define Bank1_Reg0_Reg13 storage BANK1 register 0 to 13 configuration value, Bank1_Reg14 storage BANK1 register 14 configuration value;

```
//Bank1 Register Configuration Operate==============================================
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
};
```

Then BANK1_Init() function responsible for writing pre-defined register array to the

corresponding register, complete initialization for Bank1. The codes are as follows:

```
/*****************************************************************************
Function:                   void BANK1_Init( void )
Parameter:
                            None;
Return:
                            None;
Description:
                            BANK1 regiter initialize operation
*****************************************************************************/
void BANK1_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_Buf( 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_Buf( W_REGISTER | i, &(Buff[0]), 4 );
    }
    //Configuration   Bank1 Register 14==============================================
    SPI_Write_Buf( 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_Buf( W_REGISTER | 0X04, &(Buff[0]), 4 );

    Buff[0] &= 0XF9;
    SPI_Write_Buf( W_REGISTER | 0X04, &(Buff[0]), 4 );
}
```

2.1.6 BK2421 PTX and PRX hardware process

We have to know hardware PTX and PRX implementation process before using BK2421 for data transmission.

Because of highly compatible between BK2421 and Nordic RF chip, PTX and PRX implementation

process is basically consistent with NRF24L01, so we can refer to NRF24L01 PTX and PRX process.

As Figure 11 shows NRF24L01 PTX hardware data send process while Figure 12 shows for NRF24L01 PRX hardware data receive process. Recommend “nRF24L01 Single Chip 2.4GHz Transceiver Product Specification”.

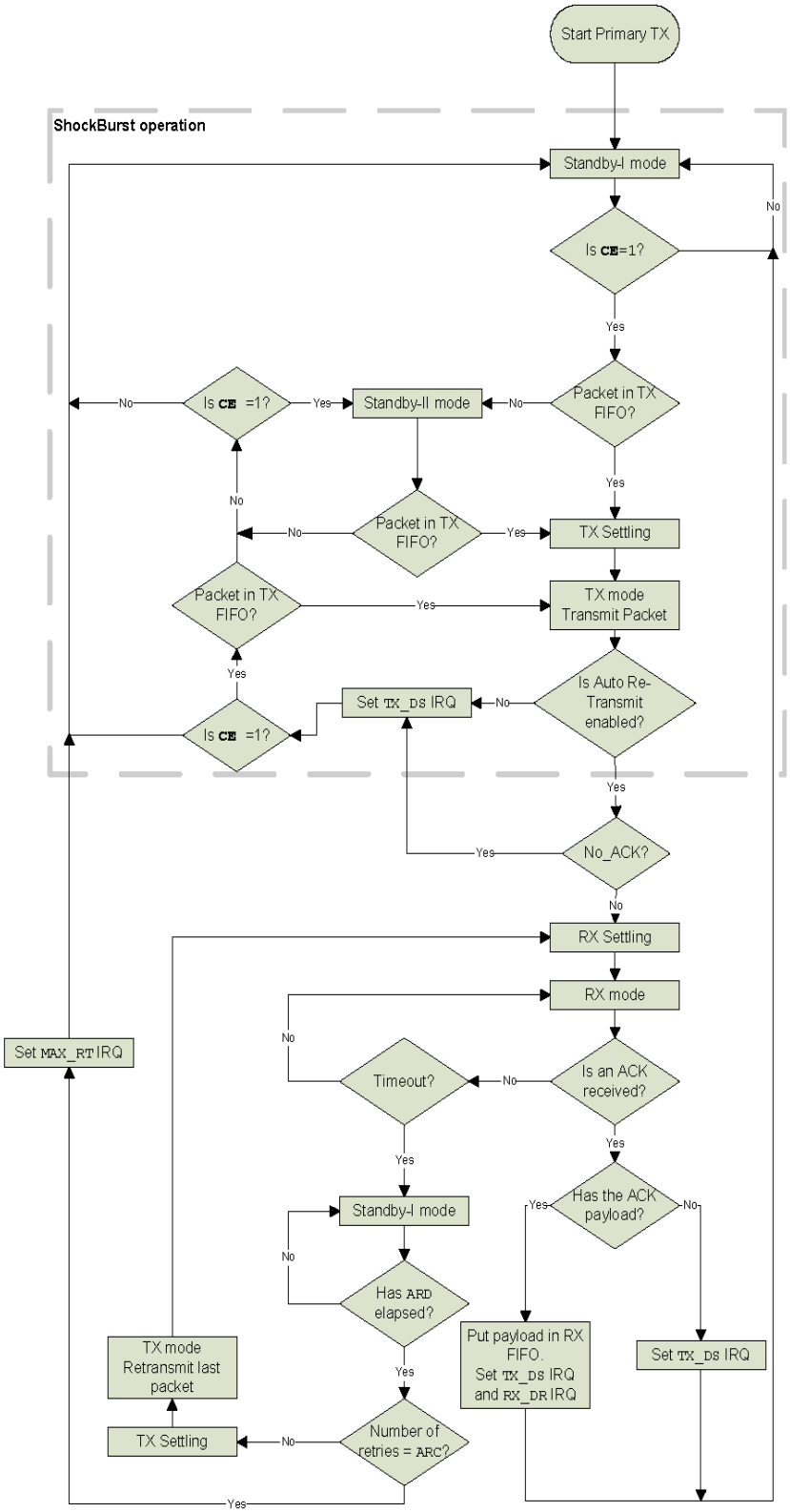

Figure 11 NRF24L01 PTX operations in Enhanced ShockBurst™

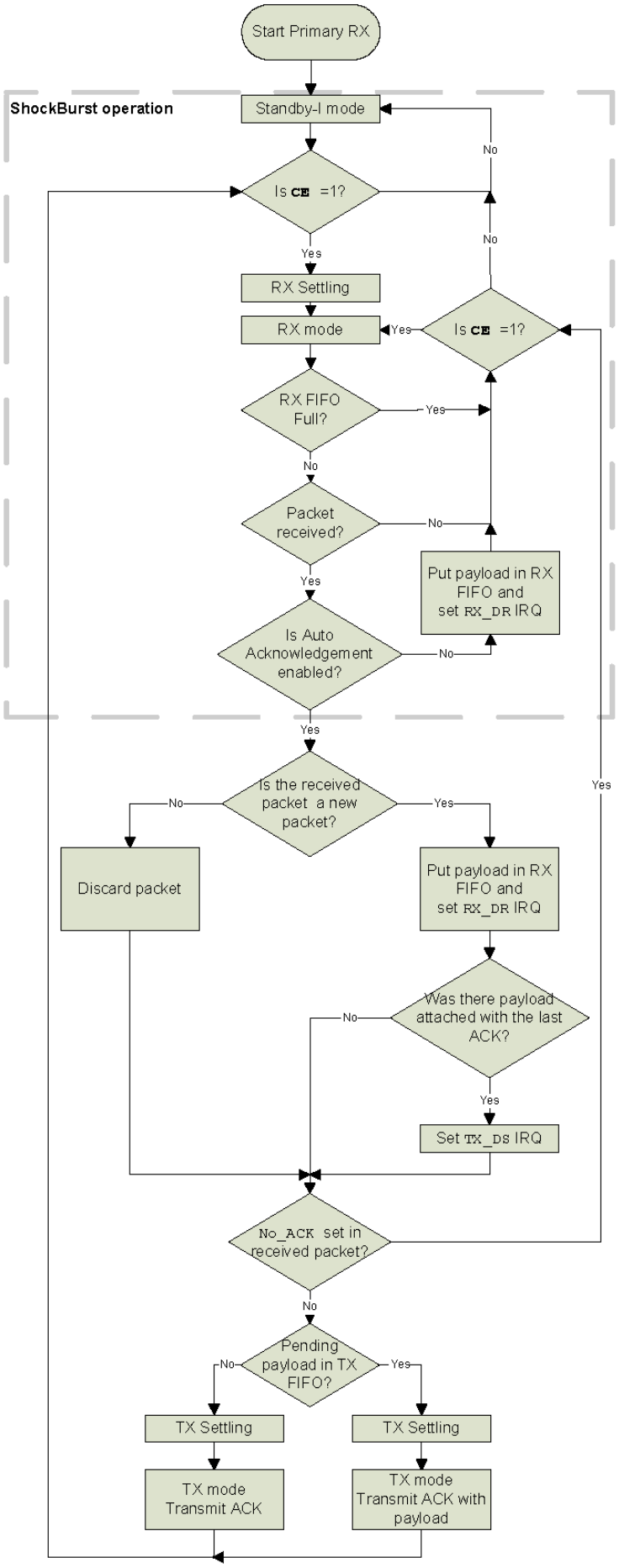

Figure 12 NRF24L01 PRX operations in Enhanced ShockBurst™

Of course there is also some difference between BK2421 and NRF24L01. Table 1 describes their differences in data transmission, recommended “BK2401/BK2421 Application Notes V2.0”.

No.	Difference	nRF24L01	BK2401/BK2421	Remark
1	Order requirement for CSN	CSN finish writing data and pull up no later than the CLK operation	After CSN writing data in low state, pull up needs to later than CLK or at least half CLK time and CSN rising time cannot greater than 100ns.	
2	CE set low，RX_DR interrupt by 0	After CE pull down, interrupt won’t be clear	When CE pulls down, RX_DR interrupt will atomically clear by o. So we must handle interrupt before CE pulling down	
3	PTX、PRX interrupt trigger time	When PRX send consist of PAYLOAD ACK's PRX port: TX_DS is one packet later than RX_DR and set 1; PTX port: set RX_DR and TX_DS 1 at the same time.	When PRX send consist of PAYLOAD ACK's PRX port: set RX_DR and TX_DSTX_DS 1 at the same time; PTX port: firstly set TX_DS 1; RX_DR is 2-3us later than TX_DS set 1.	You need to be careful if ACK consists PAYLOAD
4	PTX device overflow handling for FIFO	If RX's third FIFO fully received, the data packet won't cover FIFO's data instead □of triggering RX_DR interrupt	If RX's third FIFO fully received, the data packet won't cover FIFO's data and won't trigger RX_DR interrupt.	You need to be careful if ACK consists PAYLOAD

Table 1 BK2401/BK2421 and nRF24L01differece (only for data transmission)

2.1.7 RF data transmission process

Because RF transmission is half-duplex which is only for one-way transmission once. So RF-2410U Demo Firmware RF data transmission also only achieves half-duplex data transmission. RF-2410U Demo Firmware also adopts BK2421’s hardware auto acknowledge function to ensure the reliability of transmission.

When using auto acknowledge function, sender’s underlying hardware will automatically switch to receive mode await the counterpart reply to ACK packet after sealing the packet and send. The other side will automatically switch to sending mode to send an ACK packet after receiving the packet. When the counterpart haven’t receive ACK data packet, the underlying hardware will automatic retransmission (the times of retransmission and interval defined in SETUP_RETR register). When retransmission reaches fixed number, indicates failed to send. (pull down IRQ and Set MAX_RT flag). When receive the ACK data packet, indicates successfully sending.

2.1.7.1 RF packet format defined

RF-2410U Demo Firmware and RF-2410M Demo Firmaware RF data transmission adopt sub-packet transmission mode. Data packet format defined as follows:

Field	SN	Length	PKT_SN	Parameter
Definition	RF send serial number	Parameter 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

Table 2 RF Data Packet Format

As the table above shows, RF data packet maximum length is 28Bytes (must be less than RF TX/RX FIFO Length=32Bytes), Parameter length depends on actual situation select from 0 to 25 bytes. So the whole data packet actual length also includes dynamic packet length, the range is between 3 to 28 bytes.

2.1.7.2 RF data packet send and receive process

The operation process of RF-2410U Demo Firmware sends a data packet by RF:

- Pull down CE pin
- Switch RF to send mode
- Write data to TX FIFO by W_TX_PAYLOAD command
- Pull up CE pin, duration must be greater than 10us and then pull down CE pin
- The hardware start sending data packet
- Wait for finishing sending. The underlying hardware will set corresponding interrupt flag and pull down IRQ pin to generate interrupt after finishing sending. If successfully sent, set STATUS_TX_DS of 1; if failed, set STATUS_MAX_RT of 1. To avoid IRQ interrupt haven't been generated, sometimes need to add timeout.

RF-2410U Demo Firmware sends data packet by RF through RF_SendPacket() function. The codes are as follows:

```
/**************************************************************************************
Function:       void RF_SendPacket( void )
Parameter:
                None
Return:
                None
Description:
                Send out a packet via RF
                And clear the send packet length once success
**************************************************************************************/
void RF_SendPacket( void )
{
    UINT8 sta;
    UINT8 outflag = FALSE;
    UINT16 temp = 100;

    ENTER_CRITICAL();
    g_rf.sendmode = TRUE;
    EXIT_CRITICAL();

    SwitchtoTXMode();                       //Switch RF to TX mode
```

```
    CLR_CE();
    SPI_Write_Buf(W_TX_PAYLOAD, (UINT8 *)&g_rf_packet, sizeof(S_RF_PKT)); // Writes data to TX FIFO
    g_rf.irqvalid = FALSE;
    SET_CE();
    while( temp-- );                        //Wait for Time > 10us
    CLR_CE();

    //Wait for send over
    g_rf.sendtick = 0;
    while( 1 )
    {
        if( g_rf.irqvalid )
        {
            sta = SPI_Read_Reg( STATUS );       // read register STATUS's value

            if( sta & STATUS_MAX_RT )           //if send fail
            {
                RF_FLUSH_TX();
                outflag = TRUE;
            }
            if( sta & STATUS_TX_DS )            //TX IRQ?
            {
                outflag = TRUE;
                g_rf.packetlen = 0;              //Now send success, clear the send packet length
            }

            RF_CLR_IRQ( sta );                  // clear RX_DR or TX_DS or MAX_RT interrupt flag
            g_rf.irqvalid = FALSE;

            if( outflag )
            {
                break;
            }
        }
        else if( g_rf.sendtick >= 2 )           //if timeout
        {
            RF_FLUSH_TX();
            break;
        }
    }

    SwitchtoRXMode();                       //Switch to RX mode


    ENTER_CRITICAL();
    g_rf.sendmode = FALSE;
    EXIT_CRITICAL();
}
```

The operation process of RF-2410U Demo Firmware receives a data packet:

- Switch RF to receive mode, the underlying hardware start detecting data
- When the underlying hardware receive the packet, set RX_DR flag and pull down IRQ

pin to generate interrupt

- When detect IRQ interrupt, application program read STATUS register and judge RX_DR is 1 or not. If 1, read RX FIFO's receive packet by R_RX_PAYLOAD command.
- Completing reading, clear STATUS RX_DR bit by 0, hardware will pull up IRQ pin automatically

Please refer to "2.1.7.4RF data receive process" for RF-2410U Demo Firmware RF data receive process codes.

2.1.7.3 RF data send process

RF-2410U Demo Firmware RF data send process responsible for sending data which receive from USB and save in RF SEND FIFO to RF-2410M. Send process shows below:

When RF SEND FIFO data's valid length greater than or equal to Packet Length, or send interval is more than 3ms, read one packet from RF SEND FIFO (if less than one packet, use actual length) , then send data. If failed, re-send data until successful. The flow shows below:

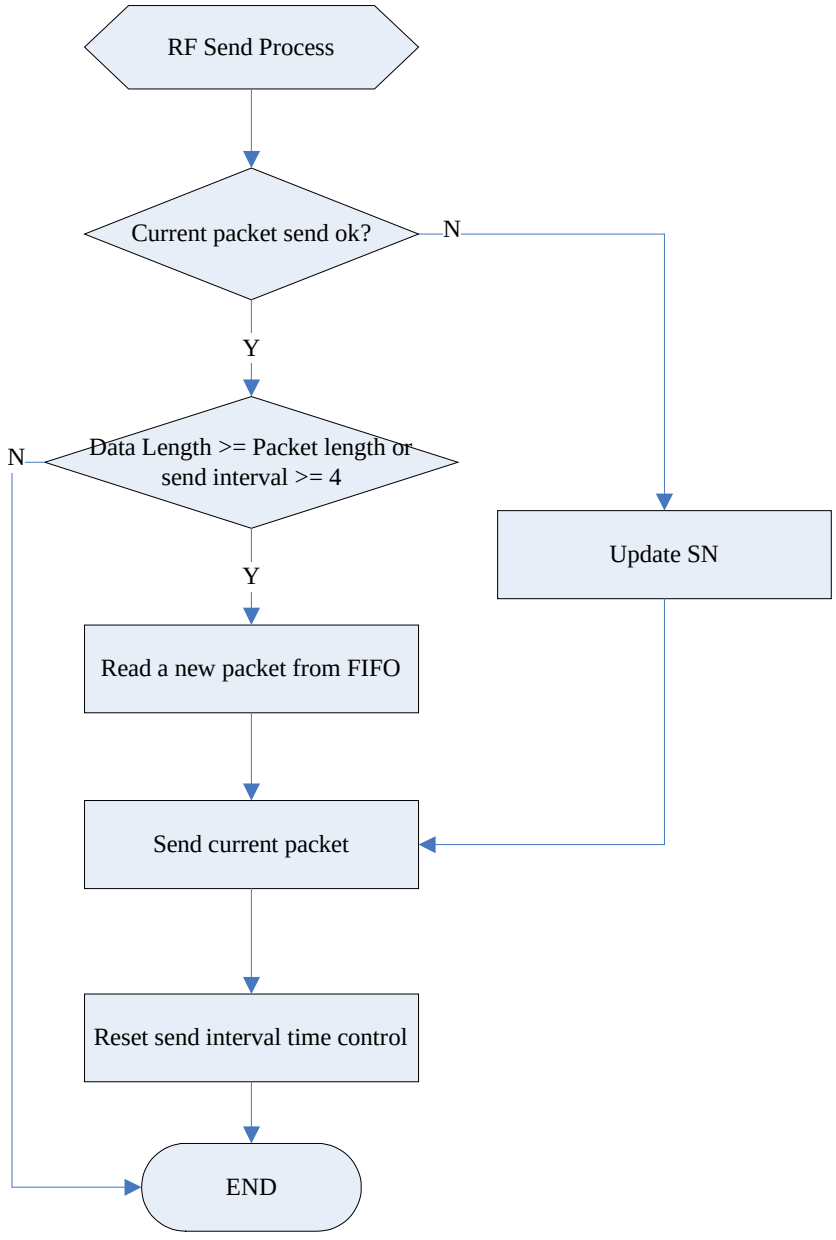

Figure 13 RF-2410U RF Send Process Flow

RF-2410U Demo Firmware achieve send process above by RF_Send_Process() function. The sample codes are as follows:

```
/*******************************************************************************
Function:          void RF_Send_Process( void )
Parameter:
                    None
Return:
                    None
Description:
                    Firstly, check the valid data length of RF SEND FIFO;
                    Secondly, read data from RF SEND FIFO and fill into the RF send packet;
                    Thirdly, send out the RF packet with retransmit till success.
*******************************************************************************/
void RF_Send_Process( void )
{
    if( g_rf.packetlen )      //Last packet sendover ?
    {
        //Resend last packet and update the send sn
        g_rf_packet.sn = g_rf.sn_rf++;

        RF_SendPacket();
    }
    else if( g_rf.length >= RF_PKT_LEN || g_rf.IntervalTick >= 4 )
    {
        //Fill a new packet data
        RF_FillPacket();

        RF_SendPacket();

        //Check wether rf send fifo has data
        if( g_rf.length )
        {
            //keep time control on
            g_rf.IntervalTick = 1;
        }
        else
        {
            //close time control
            g_rf.IntervalTick = 0;
        }
    }
}
```

2.1.7.4 RF data receive process

RF-2410U Demo Firmware RF data receive process responsible for detecting hardware IRQ interrupt. When interrupt happens, read the status of RF STATUS register. If the register status RX_DR indicates 1, read data from RX FIFO. Then write the data read to USB SEND FIFO buffer. The flow is as follows:

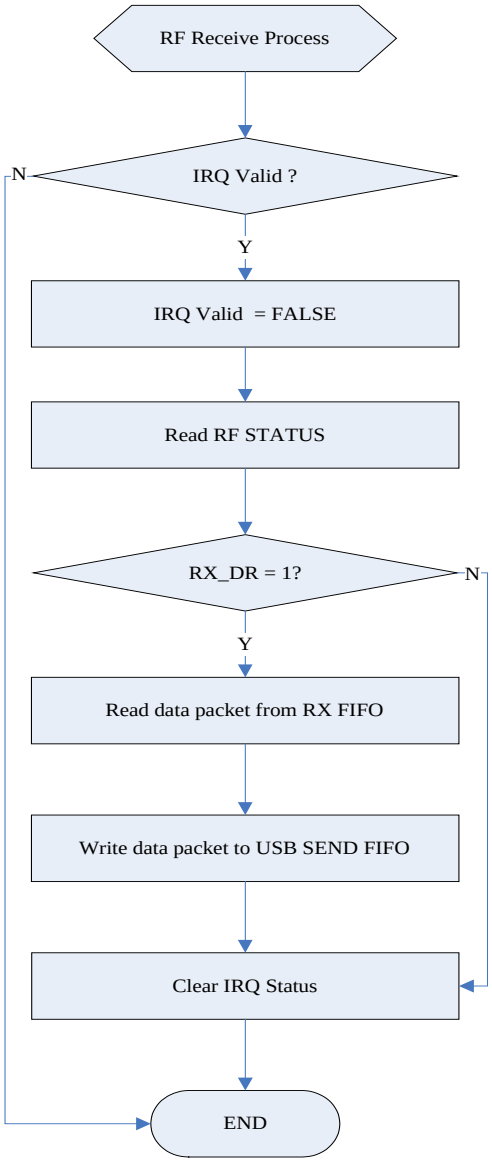

Figure 14 RF-2410U RF Receive Process Flow

RF-2410U Demo Firmware RF data receive process achieve by RF_Recv_Process() function. The sample codes are as follows:

```
/*******************************************************************************************
Function:          void RF_Recv_Process( void )
Parameter:
                   None
Return:
                   None
Description:
                   Read rf receive data out of the RF fifo, once the IRQ valid
                   then fill it to usb send fifo
*******************************************************************************************/
void RF_Recv_Process( void )
{
    UINT8 sta;
    UINT8 rlen;
    UINT8 *pPkt;

    if( g_rf.irqvalid )
```

```
    {
        g_rf.irqvalid = 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_rf_packet, sizeof(S_RF_PKT) );
            RF_FLUSH_RX();

            //Is a resend packet?
            if( g_rf_packet.sn_pkt != g_rf.sn_recv )
            {
                //records the packet sn
                g_rf.sn_recv = g_rf_packet.sn_pkt;

                //Limit the data length of received packet
                rlen = g_rf_packet.len;
                if( rlen > RF_PKT_LEN )
                {
                    rlen = RF_PKT_LEN;
                }

                //fill the data of received packet to usb send fifo
                pPkt = (UINT8 *)g_rf_packet.param;
                while( rlen-- )
                {
                    g_usb_fifo[g_usb.pos_w++] = *pPkt++;
                    if( g_usb.pos_w >= FIFO_LEN_USB )
                    {
                        g_usb.pos_w = 0;
                    }
                    g_usb.length++;
                }
            }
        }
        if( sta & STATUS_MAX_RT )           //Send fail?
        {
            RF_FLUSH_TX();                  //Flush the TX FIFO
        }

        RF_CLR_IRQ( sta );                  //Clear the IRQ flag
    }
}
```

Specifications: To improve RF-2410U Demo Firmware RF's receiving efficiency, we specifically put RF_Recv_Process() function call to IRQ interrupt service program. Please refer to "Figure 5 RF-2410U Demo Firmware Main Routine" for detail information.

2.2 RF-2410M Demo Firmware Software Architecture

2.2.1 Software features and main structure description

RF-2410M Demo Firmware finishes the following functions:

- Receive data from PC B by UART interface and save in RF FIFO
- Send data from UART FIFO to PC B by UART interface
- Receive data from RF-2410U by RF and save in UART FIFO
- Send data from RF FIFO to RF-2410U by RF

The main task routine of RF-2410M Demo Firmware divides to System Initialize, RF Receive Process, RF Send Process, UART Receive Process UART Send Process, etc. System Initialize includes C8051F330 initialization related configuration (port、system clock、timer、UART、SPI and interrupt hardware configuration) and BK2421 chip initialization configuration. RF Receive Process and RF Send Process achieve RF data transmission process; UART Receive Process and UART Send Process achieve UART data transmission process. Due to UART transmission timely require, we specifically put UART Receive Process and UART Send Process on UART interrupt service function.

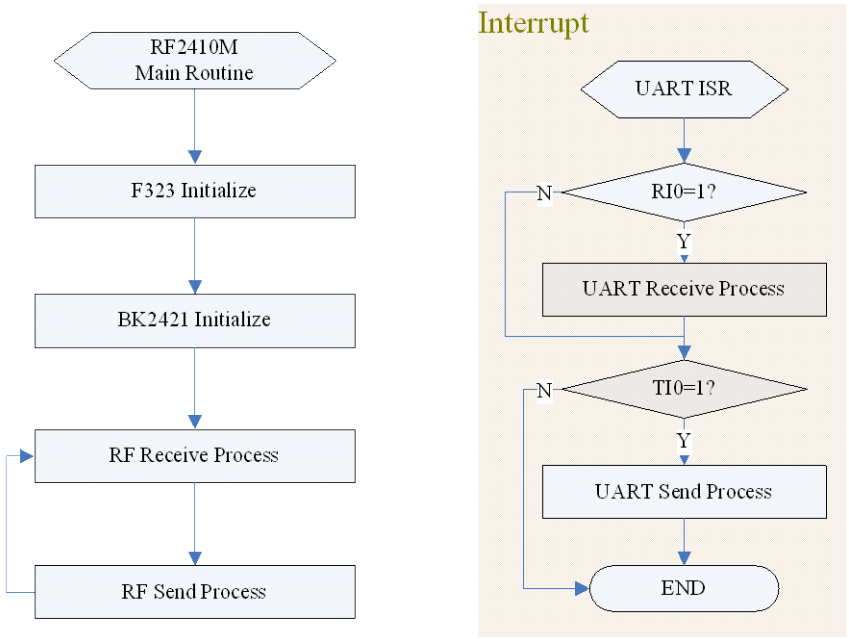

Figure 15 RF-2410M Demo Firmware Main Routine

2.2.2 BK2421 configuration and initialization

Please refer to RF-2410U Demo Firmware “BK2421 configuration and initialization” for RF-2410M Demo Firmware’s BK2421 chip configuration and initialization

2.2.3 RF data transmission process

RF-2410M Demo Firmware’s RF data transmission mechanism is the same as RF-2410U Demo Firmware. As for RF data packet format definition, please refer to RF-2410U Demo Firmware’s “RF packet format definition”. As for description of RF data packet sending and receiving process, please refer to RF-2410U Demo Firmware’s “RF data packet sending and receiving process”.

2.2.3.1 RF data send process

RF data send process constantly scanning and detecting RF FIFO's valid length. When meet any of the following conditions, send data:

- Valid length reaches a packet data length of RF
- Valid length is greater than o and send interval is more than 3ms

When sending data, MCU firstly need to read one packet data from RF FIFO and fill to the current sending data packet. Then call data packet sending function to send current packet. Read data from RF FIFO adopts circulating approach (Please refer to data cache planning and management). Because we need to perform on RF FIFO both in UART interrupt or RF send process, RF send process adopts the following mechanism for reading RF FIFO:

A. Close interrupt, read and count the current packet's length, then open the interrupt
B. Read data from RF FIFO to the current packet
C. Close interrupt, cut the read length from RF FIFO's current length, and then open the interrupt.

RF data send process shows below:

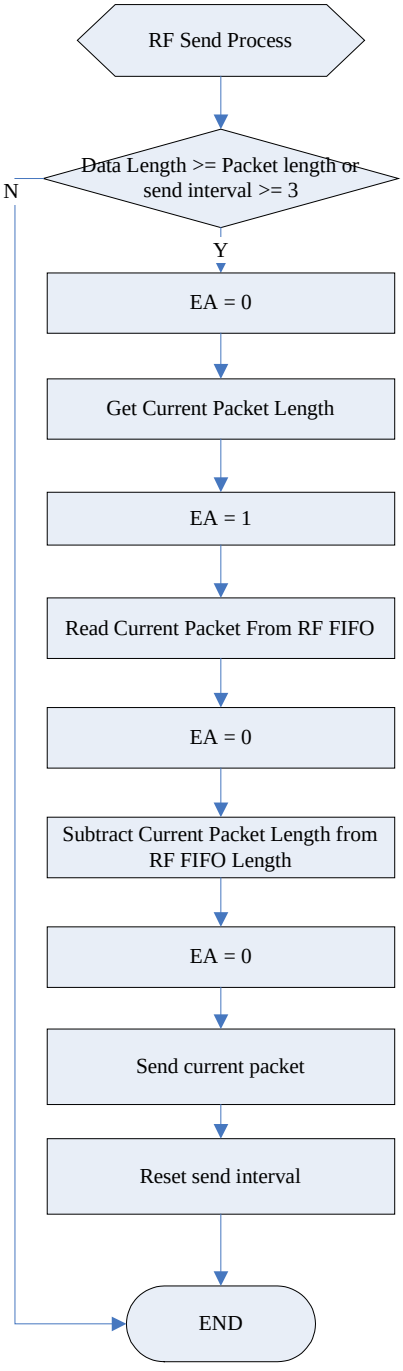

Figure 16 RF-2410M RF Send Process Flow

RF-2410M Demo firmware's RF data send process reached by RF_SendProcess()function. The codes are as follows:

```
/*****************************************************************************
Function:               void RF_SendProcess( void )
Parameter:
                        None
Return:
                        None
Description:
                        Check the RF FIFO and send out a packet according RF if
                        there is any valid data exist
*****************************************************************************/
void RF_SendProcess( void )
{
    UINT8 Len = 0;

    //RF FIFO Length reach the RF_PKT_LEN or send interval timeout?
    if( ( (g_RF.RcLen >= RF_PKT_LEN   ) || ( RF.snblankcnt >= 3 ) )   )
    {
        CLR_EA();                           //Disable Interrupt
        RF.R_Usalvewpos = g_RF.wpos;         //Save RF FIFO write position
        if( g_RF.RcLen > RFRCLEN )          //Limit the RF FIFO total length
        {
            g_RF.RcLen = RFRCLEN;
            g_RF.rpos = RF.R_Usalvewpos;
        }
        RF.U_Rsalvelen = g_RF.RcLen;        //Save the RF FIFO length
        SET_EA();                           //Reenable interrupt again

        //Copy data from RF FIFO and fill into RF send packet
        Len = CopyDataFromRFFIFO( &g_RFRcBuff );

        CLR_EA();                           //Disable interrupt
        g_RF.RcLen -= g_RFSenData.Len;      //Subtract the read length from RF FIFO total length
        SET_EA();                           //Enable interrupt

        RF_SendData( (UINT8 *)&g_RFSenData , Len ); //Send out current packet
        RF.snblankcnt = 0;                  //Clear send interval
    }
}


/*****************************************************************************
Function:               UINT8 CopyDataFromRFFIFO( UINT8 *pR )
Parameter:
                        *pR [IN]      UART receive buffer
Return:
                        None
Description:
                        copy data from RF FIFO to RF send packet
*****************************************************************************/
UINT8 CopyDataFromRFFIFO( UINT8 *pR )
{
```

```
    UINT8 i = 0;
    UINT8 Length;

    Length = RF.U_Rsalvelen;                    //Get the current packet length
    if( Length > RF_PKT_LEN )                   //Limit to RF_PKT_LEN
    {
        Length = RF_PKT_LEN;
    }

    for( i = 0; i <   Length   ; i++ )          //Copy data now, and forward move the read position
    {
        g_RFSenData.Param[i] = pR[g_RF.rpos++];
        if( g_RF.rpos >= RFRCLEN )              //Once the read position reach the end
        {                                       //reset it to start position
            g_RF.rpos = 0;
        }
    }

    //Fill the other parts of the packet
    g_RFSenData.sn    = RF.sn;
    g_RFSenData.sn_pkt    = RF.sn_pkt;
    g_RFSenData.Len = Length;
    RF.sn++;
    RF.sn_pkt++;

    return ( Length + 3 );
}
```

2.2.3.2 RF data receive process

RF data receive process responsible for constantly scanning and detecting IRQ's interrupt flag. When IRQ's interrupt flag is 1, read RX FIFO data and clear the flag. If the data length read is not o, write the data to UART FIFO and enforce to start UART sending process. Detailed RF data receive process shows below:

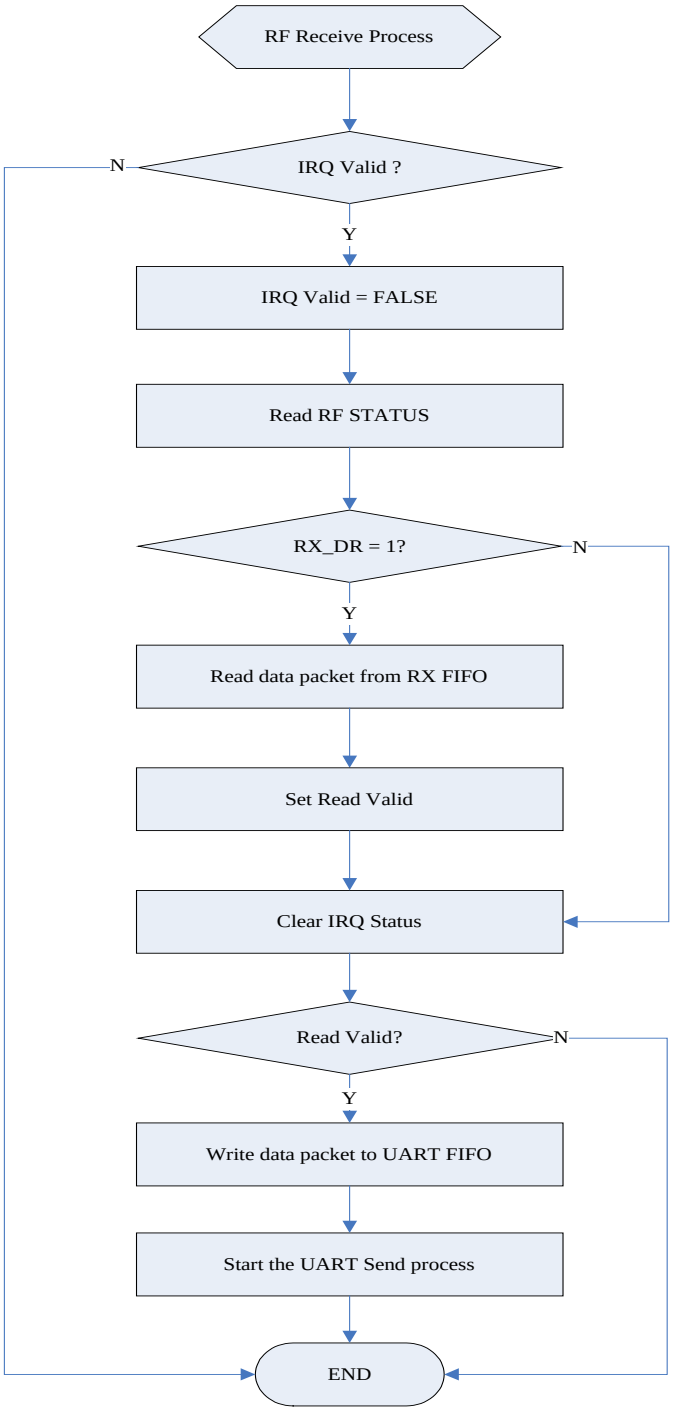

Figure 17 RF-2410M RF Receive Process Flow

RF-2410M Demo firmware RF_ReceiveProcess() function achieve RF data transmission process. The codes are as follows:

```
/**********************************************************************************
Function:               void RF_ReceiveProcess( void )
Parameter:
                        None
Return:
                        None
Description:
                        Check the IRQ periodically
                        Read out the received packet from RF RX FIFO
                        Write the received packet to UART FIFO
**********************************************************************************/
```

```
void RF_ReceiveProcess( 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 RF IRQ status;
        if( Rt & STATUS_RX_DR )                            //RX_DR is set?
        {
            //Read out the received packet from RF RX FIFO
            Len =   Read_RXPayload(( UINT8 *)&g_RFRecvData ,sizeof( RFDATAPKT ));
            FLUSH_LED2();

            isValid = TRUE;
        }

        //Add a clear MAX_RT error operation here
        if( Rt & STATUS_MAX_RT )
        {
            SPI_Write_Reg( FLUSH_TX, 0X00 );
        }

        SPI_Write_Reg( FLUSH_RX, 0X00 );                   //Flush RX FIFO
        SPI_Write_Reg( W_REGISTER | STATUS, Rt );          //Clear IRQ

        if( isValid )
        {
            CopyToUARTFifo( &g_RFRecvData , Len );         //Copy to UART FIFO

            CLR_EA();                                      //Disable interrupt
            g_uart.SnLen += RF.R_Usalvelen;                //Add the received packet length to UART FIFO
length
            if( g_uart.SnLen > UARTSNLEN )                 //Limit the UART FIFO length to UARTSNLEN
            {
                g_uart.SnLen = UARTSNLEN;
                g_uart.rpos    = g_uart.wpos;
            }
            SET_EA();                                      //Enable interrupt
        }

        if( !g_uart.SendInProgress )                       //Is UART send process stoped?
        {
            TI0 = 1;                                       //Force the UART send process to start
            g_uart.SendInProgress = TRUE;
        }
    }
}
```

```
/*******************************************************************************
Function:                 void CopyToUARTFifo( UINT8 *pRecv, UINT8 Length )
Parameter:
                          *pRecv   [IN]     RF received buffer
                          Length   [IN]     RF received length
Return:
                          None
Description:
                          Copy a packet data to UART FIFO
*******************************************************************************/
void CopyToUARTFifo( RFDATAPKT *pRecv , UINT8 Length)
{
    UINT8 i = 0;
    Length = pRecv->Len;                    //Get the copy length

    for( i = 0; i < Length; i++ )
    {
        //Copy a byte and move write position forward
        g_UARTSnBuff[g_uart.wpos++] = pRecv->Param[i];
        if( g_uart.wpos >= UARTSNLEN )   //Once the write position reach the end of FIFO
        {                                //Reset it to the start
            g_uart.wpos = 0;
        }
    }

    RF.R_Usalvelen = Length;
}
```

2.2.4 UART data transmission process

UART data transmission processes completely finished by UART interrupt routing. Please refer to “Figure 1 RF-2410M Demo Firmware main task flow”for its circuit.

UART_ISR() function is the procedure for UART interrupt service. It includes UART data receive process and UART data send process. The codes are as follows:

```
/*******************************************************************
Function:                 void UART_ISR( void )interrupt 4
Parameter:
                          None
Return:
                          None
Description:
                          UART interrupt process operation
*******************************************************************/
void UART_ISR( void ) interrupt 4
{
    if( RI0 )                                //Is Received a byte
    {
        RI0   = 0;                           //Clear RI0
        g_RFRcBuff[g_RF.wpos++] = SBUF0;     //Read a byte from UART and save to RF FIFO
                                             //Move forward RF FIFO write position
        if( g_RF.wpos >= RFRCLEN )           //Once the write position reached the end of RF FIFO
```

```
        {                                       //Reset it to the start
            g_RF.wpos = 0;
        }
        g_RF.RcLen++;                           //Increase the RF FIFO Length
    }
    if( TI0 )                                   //Is Last byte send finished?
    {
        TI0 = 0;                                //Clear TI0
        if( g_uart.SnLen )                      //Is any valid data exist in UART FIFO?
        {
            SBUF0 = g_UARTSnBuff[g_uart.rpos++];    //Featch a byte from UART FIFO and write to UART
                                                    //Move forward the UART FIFO read position
            if( g_uart.rpos >= UARTSNLEN )      //Once the read position reached the end of UART FIFO
            {                                   //Reset it to the start
                g_uart.rpos = 0;
            }
            g_uart.SnLen--;                     //Decrease the UART FIFO Length
        }
        else
        {
            g_uart.SendInProgress = FALSE;  //Mark the UART send operation stoped
        }
    }
}
```

2.3 USB To RF UART Demo software architecture (VB.net 2008)

2.3.1 Software features and main structure description

Software interface planning shows below:

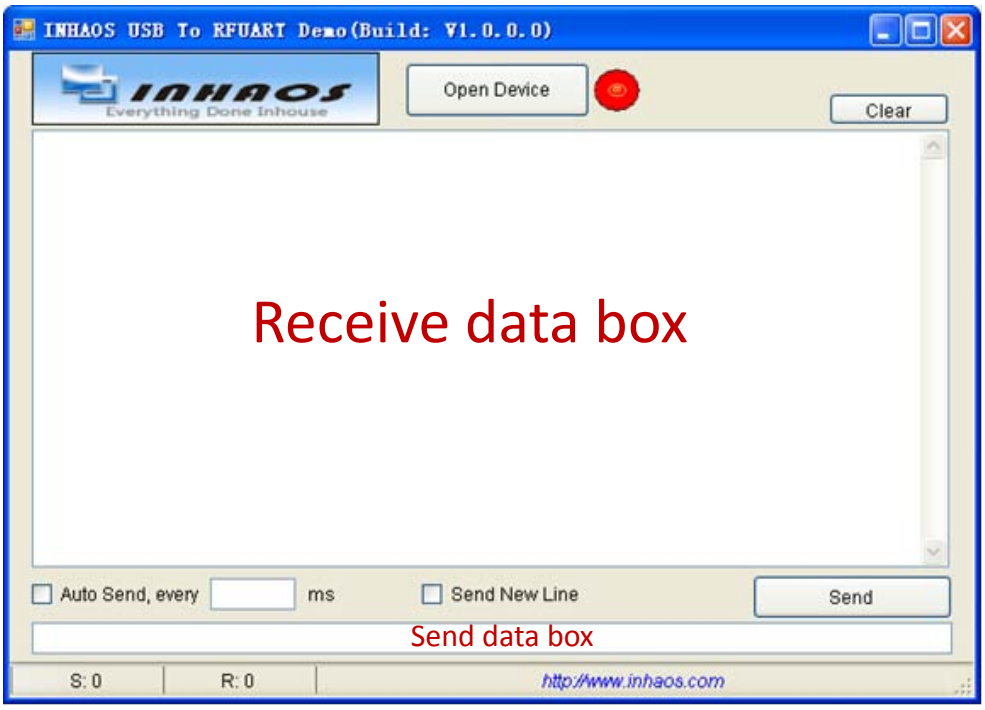

Figure 18 USB To RFUART Demo Form

USB To RFUART Demo achieves the following function

- Automatically search and open RF-2410U USBXpress device
- Write Send Data Box's string to RF-2410U USBXpress device
- Automatically read string from RF-2410U USBXpress device and show receiver list
- Have automatic and timing sending function
- Have sending line feed function
- Count the number of bytes for sending and receiving and then display

USB To RFUART Demo main architecture planning

- After starting the program, open a receive thread use for receiving RF-2410U device's data and display.
- Sending data by event trigger mechanism. When users press "Send"button, perform send process once.
- If users select "Auto Send" function, open a timer and set the time of users' specified send interval period. When timer happens overflow event, perform one send process.

2.3.2 USBXpress device access

Silicon Labs provides a Host API calling function integrated to achieve accessing to USBXpress device, The host API is provided in the form of a Windows Dynamic Link Library (DLL). The host interface DLL communicates with the USB device via the provided device driver and the operating system's USB stack. The following is a list of the host API functions available:

- SI_GetNumDevices() - Returns the number of devices connected
- SI_GetProductString() - Returns a descriptor for a device
- SI_Open() - Opens a device and returns a handle
- SI_Close() - Cancels pending IO and closes a device
- SI_Read() - Reads a block of data from a device
- SI_Write() - Writes a block of data to a device
- SI_FlushBuffers() - Flushes the TX and RX buffers for a device
- SI_SetTimeouts() - Sets read and write block timeouts
- SI_GetTimeouts () - Gets read and write block timeouts
- SI_CheckRXQueue() - Returns the number of bytes in a device's RX queue
- SI_DeviceIOControl() - Allows sending low-level commands to the device driver
- SI_GetDLLVersion() - Gets the version of the DLL currently in use
- SI_GetDriverVersion() - Gets the version of the USBXpress driver

2.3.2.1 Open the device

USB To RFUART Demo software achieve open operation for device by calling OpenDevice() function.

OpenDevice() function firstly call SI_GetNumDevices() function to gain the sum of USBXpress device. Then get each device's VID、PID compare with RF-2410U device's VID、PID until get the match one. Finally, call SI_Open() function perform open operation on the device. The sample codes are as follows：

```
Private Sub OpenDevice()
    Dim devIndex As Integer = SearchDeviceIndex()
    If devIndex >= 0 Then
```

```
            'Open device
            If SI_Open(devIndex, g_DeviceHandle) = SI_SUCCESS Then
                g_DeviceOpened = True
                btOpen.Text = "Close Device"
                lblOpen.ImageIndex = 1

                'Resume the receive thread
                evtRecvStart.Set()
            End If
        End If
    End Sub

    Private Function SearchDeviceIndex() As Integer
        Dim dwNumDevice As Integer = 0
        Dim byteVid(4) As Byte
        Dim bytePid(4) As Byte
        Dim callRes As Integer
        Dim strVid As String = ""
        Dim strPid As String = ""

        'Get the total number of usb devices current connected to PC
        callRes = SI_GetNumDevices(dwNumDevice)
        If callRes = SI_SUCCESS Then
            For i As Integer = 0 To dwNumDevice - 1
                'Read out the VID bytes
                SI_GetProductString(i, byteVid(0), SI_RETURN_VID)
                'Read out the PID bytes
                SI_GetProductString(i, bytePid(0), SI_RETURN_PID)

                'Convert the VID bytes to VID string
                strVid = System.Text.Encoding.Default.GetString(byteVid, 0, 4)
                'Convert the PID bytes to PID string
                strPid = System.Text.Encoding.Default.GetString(bytePid, 0, 4)

                'Compare VID\PID strings with us specific VID\PID
                If strVid.ToUpper = DEVICE_VID AndAlso strPid.ToUpper = DEVICE_PID Then
                    Return i
                End If
            Next
        End If

        Return -1
    End Function
```

2.3.2.2 Write data to the device

Call SI_Write() function directly write fixed length data to USBXpress device.

2.3.2.3 Read data from the device

Firstly call SI_CheckRXQueue() function to check the current receive buffer's data length. And then call SI_Read() function read fixed length data from USBXpress device.

2.3.2.4 Close the device

When completing writing and reading operation, it needs to close all USBXpress device which have been open before by calling SI_Close function.

2.3.3 "Send"Click Event

Add "Send" button's Click event response btnSend_Click () function, in btnSend_Click () function implement sending text data to USBXpress device. The sample codes are as follows:

```
Private Sub btSend_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btSend.Click
    Dim strSend As String = tbSend.Text

    'Is a new line characters need to be added?
    If chkSendNewLine.Checked = True Then
        strSend &= vbNewLine
    End If

    'Convert the send string to bytes array
    Dim arraySend() As Byte = System.Text.Encoding.Default.GetBytes(strSend)
    Dim iSendLength As Integer = arraySend.Length
    Dim iBytesWrote As Integer = 0
    Dim iWLen As Integer = 0

    'Loop to send all the byte out
    While (iBytesWrote < iSendLength)
        'Write to device
        iWLen = WriteToDevice(arraySend, iSendLength)
        iBytesWrote += iWLen

        'Update informations for sending
        iAccBytesSent += iWLen
        tsslSend.Text = "S: " & iAccBytesSent.ToString
    End While
End Sub
```

2.3.4 Data Receive Thread

Due to USBXpress API interface function haven't reached any data accepted event, so we need to open a separate receiver thread ReceiveProc to receive data from USBXpress device.

ReceiveProc thread performs the following tasks:

- Call SI_CheckRXQueue() function to check the current receiver's length
- If the current length is not 0, call SI_Read() function to read all receive data
- Convert all receive data to text and show in receiver box

ReceiveProc thread implementation codes shows below:

```
Private Sub ReceiveProc()
    Dim iReadLength As Integer
    Dim iBytesToRead As Integer
    Dim iSiStatus As Integer
    While (g_ExitFlag <> True)
```

```
            evtRecvStart.WaitOne()    'Wait for Receive start signal
            If g_ExitFlag = True Then Exit While

            If g_DeviceOpened = True Then
                iBytesToRead = 0
                'Check the exist bytes in receive fifo
                SI_CheckRXQueue(g_DeviceHandle, iBytesToRead, iSiStatus)

                If iBytesToRead > 0 Then
                    iBytesToRead = IIf(iBytesToRead > RECV_BUFF_LEN, RECV_BUFF_LEN, iBytesToRead)
                    iReadLength = 0
                    SI_SetTimeouts(500, 500)
                    'Read out the data
                    If SI_Read(g_DeviceHandle, arrayRecvBuffer(0), iBytesToRead, iReadLength, 0) = SI_SUCCESS
Then
                        If iReadLength > 0 Then
                            'Display data to tbRecv control
                            DisplayRecvText(arrayRecvBuffer, iReadLength)
                        End If
                    End If
                Else
                    Threading.Thread.Sleep(1)
                End If

            End If
        End While
    End Sub
```

3 System Performance Testing

3.1 Test environment preparation

Test spaces require the following condition:

- Make sure there is no big electromagnetic interference around
- The distance between RF-2410U and RF-2410M is 5m and without any block.

Hardware preparation:

- PC: PC A、PC B two sets in total
- RF-2410U one
- RF-2410M ne
- UC-2000 one
- Power Supply ne set

Please refer to “Figure 1 system hardware architecture diagram” for hardware connection.

Software environment:

- PC A port: Operate USB o RFUART Demo.exe and set according to the following figure

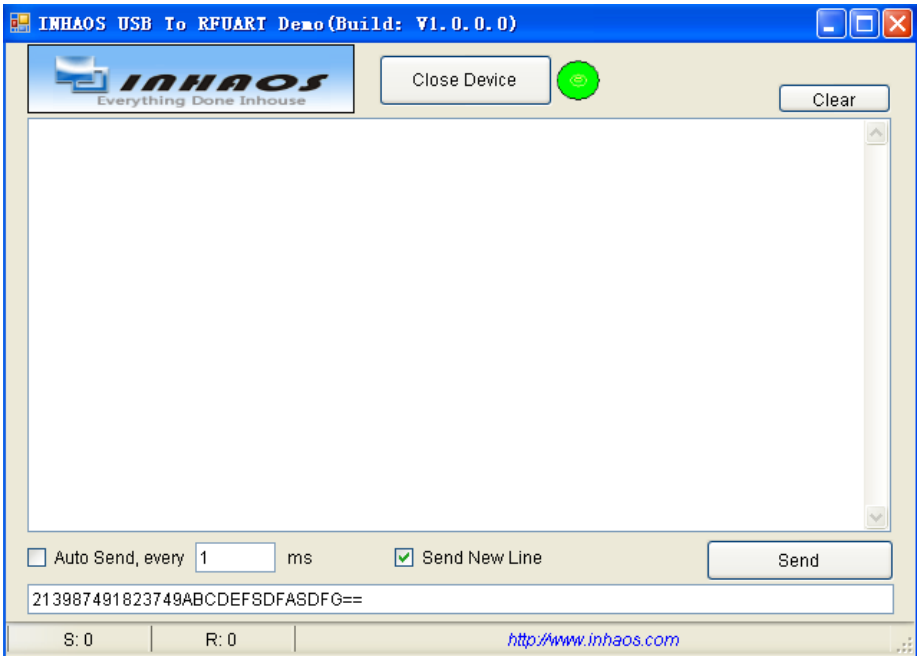

Figure 19 USB To UART Demo Setting

- RF-2410U port： Perform RF-2410U Demo Firmware；
- RF-2410M port: Perform RF-2410M Demo Firmware；
- PC B port: Perform SSCOM32e.exe and select CommNum for UC-2000 communication port. Communication setting shows below

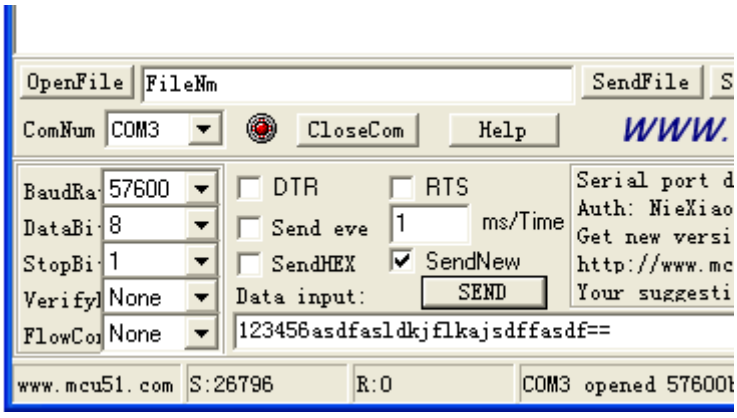

Figure 20 SSCOM32 Setting

3.2 Test methods and procedures

To intuitively come to data transmission performance between PC A and PC B, here adopts send terminal continuously send data more than 100000 bytes. Then calculates BER by receive end's bytes.

3.2.1 PC A transfer data to PC B

3.2.1.1 PC A transfer data to PC B test procedure:

- USB To RFUART Demo interface: Click "Open Device"to open RF-2410U communication port; Click "Clear"button to clear the value of sending and receiving.
- SSCOM32 interface: Click "Clear" button, clear the value of sending and receiving.
- USB To RFUART Demo interface, tick "Send Every", start dispatching regularly. When dispatch bytes are more than 100,000, cancel the tick.

Test results:

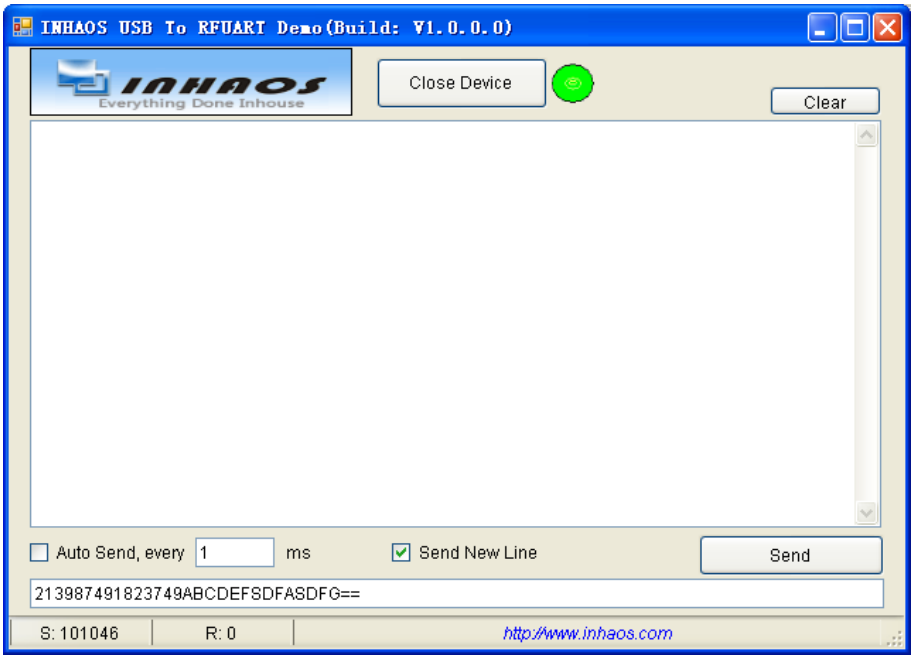

Figure 21 PC A to B Test Result _ USB To RFUART Demo

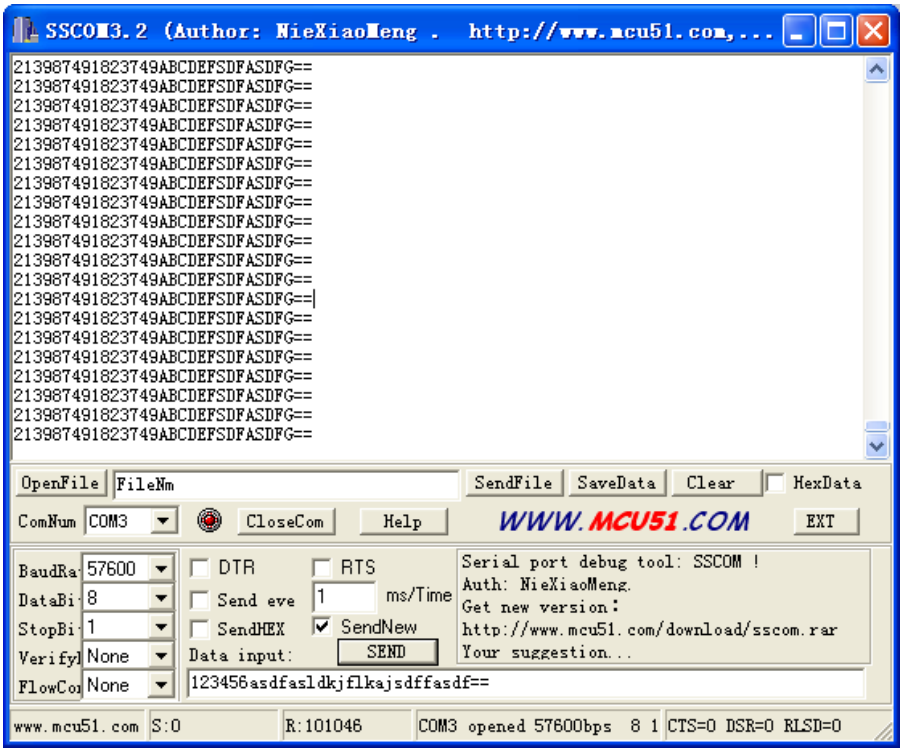

Figure 22 PC A to B Test Result _ SSCOM32

As the two screenshots show above, USB To RFUART Dem send the number of bytes is 101046 totally the same with SSCOM32 receive number bytes of 101046 which means the BER is 0%.

3.2.1.2 PC B transfer data to PC A

- USB To RFUART Demo interface: click “Open Device” to open RF-2410U communication port; click “Clear”button to clear the value of sending and receiving.
- SSCOM32 interface, click “Clear” button, clears the value of sending and receiving.
- SSCOM32 interface, tick “Send Every”, start dispatching regularly. When dispatch bytes are more than 100,000, cancel the tick.

Test results:

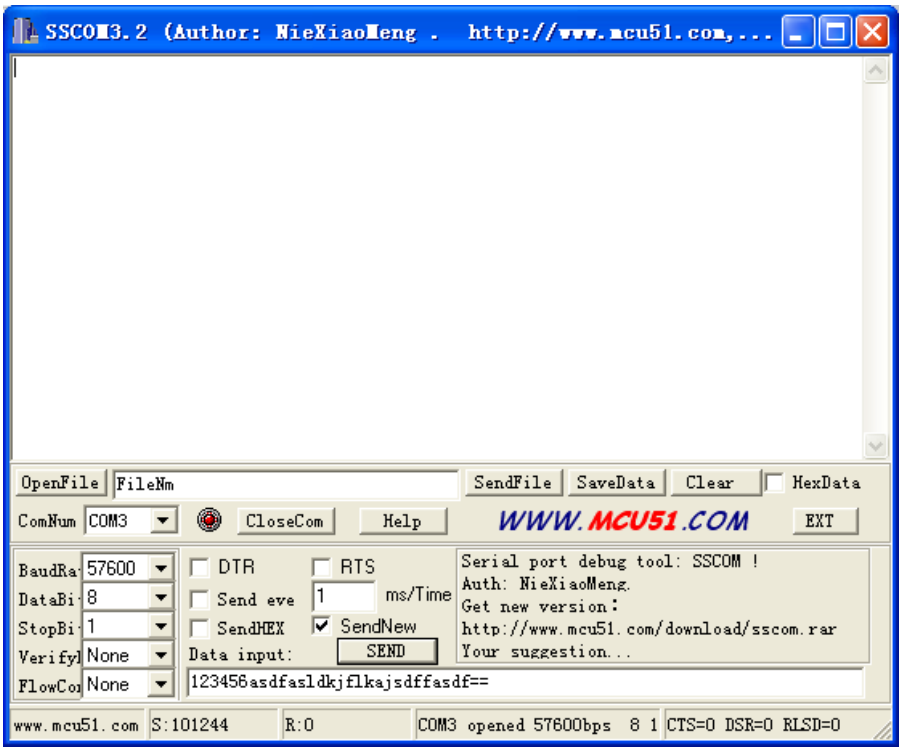

Figure 23 PC B to A Test Result _ SSCOM32

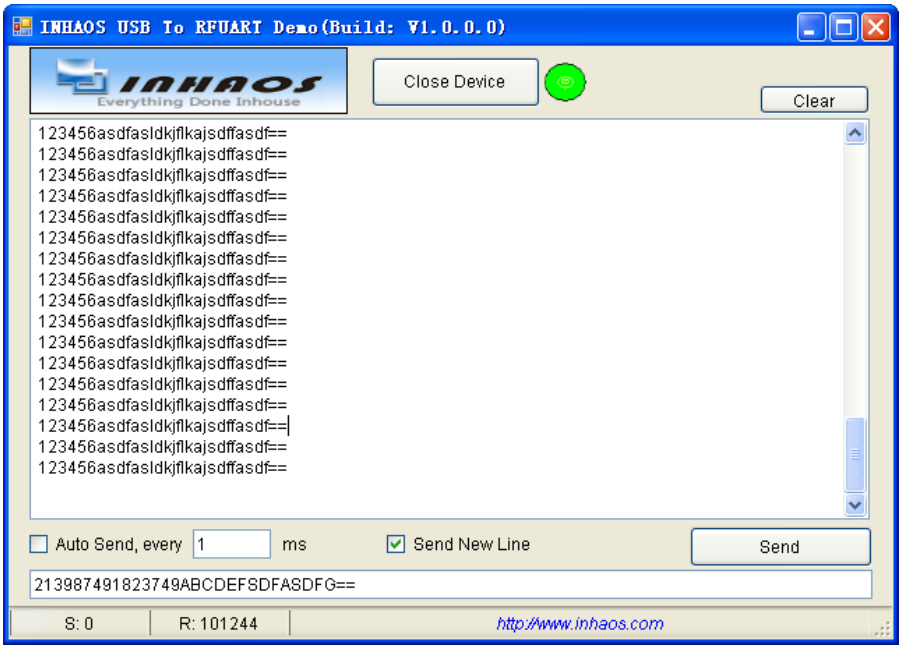

Figure 24 PC B to A Test Result _ USB To RFUART Demo

As the two screenshots show above, SSCOM32 send the number of bytes 101244 and USB To RFUART Demo receive the number of bytes is totally the same. So the BER is 0%.

3.3 Software performance summary

As the test above show, this sample system can be quite stably transfer data between PCA and PCB one way.

Specifications:

◆We cannot ensure all users come to the exactly results due to RF have rigorous requirement for the surrounding environment.

◆All the tests above are one way data transmission. So we cannot promise any reliability for data transmission of double-side.

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