# BC66F8x0 RF Transceiver Demo Code Description

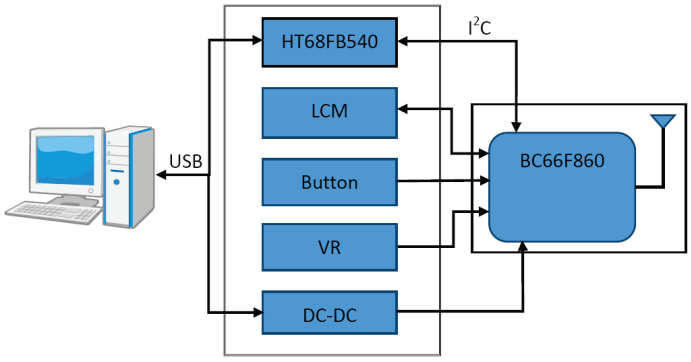
**文件编码：AN0359S**

## 概述

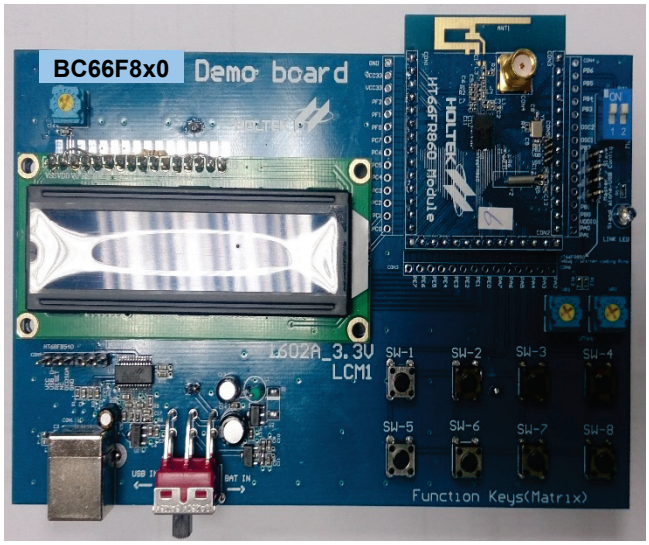
BC66F8x0 是一款集成 MCU + 2.4GHz RF Transceiver 的 SIP，通过 SPI 传输方式在 RF Transceiver 与主控 MCU 之间进行控制与数据的交换。由于是 RF Transceiver，所以可进行无线的双向传输，可适用于多种家电、计算机外设、玩具等的无线摇控。

本范例介绍利用 BC66F8x0 demo board 及 demo code 的 API 以便用户修改 demo code 主程序进行应用方案的开发。

BC66F8x0 Demo board 主要分成三部分：
1.PC 端 GUI 应用程序：通过 USB 与 Demo board 进行通讯。
2.Main Board：包含 USB 芯片、LCM displa、按键及电源处理。
3.BC66F8x0 模块：BC66F860 芯片及天线。




**BC66F8x0 Demo Board 方框图**




## 功能描述

本范例功能分成 USB Mode 及 Stand Alone Mode。

### USB Mode

通过 PC 的 USB 下 command 给 BC66F8x0 做 RF 的调试及验证。

- Init : 对 RF 做基本的设置
- Continuous PTX : 连续发射
- Continuous PRX : 连续接收
- Carrier Signal : Carrier 连续发射
- Standby-I : RF 进入 standby I mode
- Standby-II : RF 进入 standby II mode
- Power-Down : RF 进入 power down mode
- BER Test : Rx Sensitivity test.(须搭配可传送 PN9 code 的 RF 信号产生器)
- RF Transfer : RF Transfer test(须 2 套 demo board)
- 2.2. Stand Alone Mode
- 可切换成 Master 和 Slave 两种
- 两片板子通过 RF 传输互显示按键及电位计电压状态

### Stand Alone Mode

- 可切换成 Master 和 Slave 两种
- 两片板子通过 RF 传输互显示按键及电位计电压状态


Note：当两片 Demo board 互传数据时。

1. 会自动配对。
2. 底层 F/W 已经 implement 自动跳频机制(FHSS)可避免干扰。

# 档案说明

**Project 名称**

BC66F8x0 Demo.pjtx。

**文件列表**

| 文件名 | 说明 |
|---|---|
| main.c | demo code 的主程序 |
| radio.c | Tx/Rx 及频率 hopping 处理程序 |
| hal_bc.c | RF control 的相关子程序 |
| hal_button.c | 读取 Button Status |
| hal_adc.c | 读取 ADC 的值 |
| hal_lcm.c | LCM display 处理程序 |
| hal_i2c.c | $I^2C$ Slave 处理程序 |
| hal_mem.asm | MCU memory 处理程序 |
| hal_spi.asm | RF SPI 处理程序 |
| table.asm | RF 初始表及 LCM 显示字符串 Table 表 |
| vector.asm | MCU 中断向量表 |

**HT-IDE3000 编译设定注意事项**

- 编译选择 Holtek C Compiler V3/Assembler
- 配置选项
  - VDDIO/GPIO selection : VDDIO
  - SPI : Enable
  - SPI CSEN : Enable
- 编译参数的选项要有以下选项
  - 汇编语言区分大小写
  - 将未初始化的全局变量，全局/局部静态变量默认值设为 0

# 参数说明

### main.c

| 参数类型 | 参数名称 | 说明 |
|---|---|---|
| uint8 | radio_transm_dat | RF 传输 buffer |
| uint8 | radio_transm_period_counter | RF 传输间隔时间(ms) |
| uint8 | radio_pairing_temp_addr | master/Slave 配对的 address |
| uint16 | radio_link_timeout | RF data loss 的时间计时(ms) |

### radio.c

| 参数类型 | 参数名称 | 说明 |
|---|---|---|
| uint8 | radio_status_var | radio 模式(IDLE/TX_ACTIVE/RX_ACTIVE) |
| uint8 | radio_address | radio TX/RX 的 address |
| radio_operate_t | radio_sts | radio 的状态标志位<br>typedef union<br>{<br>uint8 value;<br>struct<br>{<br>uint8 max_rt : 1; /*中断标志位 */<br>uint8 tx_ds : 1; /*中断标志位 */<br>uint8 rx_dr : 1; /*中断标志位 */<br>uint8 sync_on : 1; /*channel 同步标志位 */<br>uint8 tx_active : 1; /*PTX 工作标志位 */<br>uint8 tx_success : 1; /*PRX 工作标志位 */<br>uint8 pairing_enable : 1;<br>uint8 rx_power_high : 1;<br>}bits;<br>}radio_operate_t; |
| uint8 | Payloadwidth | 读出 RX FIFO 的数据长度 |
| uint8 | TxPayloadBuffer | 要写入 TX FIFO 的 buffer |
| uint8 | RxPayloadBuffer | 读出 RX FIFO 的 buffer |
| uint8 | radio_sync_period | channel 同步时间计时 |
| uint8 | radio_timeout_counter | radio TX/RX timer out 时间 |

### hal_i2c.c

| 参数类型 | 参数名称 | 说明 |
|---|---|---|
| hal_i2c_operate_t | i2c_operate_flag | i2c 工作的相关标志位<br>typedef union<br>{<br>uint8 value;<br>struct<br>{<br>uint8 trans_work : 1; /*I2C 工作标志位 */<br>const uint8 : 2;<br>uint8 mode : 1; /*I2C R/W 标志位 */<br>uint8 first_byte : 1;<br>uint8 slave_transm : 1; /*Slave 传送数据标志位 */<br>uint8 slave_busy : 1; /*Slave 忙录标志位 */<br>uint8 addr_match : 1; /*Slave address 标志位 */<br>} bits;<br>} hal_i2c_operate_t; |
| uint8 | i2c_buffer_index | i2c 读写 buffer 的 pointer |
| uint8 | i2cTxBuffer | i2c 写入的 buffer |
| uint8 | i2cRxBuffer | i2c 读出的 buffer |

# 函数列表

## main.c

| 函数型态 | 函数名称 | 说明 |
|---|---|---|
| void | timer0_init(void) | 设定 timer 0 为 1ms 定时器 |
| void | DelayXmSec(uint8 xs) | 250us 单位的延迟 |

## radio.c

| 函数型态 | 函数名称 | 说明 |
|---|---|---|
| void | radio_param_init(void) | 初始化 radio 相关参数 |
| void | radio_timer_period_init(void) | 设定 timer 2 为 channel 间隔时间定时器 |
| void | radio_rx_start(void) | 启动 radio 成 Rx 模式 |
| void | radio_rx_fifo_read(void) | 读取 RF 的 Rx FIFO |
| uint8 | radio_tx_data(uint8 len) | 启动 radio 成 Tx 模式，len = payload length |
| void | radio_new_tx_data(uint8 chmd) | 启动 RF 开始传送一笔数据 |
| void | reload_tries_pr_channel_counter(void) | 设定在一个 channel 上要 retry 的次数 |
| void | Radio_ISR(void) | RF 的中断处理程序 |
| void | radio_timer_period_ISR(void) | RF timer period 中断处理程序 |

## hal_bc.c

| 函数型态 | 函数名称 | 说明 |
|---|---|---|
| void | hal_bc_SPI_IRQ_Init(void) | 设定 MCU 的 SPI 连接 RF |
| void | hal_bc_Init(void) | RF 基本的初始化 |
| void | hal_bc_Bank1Init(void) | RF bank 1 的初始化 |
| void | hal_bc_Bank1Register(uint8 idx) | 写入 RF bank 1 的寄存器 |
| void | hal_bc_ActivateFrature(void) | 启动 RF dynamic payload length/Payload with ACK 等功能 |
| void | hal_bc_OperationMode(uint8 mode) | 设定 RF 的 TX/RX 模式 |
| void | hal_bc_PowerMode(uint8 mode) | 设定 RF power up/power down |
| void | hal_bc_OutputPower(uint8 power) | 设定 RF 的输出功率 |
| void | hal_bc_DataRate(uint8 dr) | 设定 RF 传输的速率(250K/1M/2M) |
| void | hal_bc_RFChannel(uint8 ch) | 设定 RF 的工作频率 |
| void | hal_bc_SetAddress(uint8 pipe,uint8 addr[]) | 设定 TX/RX 的 address |
| void | hal_bc_auto_retr(uint8 retr,uint8 delay) | 设定 TX Auto Retransmission Delay 及 counter |
| void | hal_bc_RegsBankSw(uint8 b) | 读写 bank0/bank1 寄存器的切换 |
| void | hal_bc_PulseCE(void) | RF 的 CE PIN 产生约 16us 的高脉波信号 |

### hal_spi.asm

| 函数型态 | 函数名称 | 说明 |
|---|---|---|
| uint8 | Read_Register(uint8 addr) | 读取 RF bank 0 的寄存器 |
| void | Write_Register(uint8 addr,uint8 data) | 写入 RF bank 0 的寄存器 |
| uint8 | Read_Status(void) | 读取状态寄存器 |
| uint8 | Reuse_Payload(void) | reuse TX FIFO |
| uint8 | Flush_TxFIFO(void) | 清除 TX FIFO |
| uint8 | Flush_RxFIFO(void) | 清除 RX FIFO |
| uint8 | GetRxPayloadWidth(void) | 读取 RX payload 的长度 |
| void | Write_Activate(uint8 value) | 写入 RF Activate command |
| void | Write_ACK_Payload(uint8 width) | 将 payload 数据写入 TX FIFO (W_TX_PAYLOAD command) |
| void | Write_NACK_Payload(uint8 width) | 将 payload 数据写入 TX FIFO (W_TX_PAYLOAD_NOACK command) |
| void | Write_RxACK_Payload (uint8 pipe, uint8 width) | 将 payload 数据写入 TX FIFO (W_ACK_PAYLOAD command) |
| void | Read_Rx_Payload(uint8 width) | 读取 RF FIFO 数据 |
| void | WriteBank1Register(uint8 idx) | 写入 RF bank 1 寄存器 |

### hal_i2c.c

| 函数型态 | 函数名称 | 说明 |
|---|---|---|
| void | hal_I2C_Init(void) | 初始化 i2c 功能 |
| void | MultiFunction4_ISR(void) | i2c 中断处理程序 |

# 函数使用说明

### RF Power-On 初始化

```
_clkoen = TRUE;                    /*打开 16MHZ 的 clock 到 RF IC*/
hal_bc_SPI_IRQ_Initial();          /*初始化 RF IC 的 SPI*/
hal_bc_Bank1Initial();             /*初始化 RF IC bank1 寄存器*/
radio_param_initial();             /*初始化 radion 所有参数*/
radio_timer_period_initial();      /*初始化每个 channel 间隔时间计时.*/
```

### hal_bc_Init()  /* 初始化 RF bank 0 寄存器 */

a. hal_bc_Init()函数会读取_BC2422ConfigTable 表的数据对 RF IC 做初始化。

b. 函数会清空 RF IC 内的 TX/RX FIFO 及所有中断的状态。

c. Table 表的格式 high byte = register address, low byte = initial data。

_BC2422ConfigTable：

```
DW CONFIG_REGS SHL 8) + 0001110B              ;EN_CRC=1,CRCO=2,PWR_UP=1,PTX
DW      (EN_AA_REGS SHL 8) + 00000000B        ;Enable 『Auto Acknowledgment』
DW      (EN_RXADDR_REGS SHL 8)+00000001B      ;设定 RX active register
DW SETUP_AW_REGS SHL 8) + 03H                 ;Address Widths = 5 byte
DW SETUP_RETR_REGS SHL 8)+00000000B           ;ReTransmit Delay=2336us
                                              ;ReTransmit Count=4
```

### void hal_bc_OperationMode(uint8 mode)  /* 设定 RF 的工作模式*/

```
hal_bc_OperationMode(HAL_PTX);          /**< Primary TX operation */
hal_bc_OperationMode(HAL_PRX);          /**< Primary RX operation */
```

### void hal_bc_PowerMode(uint8 mode)

```
hal_bc_PowerMode(HAL_PWR_DOWN);          /**< Device power-down */
hal_bc_PowerMode(HAL_PWR_UP);            /**< Device power-up */
```

＊在做 power-up/power-down 时，请注意_clkoen control 的设定。

### void hal_bc_OutputPower(uint8 power) /* 设定 RF 输出功率 */

power level0~3 共有 4 阶。

```
hal_bc_OutputPower(0);                   /**< Output power set to min */
hal_bc_OutputPower(3);                   /**< Output power set to max */
```

### void hal_bc_DataRate(uint8 dr) /* 设定 RF 传输速率 */

```
hal_bc_DataRate(HAL_250KBPS);            /**< Datarate set to 250 kbps*/
hal_bc_DataRate(HAL_1MBPS);              /**< Datarate set to 1 Mbps   */
hal_bc_DataRate(HAL_2MBPS);              /**< Datarate set to 2 Mbps   */
```

### void hal_bc_RFChannel(uint8 ch) /* 设定 RF 工作频率 */

工作频率 ch = 2400MHZ~2527MHZ(0~127)。

### void hal_bc_SetAddress(uint8 pipe, uint8 addr[]) /* 设定 RF 封包传输的地址 */

a. TX/RX 的 address 长度可以是 3~5byte，由寄存器 SETUP_AW (Setup of Address Widths, address=0x03) 设定。

b.TX/RX0/RX1 address 长度可以是 3~5byte，而 RX2~5 只能设定 1byte，byte1~4 会共享 RX1 的。

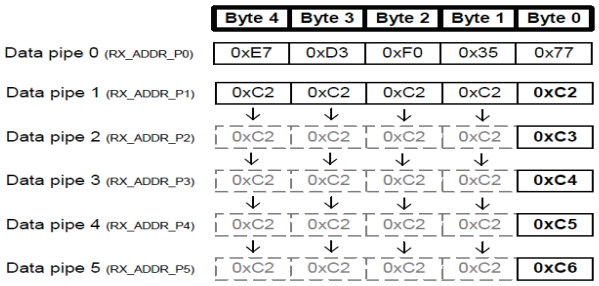



*Figure 13. Addressing data pipes 0-5*

```
Address = 0x8295E5DC6E;
radio_address[0] = 0x6E;
radio_address[1] = 0xDC;
radio_address[2] = 0xE5;
radio_address[3] = 0x95;
radio_address[4] = 0x82;
hal_bc_SetAddress(HAL_TX_ADDR,radio_address);      /**< Refer to TX address*/
hal_bc_SetAddress(HAL_RX_PIPE0,radio_address);/**< Refer to RX0 address*/
```

### uint8 radio_tx_data(uint8 len) /* RF 进入传输模式(PTX) */

a. 首先将要传送的数据搬到 TxPayloadBuffer。在调用 radio_tx_data()的子程序。

b. 调用 radio_tx_data()后 radio_status_var = TX_ACTIVE 模式。

c. 当 radio_status_var 再次等于 IDLE 模式。表示已经停止传送数据，这时可以检查 radio_sts.bits.tx_success 标志位，0：表示传送失败、1：表示传送成功。

```
/*将数据 copy 到 TxPayloadBuffer */
memorycopy((TxPayloadBuffer,radio_transm_data,4);
radio_tx_data(4);                          /*调用发射程序，长度=4 byte*/
while(radio_status_var != RADIO_IDLE);     /*等待数据传送完毕*/
if(radio_sts.bits.tx_success)              /*判断传送是否成功*/
{传送成功}
Else
{传送失败}
```

### RF 进入接收模式(PRX)

a. 首先调用 radio_rx_start()进入接收模式。

b. 再来调用 radio_rx_fifo_read()检查是否有收到数据，如果有收到数据会将数据放在 RxPayloadBuffer[]中。

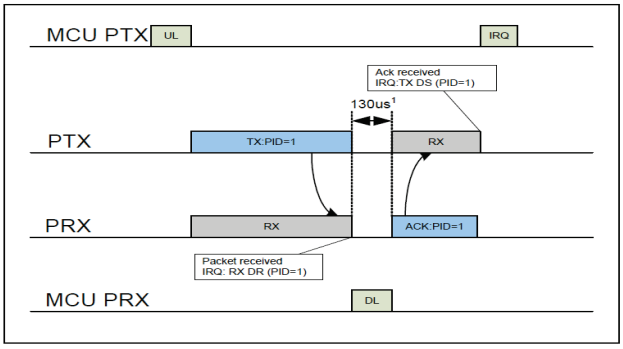
```
radio_rx_start();                          /** 启动 RX 模式 */
radio_rx_fifo_read();                      /** 检查 RX FIFO */
if(payloadwidth)                           /* 检查是否有数据 */
{
        memorycopy((uint8 *)radio_transm_data,(uint8 *)RxPayloadBuffer,payloadwidth);
                                           /*将数据 copy 出来 */
…
}
```

### void Write_ACK_Payload(uint8 width) /* 写入一笔数据到 RF 的 FIFO */

a. 这个函数会随着寄存器 EN_AA(Enable Auto Acknowledgment,0x01)的设定而定。EN_AA = 1 则会接收 ACK 信号，EN_AA = 0 是不会接收 ACK 信号。

b. 调用此函数时需先将发射数据放于 TxPayloadBuffer 中。

```
/*将数据 copy 到 TxPayloadBuffer */
memorycopy((TxPayloadBuffer,radio_transm_data,8);
Write_ACK_Payload(8);                      /*将数据写入 TX FIFO */
```




**Auto Acknowledgment Enable Cycles**

**void Write_NACK_Payload(uint8 width) /* 写入一笔数据到 RF 的 FIFO */**

a. 这个函数必须启动 Feature Register(0x1D)的 EN_DYN_ACK 时才有用。

b. 这个函数会不会随着寄存器 EN_AA(Enable Auto Acknowledgment,0x01)的设定而定。一律不会接收 ACK 信号。

c. 调用此函数时需先将发射数据放于 TxPayloadBuffer 中。

```
/*将数据 copy 到 TxPayloadBuffer */
memorycopy((TxPayloadBuffer,radio_transm_data,8);
Write_NACK_Payload(8);                          /*将数据写入 TX FIFO */
```

**void Write_RxACK_Payload(uint8 pipe, uint8 width) / *写入一笔数据到 RF 的 FIFO */**

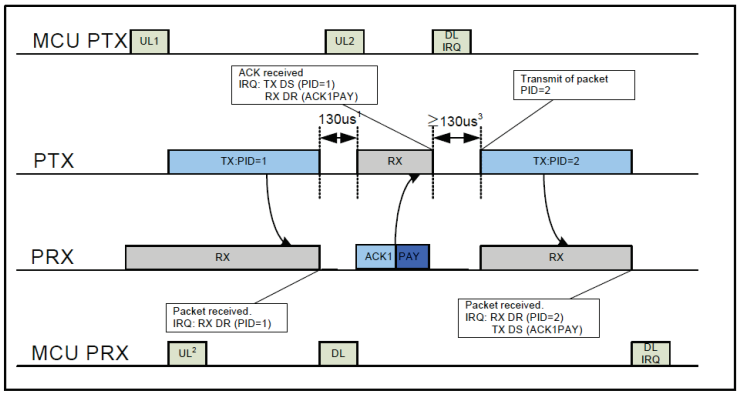
a. 此函数属于 PRX 模式在 ACK 时一起将数据传送给 TX。

b. 这个函数必须启动 Feature Register(0x1D)的 EN_ACK_PAY 时才有用。

c. 参数中的 pipe 所指定的是 6 组接收 address 中的其中一组。

d. 调用此函数时需先将发射数据放于 TxPayloadBuffer 中。

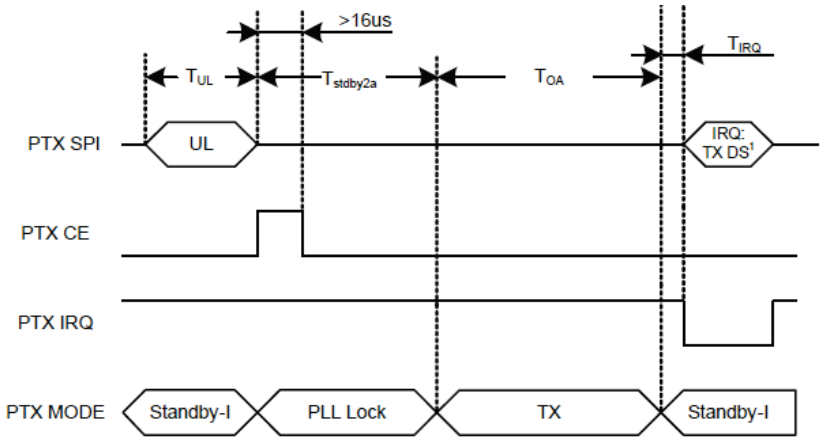
```
/*将数据 copy 到 TxPayloadBuffer */
memorycopy((TxPayloadBuffer,radio_transm_data,8);
Write_RXACK_Payload(8);                         /*将数据写入 TX FIFO */
```




在 PRX 中的 PAY 既是用 Write_RXACK_Payload 所写入的数据。

# 时序图

**Transmitting One Packet with NO_ACK On**




**Transmitting One Packet with ACK On**