# Content

# BC5602 HT32 Standard Demo Code User Guide

## Introduction

Holtek releases a new 2.4GHz Tx/Rx RF device, the BC5602. The BM5602-60-1 is a high-performance 2.4GHz RF GFSK transceiver module whose development is based on the 2.4GHz BC5602 device. The module supports bidirectional data transmission and can be widely used in 2.4GHz wireless applications, such as various switch remote controls, office automation and smart home wireless control applications. The device has integrated a high power amplifier, a frequency synthesizer and digital demodulation functions, which will result in much simplified external circuitry. The RF characteristics comply with ETSI/FCC specifications.

The BC5602 has an operating voltage ranging from 1.9V to 3.6V and a programmable transmitting power ranging from -10dBm to +6dBm. Its receiving sensitivity can be up to -98dBm at a data rate of 125Kbps. Its receiving current is 17mA at a data rate of 250/500Kbps. The device also supports a Low Sleep current of 0.5μA, a Middle Sleep Mode for crystal fast wake-up and an ATR (Auto-Transmit-Receive) function.

This user guide will introduce how to use the M0+ system development board, BCE-GENTrx32-001, together with the BM5602-60-1 module and a variety of RF receiving/transmitting Demo code to operate various functions of the BC5602, allowing users to understand its individual functions and have an overall system architecture. Through this introduction, users can develop RF wireless products for suitable application situations. The user guide will first introduce how to setup the development platform.

For more BC5602 technical information consult the Holtek website: https://www.holtek.com/productdetail/-/vg/BC5602.


**Figure 1. BM5602-60-1 Module**

# Applications

Various switch remote controls, office automation and smart home wireless control applications.

# Development Platform Description

This development platform uses the system development board, BCE-GENTrx32-001, which uses the Holtek 32-bit Arm® Cortex®-M0+ microcontroller HT32F52352 as the master controller. It also includes the BC5602 module board, BM5602-60-1. The entire platform architecture is shown below.

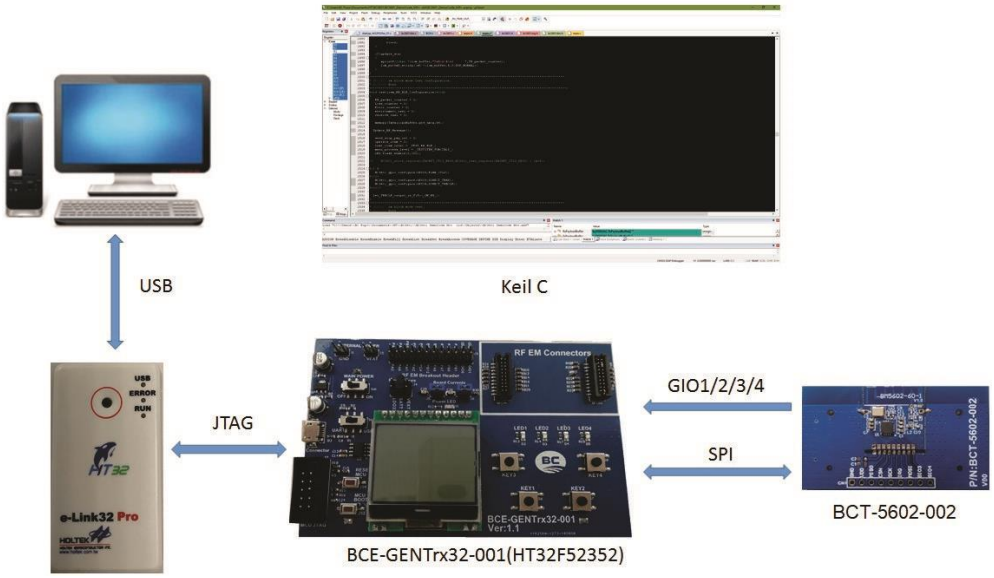



**Figure 2. System Architecture Diagram**

As the HT32F52352 is an Arm® Cortex®-M0+ based microcontroller, users need to first install the Keil uVision IDE on their computer. Then use Holtek's e-Link32 Pro and the JTAG interface to edit the firmware for the HT32F52352. For more information regarding the HT32F52352, refer to the following website: https://www.holtek.com/productdetail/-/vg/HT32F52342-52.

# System Development Board Introduction

The BCE-GENTrx32-001 development board provides satisfactory human-machine interfaces for convenient operations, as shown below.

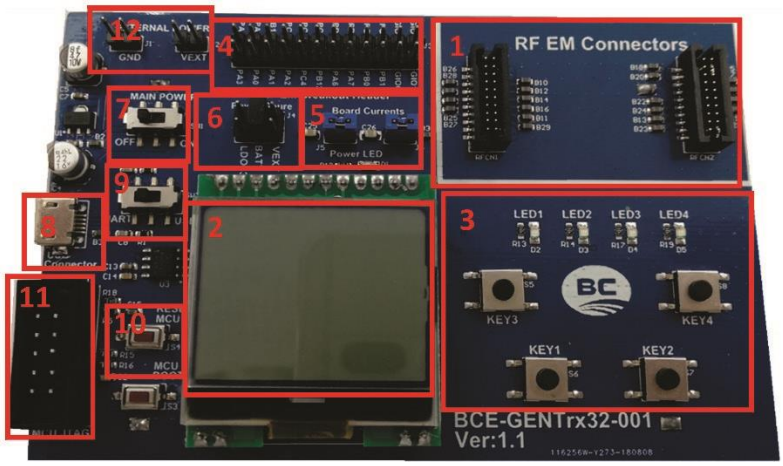



**Figure 3. System Development Board Composition**

The development board includes the following parts:

1. RF module interface: The RF transmitting/receiving device is connected to this interface. In this example, the BM5602-60-1 is used.

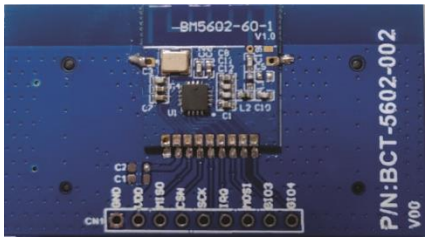



**Figure 4. BCT-5602-002**

2. LCD display supports 128×64 pixels: For more information refer to the demo code which contains the LCD control library.

3. Indicators: Four LEDs and four keys, which can be used by the user for indication and input control during program development. In this demo code, the key functions are KEY2 = Enter and the other key functions are defined by the user.

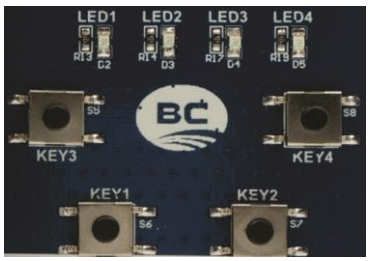



**Figure 5. LED & KEY**

4. I/O Interface, which includes the BCE-GNTrx32-001 I/O pins and the BM5602-60-1 corresponding pins, as shown below.

| BCE-GENTrx32-001 | BM5602-60-1 |
|---|---|
| PA3 | CSN |
| PA0 | SCK |
| PA1 | SDIO |
| PA2 | GIO1 |
| PC4 | GIO2 |
| GIO2 | GIO3 |
| GIO3 | GIO4 |

**Figure 6. I/O Interface**

5. MCU and BC5602 module board power current detection point.
6. System power selection: As shown in the Figure 7, if the jumper is connected to the LDO33, the power is supplied by the USB port; if the jumper is connected to the BATT, the power is supplied by the battery holder, which is powered by two 1.5V AA batteries on the back of the board; if the jumper is connected to the VEXT, the power is supplied by the external power as shown in Figure 9. It should be noted that the maximum operating voltage of the external power is 3.6V.

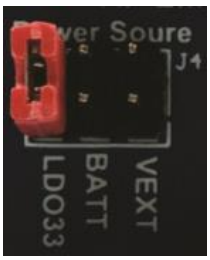



**Figure 7. Power Source Selection**

7. Main power switch. Turn the switch to the left to turn off the power while turn the switch to the right to turn on the power.

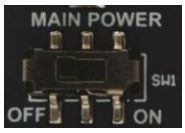



**Figure 8. Power Switch**

8. Micro USB interface, which can be used as a system power input if the power supply is selected by the LDO33.
9. UART/USB interface selection. When the UART is selected, the log will be dynamically output and the BC5602 status can be observed by the computer serial interface software.
10. System reset key.
11. JTAG interface, which can be used with the IDE interface to emulate and download programs.

12. External power connection point.


**Figure 9. External Power Connection Point**

# Pre-development Guidelines

The BC5602 demo code is currently developed using the Keil C software. The following will introduce how to use the Keil C software to compile and download the BC5602 demo code.

## Demo Code Type

The BC5602 demo code integrated in the BCE-GENTrx32-001 development platform is divided into seven types, including TX Carry, PER Mode, SB Mode, ESB Mode, DPL, DYN_ACK and ACK PLD. Refer to the BC5602 datasheet for the detailed operation mode description.

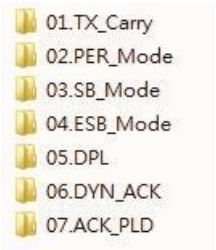



**Figure 10. Demo Code**

1. TX Carry: This program provides a single RF carrier output without the payload data field and the modulation.
2. PER (Packet Error Rate) Mode: The program is divided into the PTX and PRX parts. It can continuously transmit 32-byte packets and display PER on the PTX device.
3. SB (ShockBurst) Mode: The program is divided into the PTX and PRX parts. Press once to send a unidirectional 32-byte packet.
4. ESB (Enhanced ShockBurst) Mode: This program is divided into the PTX and PRX parts. Press once to send a bidirectional 32-byte packet.
5. DPL (Dynamic Payload Length): The program is divided into the PTX and PRX parts. Press once to send a bidirectional packet with adjustable length.
6. DYN_ACK (Dynamic ACK): The program is divided into the PTX and PRX parts. Press once to send a unidirectional 32-byte packet or a bidirectional 32-byte packet.
7. ACK PLD (PRX acknowledge with payload): The program is divided into the PTX and PRX parts. After receiving, the PRX will respond a 32-byte packet with ACK+Payload.

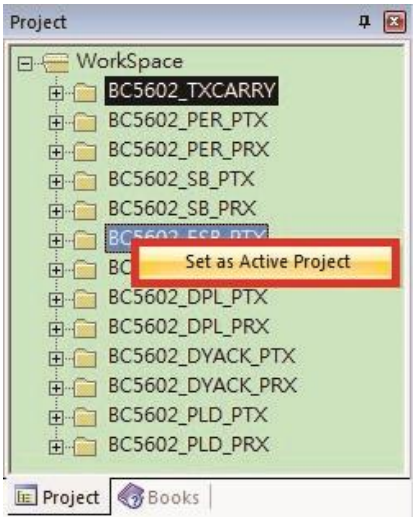
## Development Platform Hardware Installation

The hardware is mainly composed of an e-Link32 Pro, a BEC-GENTrx32-001 development board and a BM5602-60-1 module. The BM5602-60-1 module and the e-Link32 Pro interface are installed on the BEC-GENTrx32-001 development board. If the installations are completed, start the Keil C to start development.


**Figure 11. Development Platform Hardware Installation Diagram**

## Select Demo Code

Right-click on each project to select the desired active project.



# Demo Code Introduction

The demo code is divided into seven categories including 13 projects. The parameters will be introduced first, followed by the operation procedure and usage of each program example.

## Parameter Settings

Each project folder has a main.c, and each main.c includes parameter settings at the beginning, which is convenient for users to understand the parameters that need to be set in this project.

```
///************************ RF parameters Setup value *********************************/
#define _DEFAULT_ADDWidth          AW_5BYTE      //AW_3BYTE / AW_4BYTE / AW_5BYTE
#define _DEFAULT_CRCSET            CRC_16BIT     //0:CRC_OFF / 1:CRC_8BIT / 2:CRC_16BIT
#define _DEFAULT_RF_Channel        2442          //2400~2485MHz
#define _DEFAULT_RF_Power          P6dBm         //N5dBm / P0dBm / P5dBm / P6dBm
#define _DEFAULT_DATA_RATE         DR125KBPS     //DR125KBPS / DR250KBPS / DR500KBPS
#define _DEFAULT_PKT_Length        32            //1~32
/* set FEATURE register */
#define _DEFAULT_DYN_ACK           DIS_DYN_ACK   //EN_DYN_ACK / DIS_DYN_ACK
#define _DEFAULT_ACK_PAY           DIS_ACK_PAY   //EN_ACK_PAY / DIS_ACK_PAY
#define _DEFAULT_DPL               DIS_DPL       //EN_DPL / DIS_DPL
/* set pipe dynamic length */
#define _DEFAULT_DPL_P0            DIS_DPL_P0    //EN_DPL_P0 / DIS_DPL_P0
#define _DEFAULT_DPL_P1            DIS_DPL_P1    //EN_DPL_P1 / DIS_DPL_P1
#define _DEFAULT_DPL_P2            DIS_DPL_P2    //EN_DPL_P2 / DIS_DPL_P2
#define _DEFAULT_DPL_P3            DIS_DPL_P3    //EN_DPL_P3 / DIS_DPL_P3
#define _DEFAULT_DPL_P4            DIS_DPL_P4    //EN_DPL_P4 / DIS_DPL_P4
#define _DEFAULT_DPL_P5            DIS_DPL_P5    //EN_DPL_P5 / DIS_DPL_P5
/* set pipe active */
#define _DEFAULT_PIPE0             DIS_P0        //EN_P0 / DIS_P0
#define _DEFAULT_PIPE1             DIS_P1        //EN_P1 / DIS_P1
#define _DEFAULT_PIPE2             DIS_P2        //EN_P2 / DIS_P2
#define _DEFAULT_PIPE3             DIS_P3        //EN_P3 / DIS_P3
#define _DEFAULT_PIPE4             DIS_P4        //EN_P4 / DIS_P4
#define _DEFAULT_PIPE5             DIS_P5        //EN_P5 / DIS_P5
/* set RX pipe auto ack */
#define _DEFAULT_AAP0              DIS_AAP0      //EN_AAP0 / DIS_AAP0
#define _DEFAULT_AAP1              DIS_AAP1      //EN_AAP1 / DIS_AAP1
#define _DEFAULT_AAP2              DIS_AAP2      //EN_AAP2 / DIS_AAP2
#define _DEFAULT_AAP3              DIS_AAP3      //EN_AAP3 / DIS_AAP3
#define _DEFAULT_AAP4              DIS_AAP4      //EN_AAP4 / DIS_AAP4
#define _DEFAULT_AAP5              DIS_AAP5      //EN_AAP5 / DIS_AAP5
/* set auto repeat counter & delay timer */
#define _DEFAULT_ARD               ARD2750       //
#define _DEFAULT_ARC               DIS_ARC       //
/* set rx pipe payload width */
#define DEFAULT_RX_PW_P0           32            //1~32Bytes
#define DEFAULT_RX_PW_P1           32            //1~32Bytes
#define DEFAULT_RX_PW_P2           32            //1~32Bytes
#define DEFAULT_RX_PW_P3           32            //1~32Bytes
#define DEFAULT_RX_PW_P4           32            //1~32Bytes
#define DEFAULT_RX_PW_P5           32            //1~32Bytes
///************************************************************************************/
```

**Figure 12. Parameter List**

The parameters are described as follows.

1. _DEFAULT_ADDWidth: Address length (in byte) selection, 3 bytes, 4 bytes or 5 bytes.
2. _DEFAULT_CRCSET: CRC on/off and 8-bit or 16-bit CRC format selection.
3. _DEFAULT_RF_Channel: RF frequency selection, setting range: 2400~2485MHz.
4. _DEFAULT_RF_Power: RF output power setting, -5dBm, 0dBm, 5dBm or 6dBm.

5. _DEFAULT_DATA_RATE: Data rate setting, 125Kbps, 250Kbps or 500Kbps.
6. _DEFAULT_PKT_Length: Packet length, 1~32 bytes.
7. _DEFAULT_DYN_ACK: Enable “TX FIFO with No-Auto-ACK” command.
8. _DEFAULT_ACK_PAY: PRX acknowledge with payload function enable.
9. _DEFAULT_DPL: Dynamic payload length enable.
10. _DEFAULT_DPL_P0~_DEFAULT_DPL_P5: Dynamic Payload Length Control for pip0~pipe5.
11. _DEFAULT_PIPE0~_DEFAULT_PIPE5: Enable RX data pipe0~pipe5.
12. _DEFAULT_AAP0~_DEFAULT_AAP5: Enable auto acknowledgement data pipe0~pipe5.
13. _DEFAULT_ARD: Auto Retransmit Delay, 250μs~4000μs.
14. _DEFAULT_ARC: Auto Retransmit Count, 1~15 times.
15. DEFAULT_RX_PW_P0~DEFAULT_RX_PW_P5: Number of bytes in RX payload in data pipe0~pipe5.

## Main Program Flow

1. Initialisation: There are two initialisation types. One is the MCU function initialisation, including the CKCU, GPIO, LED, Button, LCM, BFTM and UART functions, the other is the BC5602 basic function initialisation, including the BC5602 interface configuration and BC5602 register initialisation.
2. Detect key action: Read and determine the key status. KEY2 is used for the start or stop function. KEY1 and KEY3 have different functions in different program examples.
3. Check IRQ state: Check the BC5602 hardware IRQ pin state. If it is low, this indicates that the IRQ flag has been set, after which the current IRQ state will be stored in the corresponding register and the BC5602 IRQ state will be cleared.
4. Program main loop (BC560x Program): TX Carry, PER Mode, SB Mode, ESB Mode, DPL, DYN_ACK and ACK PLD projects, the details of which will be described in the following sections.

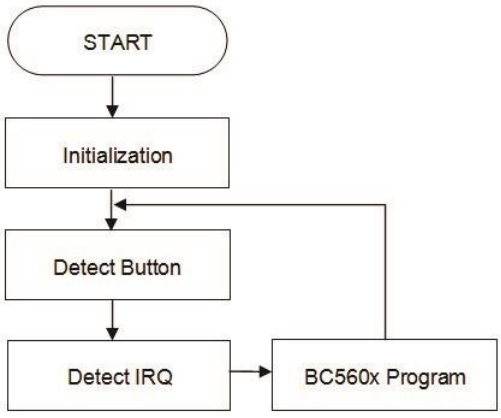



**Figure 13. Main Program Flowchart**

## TX Carry

This program provides a single tone RF carrier output without the payload data field and the modulation. When KEY2 is pressed, this program will continuously transmit a TX carrier signal until KEY2 is pressed again. KEY1 and KEY3 can also be used to adjust the frequency up and down.

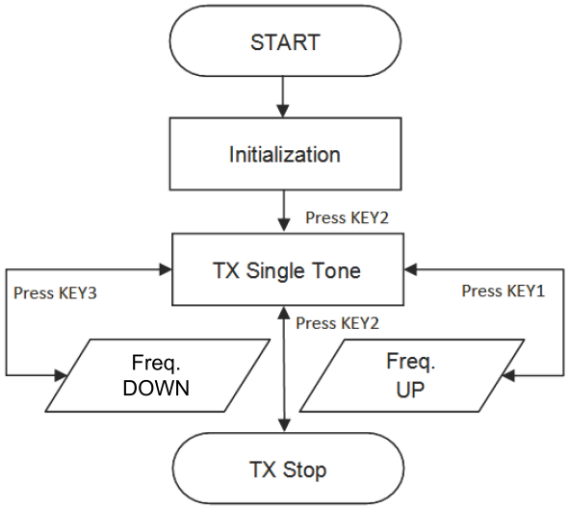



**Figure 14. TX Carry Flowchart**

## PER Mode

The PTX consecutively transmits 32-byte PN9 packets to the PRX. When the PTX receives an ACK packet sent by the PRX, a transmission operation is completed. The current PER will be dynamically displayed on the PTX device. The parameter settings are as follows:

```
/* set pipe active */
#define _DEFAULT_PIPE0          EN_P0       //EN_P0 / DIS_P0
#define _DEFAULT_PIPE1          DIS_P1      //EN_P1 / DIS_P1
#define _DEFAULT_PIPE2          DIS_P2      //EN_P2 / DIS_P2
#define _DEFAULT_PIPE3          DIS_P3      //EN_P3 / DIS_P3
#define _DEFAULT_PIPE4          DIS_P4      //EN_P4 / DIS_P4
#define _DEFAULT_PIPE5          DIS_P5      //EN_P5 / DIS_P5
/* set RX pipe auto ack */
#define _DEFAULT_AAP0           EN_AAP0     //EN_AAP0 / DIS_AAP0
#define _DEFAULT_AAP1           DIS_AAP1    //EN_AAP1 / DIS_AAP1
#define _DEFAULT_AAP2           DIS_AAP2    //EN_AAP2 / DIS_AAP2
#define _DEFAULT_AAP3           DIS_AAP3    //EN_AAP3 / DIS_AAP3
#define _DEFAULT_AAP4           DIS_AAP4    //EN_AAP4 / DIS_AAP4
#define _DEFAULT_AAP5           DIS_AAP5    //EN_AAP5 / DIS_AAP5
```

Operation method: Press KEY2 to start the PER Mode. and press KEY2 next time to stop the PER Mode.

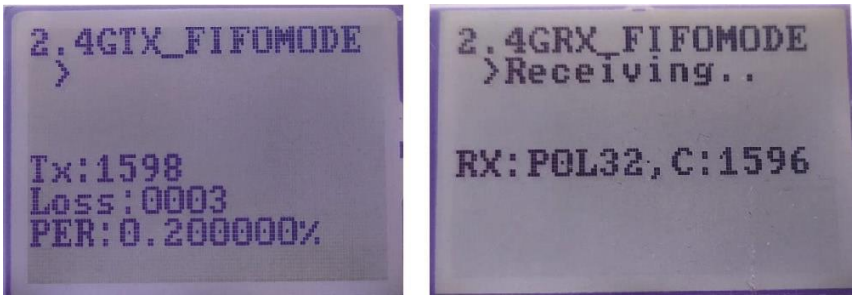



**Figure 15. PER Display LCM Data**

## SB Mode (Unidirectional Transmission)

The PTX transmits a 32-byte PN9 packet to the PRX each time.

Operation method: Press KEY2 to start the SB Mode, during which the PTX transmits a 32-byte PN9 packet to the RRX each time KEY1 is pressed. The SB Mode will be stoped when KEY2 is pressed again.

## ESB Mode (Bidirectional Transmission)

The PTX transmits a 32-byte PN9 packet to the PRX each time. When the PTX receives an ACK packet sent by the PRX, a transmission operation is completed.

Operation method: Press KEY2 to start the ESB Mode, during which the PTX transmits a 32-byte PN9 packet to the RRX each time KEY1 is pressed. The ESB Mode will be stoped when KEY2 is pressed again.

The parameter settings are the same as the PER Mode.

## DPL

The PTX transmits a PN9 packet with dynamic length of 0~32 bytes to the PRX each time.

Operation method: Press KEY2 to start the DPL, during which the PTX transmits a PN9 packet with the length being increased by 1 byte to the RRX each time KEY1 is pressed; the PTX transmits a PN9 packet with the length being decreased by 1 byte to the RRX each time KEY3 is pressed. The DPL will be stoped when KEY2 is pressed again.

The parameter settings are as follows:

```
/* set FEATURE register */
#define _DEFAULT_DYN_ACK          DIS_DYN_ACK   //EN_DYN_ACK / DIS_DYN_ACK
#define _DEFAULT_ACK_PAY          DIS_ACK_PAY   //EN_ACK_PAY / DIS_ACK_PAY
#define _DEFAULT_DPL              EN_DPL        //EN_DPL / DIS_DPL
/* set pipe dynamic length */
#define _DEFAULT_DPL_P0           EN_DPL_P0     //EN_DPL_P0 / DIS_DPL_P0
#define _DEFAULT_DPL_P1           DIS_DPL_P1    //EN_DPL_P1 / DIS_DPL_P1
#define _DEFAULT_DPL_P2           DIS_DPL_P2    //EN_DPL_P2 / DIS_DPL_P2
#define _DEFAULT_DPL_P3           DIS_DPL_P3    //EN_DPL_P3 / DIS_DPL_P3
#define _DEFAULT_DPL_P4           DIS_DPL_P4    //EN_DPL_P4 / DIS_DPL_P4
#define _DEFAULT_DPL_P5           DIS_DPL_P5    //EN_DPL_P5 / DIS_DPL_P5
/* set pipe active */
#define _DEFAULT_PIPE0            EN_P0         //EN_P0 / DIS_P0
#define _DEFAULT_PIPE1            DIS_P1        //EN_P1 / DIS_P1
#define _DEFAULT_PIPE2            DIS_P2        //EN_P2 / DIS_P2
#define _DEFAULT_PIPE3            DIS_P3        //EN_P3 / DIS_P3
#define _DEFAULT_PIPE4            DIS_P4        //EN_P4 / DIS_P4
#define _DEFAULT_PIPE5            DIS_P5        //EN_P5 / DIS_P5
/* set RX pipe auto ack */
#define _DEFAULT_AAP0             EN_AAP0       //EN_AAP0 / DIS_AAP0
#define _DEFAULT_AAP1             DIS_AAP1      //EN_AAP1 / DIS_AAP1
#define _DEFAULT_AAP2             DIS_AAP2      //EN_AAP2 / DIS_AAP2
#define _DEFAULT_AAP3             DIS_AAP3      //EN_AAP3 / DIS_AAP3
#define _DEFAULT_AAP4             DIS_AAP4      //EN_AAP4 / DIS_AAP4
#define _DEFAULT_AAP5             DIS_AAP5      //EN_AAP5 / DIS_AAP5
```

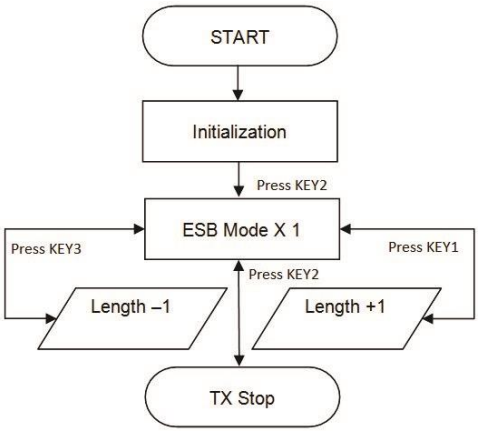



**Figure 16. DPL Flowchart**

## DYN_ACK

To facilitate users to switch between SB Mode (unidirectional transmission) and ESB Mode (bidirectional transmission) at any time, only EN_DYN_ACK needs to be set. After setting, two Strobe Commands, WRITE_NACKPLD_CMD (0x13) and WRITE_ACKPLD_CMD (0x11), can be used to easily switch between unidirectional and bidirectional transmission.

```
/* set FEATURE register */
#define _DEFAULT_DYN_ACK          EN_DYN_ACK    //EN_DYN_ACK / DIS_DYN_ACK
#define _DEFAULT_ACK_PAY          DIS_ACK_PAY   //EN_ACK_PAY / DIS_ACK_PAY
#define _DEFAULT_DPL              DIS_DPL       //EN_DPL / DIS_DPL
```

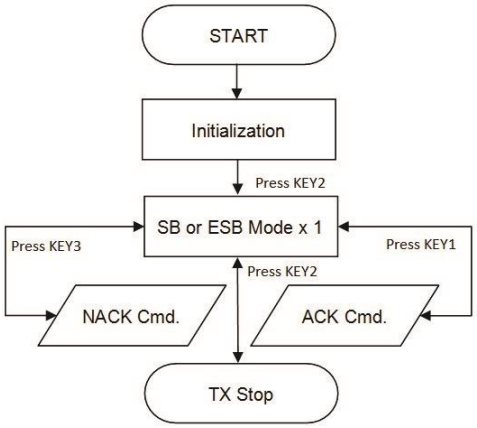



**Figure 17. DYN_ACK Flowchart**

### ACK PLD

When the PRX device has an information to send back to the PTX device, the data can be carried while the ACK is being returned. Both EN_ACK_PAY and EN_DPL must be set for normal operation.

```
/* set FEATURE register */
#define _DEFAULT_DYN_ACK        DIS_DYN_ACK   //EN_DYN_ACK / DIS_DYN_ACK
#define _DEFAULT_ACK_PAY        EN_ACK_PAY    //EN_ACK_PAY / DIS_ACK_PAY
#define _DEFAULT_DPL            EN_DPL        //EN_DPL / DIS_DPL
```

Note: 1. Use the Strobe command, Write Pipe 0 ACK Payload Command (0x18), before the PRX receives RX_DR IRQ to fill in the data that needs to be returned. If it is not filled in time, only ACK will be returned. The data will not be sent to the PTX along with ACK until the PRX receives RX_DR IRQ next time.

2. After the PTX has received the data returned by the PRX, when the PRX receives the next packet from the PTX, in addition to RX_DR IRQ being enabled, the TX DS(ACKPAY) IRQ will also be enabled which indicates that the PTX has received the data returned by the PRX.

# BC5602 API Function Introduction

The API list (51 items) and its functions can be obtained from the BC5602.h file in the demo code.

| Command Name | Functional Description |
|---|---|
| BC5602SoftwareReset() | Software Reset |
| BC5602RegisterBank(regbk) | Set Register Bank |
| BC5602DeepSleepMode() | Deep Sleep Mode Setting |
| BC5602MiddleSleepMode() | Middle Sleep Mode Setting |
| BC5602LightSleepMode() | Light Sleep Mode Setting |
| BC5602StandbyMode() | Standby Mode Setting |
| BC5602TransmitterMode() | TX Mode Trigger |
| BC5602ReceiveMode() | RX Mode Trigger |

| Command Name | Functional Description |
|---|---|
| BC5602TxFIFOFlush() | TX FIFO Flush |
| BC5602RxFIFOFlush() | RX FIFO Flush |
| BC5602WriteTxPayload(pbuf,len) | Write TX FIFO with Auto-ACK Mode Command |
| BC5602WriteTxNAckPayload(pbuf,len) | Write TX FIFO with No-Auto-ACK Mode Command |
| BC5602WriteRxAckPayload(pipe,pbuf,len) | Write len bytes of pbuf data to the Ack packet |
| BC5602ReadRxPayload(pbuf,len) | Read len bytes of data from RX Payload to pbuf |
| BC5602GetIRQLineStatus(void) | Obtain the current IRQ pin status |
| BC5602RegisterConfigure(void) | Configure the BC5602 special registers to their default values |
| BC5602CrystalSetup(u8 xo_il,u8 xo_trim) | Set the crystal current mode and internal capacitor trim value |
| BC5602WaitCrystalReady(void) | Wait for the crystal to stabilise |
| BC5602SetPrimaryMode(u8 op_mode) | Set the BC5602 primary mode |
| BC5602GetOperationMode(void) | Read the current BC5602 operation mode status |
| BC5602SetCRCMode(u8 crc_mode) | Set the BC5602 CRC format |
| BC5602SetIRQMode(u8 int_source, u8 irq_state) | Control the BC5602 IRQ (TX_DS, RX_DR, MAX_RT) enable/disable |
| BC5602GetClearIRQFlags(void) | Clear the interrupt control register 1 flags and return the value before clearing |
| BC5602SetAddressWidth(u8 aw) | Set the BC5602 address length |
| BC5602GetAddressWidth(void) | Read the BC5602 address length |
| BC5602SetAddress(const u8 adr,const u8 *adrbuf) | Set the PTX, PRX PIPE0, PIPE1, PIPE2, PIPE3, PIPE4 and PIPE5 addresses |
| BC5602GetAddress(const u8 adr,u8 *adrbuf) | Read the PTX, PRX PIPE0, PIPE1, PIPE2, PIPE3, PIPE4 and PIPE5 addresses |
| BC5602SetDataRate(u8 datarate) | Set the BC5602 data rate |
| BC5602OpenPipe(u8 pipe_num, u8 auto_ack) | Control a certain pipe open and Auto-ACK enable/disable |
| BC5602ClosePipe(u8 pipe_num) | Control a certain pipe close |
| BC5602SetAutoRetr(u8 retr, u8 delay) | Set the number of BC5602 auto retransmissions and the retransmission delay time |
| BC5602SetRFChannel(channel) | Set the BC5602 communication frequency |
| BC5602SetOutputPower(u8 power) | Set the BC5602 output power |
| BC5602GetIRQFlags(void) | Read the BC5602 interrupt control register 1 Bit4 ~ Bit6 |
| BC5602ClearIRQFlags(source) | Clear the BC5602 interrupt control register 1 flags |
| BC5602SetRxPayloadWidth(pipe_num,width) | Set the payload length of a certain BC5602 receive pipe |
| BC5602GetRxPayloadWidth(pipe_num) | Read the payload length of a certain BC5602 receive pipe |
| BC5602GetAutoRetrStatus(void) | Read the retransmission control register |

| Command Name | Functional Description |
|---|---|
| BC5602GetPacketLostCtr(void) | Read the number of packet lost times in the same frequency |
| BC5602GetFIFOStatus(void) | Read the TX FIFO and RX FIFO status |
| BC5602SetupFeature(setup) | Select the BC5602 Feature |
| BC5602SetupDynamicPayload(setup) | Enable the dynamic payload length function of a certain pipe |
| BC5602ReadRxPayloadWidth(void) | Read the RX data length from RX FIFO |
| BC5602SetupCE(setup) | Control the BC5602 enable/disable |
| BC5602GetAddressRSSI(void) | Read the received signal strength value when addresses match |
| BC5602GetMeasurementRSSI(void) | Read the current received signal strength value |
| BC5602StrobeCommand(u8 cmd) | Send the Strobe command |
| BC5602ReadRegister(u8 regs) | Read the value from a certain register |
| BC5602WriteRegister(u8 regs,u8 data) | Write a value to a certain register |
| BC5602ReadMultibyteRegister(u8 cmd,u8 *pbuf,u8 length) | Read n bytes of data from a certain register |
| BC5602WriteMultibyteRegister(u8 cmd,u8 *pbuf,u8 length) | Write n bytes of data to a certain register |

**Table 1. BC5602 API List**

The details regarding the module API are described as follows:

| **API Name** | **BC5602SoftwareReset()** |
|---|---|
| API Function | Software reset |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, a software reset will occur. |
| **API Name** | **BC5602RegisterBank(regbk)** |
| API Function | Select the BC5602 register bank |
| Input Parameter | REGS_BANK0 / REGS_BANK1 / REGS_BANK2 / REGS_BANK3 |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use REGS_BANK0 / REGS_BANK1 / REGS_BANK2 / REGS_BANK3 to operate this API.<br>1. REGS_BANK0 is equivalent to 0. After use, the BC5602 register bank will be set to Bank 0.<br>2. REGS_BANK1 is equivalent to 1. After use, the BC5602 register bank will be set to Bank 1.<br>3. REGS_BANK2 is equivalent to 2. After use, the BC5602 register bank will be set to Bank 2.<br>4. REGS_BANK3 is equivalent to 3. After use, the BC5602 register bank will be set to Bank 3. |
| **API Name** | **BC5602DeepSleepMode()** |
| API Function | Control the BC5602 to enter the Deep Sleep mode |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 will enter the Deep Sleep mode. |
| **API Name** | **BC5602MiddleSleepMode()** |

| | |
|---|---|
| API Function | Control the BC5602 to enter the Middle Sleep mode |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 will enter the Middle Sleep mode. |
| **API Name** | **BC5602LightSleepMode()** |
| API Function | Control the BC5602 to enter the Light Sleep mode |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 will enter the Light Sleep mode. |
| **API Name** | **BC5602StandbyMode()** |
| API Function | Control the BC5602 to enter the Standby mode |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 will enter the Standby mode. |
| **API Name** | **BC5602TransmitterMode()** |
| API Function | Control the BC5602 to enter the Transmit mode |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 will enter the Transmit mode. |
| **API Name** | **BC5602ReceiveMode()** |
| API Function | Control the BC5602 to enter the Receive mode |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 will enter the Receive mode. |
| **API Name** | **BC5602TxFIFOFlush()** |
| API Function | Clear TX FIFO data |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 TX FIFO data will be cleared. |
| **API Name** | **BC5602RxFIFOFlush()** |
| API Function | Clear RX FIFO data |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 RX FIFO data will be cleared. |
| **API Name** | **BC5602WriteTxPayload(pbuf,len)** |
| API Function | Write len bytes of pbuf data to TX FIFO with Auto-Ack mode |
| Input Parameter 1 | Data Memory Bank |
| Input Parameter 2 | Data length (1~32) |
| Output Parameter | — |
| Program Description | Note: The maximum length of the Input Parameter 2 cannot exceed 32 bytes. |

| API Name | BC5602WriteTxNAckPayload(pbuf,len) |
|---|---|
| API Function | Write len bytes of pbuf data to TX FIFO with No-Auto-Ack mode |
| Input Parameter 1 | Data Memory Bank |
| Input Parameter 2 | Data length (1~32) |
| Output Parameter | — |
| Program Description | Note: The maximum length of the Input Parameter 2 cannot exceed 32 bytes. |
| **API Name** | **BC5602WriteRxAckPayload(pipe,pbuf,len)** |
| API Function | Write len bytes of pbuf data to the Ack packet |
| Input Parameter 1 | pipe0 (0) / pipe1 (1) / pipe2 (2) / pipe3 (3) / pipe4 (4) / pipe5 (5) |
| Input Parameter 2 | Data Memory Bank |
| Input Parameter 3 | Data length (1~32) |
| Output Parameter | — |
| Program Description | Note: The maximum length, len (Input Parameter 2), cannot exceed 32 bytes. |
| **API Name** | **BC5602ReadRxPayload(pbuf,len)** |
| API Function | Read len bytes of data from RX Payload to pbuf |
| Input Parameter 1 | Data Memory Bank |
| Input Parameter 2 | Data length (1~32) |
| Output Parameter | — |
| Program Description | Note: The maximum length, len (Input Parameter 2), cannot exceed 32 bytes. |
| **API Name** | **BC5602GetIRQLineStatus(void)** |
| API Function | Obtain the current IRQ pin status |
| Input Parameter | — |
| Output Parameter | TRUE / FALSE |
| Program Description | In the program example, after this API is executed, if the BC5602 generates an interrupt but is not reset, this function will return TRUE; otherwise return FALSE. |
| **API Name** | **BC5602RegisterConfigure(void)** |
| API Function | Configure the BC5602 special registers to their default values |
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, the BC5602 special registers will be configured to improve the RF performance. |
| **API Name** | **BC5602CrystalSetup(u8 xo_il,u8 xo_trim)** |
| API Function | Set the crystal current mode and internal capacitor trim value |
| Input Parameter 1 | TRUE / FALSE |
| Input Parameter 2 | 0~31 |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter 1 can use TRUE/FALSE to operate this API.<br>1. TRUE is equivalent to 1. After use, the crystal low current mode will be enabled.<br>2. FALSE is equivalent to 0. After use, the crystal low current mode will be disabled.<br>The Input Parameter 2 can trim the crystal internal capacitor load value. Refer to the datasheet for the exact value. |
| **API Name** | **BC5602WaitCrystalReady(void)** |
| API Function | Wait for the crystal to stabilise |

| | |
|---|---|
| Input Parameter | — |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, it will determine whether the BC5602 crystal is stable or not. If it is, it will proceed to execute other programs. If it is unable to be stabilised, it will continue to execute this function and cannot leave. |

| **API Name** | **BC5602SetPrimaryMode(u8 op_mode)** |
|---|---|
| API Function | Set the BC5602 primary mode |
| Input Parameter | PRIM_PTX / PRIM_PRX |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use PRIM_PTX / PRIM_PRX to operate this API.<br>1. Input Parameter PRIM_PTX is equivalent to 0. After use, the BC5602 will be set as a PTX device.<br>2. Input Parameter PRIM_PRX is equivalent to 1. After use, the BC5602 will be set as a PRX device. |

| **API Name** | **BC5602GetOperationMode(void)** |
|---|---|
| API Function | Read the current BC5602 operation mode status |
| Input Parameter | — |
| Output Parameter | 0H / 1H / 2H / 3H / 4H / 5H / 6H |
| Program Description | In the program example, after this API is executed, it will return to the current BC5602 operation mode status:<br>0H: Deep Sleep mode<br>1H: Middle Sleep mode<br>2H: Light Sleep mode<br>3H: Standby mode<br>4H: TX mode<br>5H: RX mode<br>6H: Calibration mode |

| **API Name** | **BC5602SetCRCMode(u8 crc_mode)** |
|---|---|
| API Function | Set the BC5602 CRC format |
| Input Parameter | CRC_OFF / CRC_8BIT / CRC_16BIT |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use CRC_OFF / CRC_8BIT / CRC_16BIT to operate this API.<br>1. Input Parameter CRC_OFF is equivalent to 0. After use, the BC5602 CRC will be disabled.<br>2. Input Parameter CRC_8BIT is equivalent to 1. After use, the BC5602 CRC format will be set to CRC8=$X^8+X^2+X+1$.<br>3. Input Parameter CRC_8BIT is equivalent to 2. After use, the BC5602 CRC format will be set to CRC16=$X^{16}+X^{12}+X^5+1$. |

| **API Name** | **BC5602SetIRQMode(u8 int_source, u8 irq_state)** |
|---|---|
| API Function | Control the BC5602 IRQ (TX_DS, RX_DR, MAX_RT) enable/disable |
| Input Parameter 1 | IRQ_MAXRT / IRQ_TXDS / IRQ_RXDR / IRQ_ALL |
| Input Parameter 2 | ENABLE / DISABLE |
| Output Parameter | — |

| | |
|---|---|
| Program Description | In the program example, the Input Parameter 1 can directly use IRQ_MAXRT / IRQ_TXDS / IRQ_RXDR / IRQ_ALL, and the Input Parameter 2 can use ENABLE / DISABLE to operate this API.<br>1. The Input Parameter 1 IRQ_MAXRT is equivalent to 4. If the Input Parameter 2 is DISABLE, when the maximum number of BC5602 retransmissions reaches, no interrupt will be generated; otherwise, the MAX_RT interrupt will be generated.<br>2. The Input Parameter 1 IRQ_TXDS is equivalent to 5. If the Input Parameter 2 is DISABLE when the BC5602 detects the completion of a data transfer, no interrupt will be generated; otherwise, the TX_DS interrupt will be generated.<br>3. The Input Parameter 1 IRQ_RXDR is equivalent to 6. If the Input Parameter 2 is DISABLE, when the BC5602 detects that data has been received, no interrupt will be generated; otherwise, the RX_DR interrupt will be generated.<br>4. The Input Parameter 1 IRQ_ALL is equivalent to 0xFF. If the Input Parameter 2 is DISABLE, all of the IRQ_MAXRT, TX_DS and RX_DR interrupts will be disabled; otherwise, these interrupts will be enabled. |
| **API Name** | **BC5602GetClearIRQFlags(void)** |
| API Function | Clear the interrupt control register 1 flags and return the value before clearing |
| Input Parameter | — |
| Output Parameter | Interrupt control register 1 (04H) value |
| Program Description | In the program example, after this API is executed, it can clear Bit 4 (MAX_RT), Bit 5 (TX_DS) and Bit 6 (RX_DR) in the BC5602 interrupt control register (04H) and return the register value before clearing. |
| **API Name** | **BC5602SetAddressWidth(u8 aw)** |
| API Function | Set the BC5602 address length |
| Input Parameter | AW_3BYTE / AW_4BYTE / AW_5BYTE |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use AW_3BYTE / AW_4BYTE / AW_5BYTE to operate this API.<br>1. AW_3BYTE is equivalent to 1. After use, the BC5602 address length will be set to 3 bytes.<br>2. AW_4BYTE is equivalent to 2. After use, the BC5602 address length will be set to 4 bytes.<br>3. AW_5BYTE is equivalent to 3. After use, the BC5602 address length will be set to 5 bytes. |
| **API Name** | **BC5602GetAddressWidth(void)** |
| API Function | Read the BC5602 address length |
| Input Parameter | — |
| Output Parameter | 1 / 2 / 3 |
| Program Description | After this API is executed, it will return the current BC5602 address length.<br>1. If the Output Parameter returns 1, the current BC5602 address length will be set to 3 bytes.<br>2. If the Output Parameter returns 2, the current BC5602 address length will be set to 4 bytes.<br>3. If the Output Parameter returns 3, the current BC5602 address length will be set to 5 bytes. |
| **API Name** | **BC5602SetAddress(const u8 adr, const u8 *adrbuf)** |
| API Function | Set the PTX, PRX PIPE0, PIPE1, PIPE2, PIPE3, PIPE4 and PIPE5 addresses |
| Input Parameter 1 | TRXADDR / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 |
| Input Parameter 2 | Input address data |
| Output Parameter | — |

| | |
|---|---|
| Program Description | In the program example, after this API is executed, it will set the address shifted by the Input Parameter 2 to the communication address of the corresponding pipe (Input Parameter 1). The Input Parameter 1 can directly use TRXADDR / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 to operate this API.<br>1. The Input Parameter 1 TRXADDR is equivalent to 0, and the Input Parameter 2 is the address for setting the PTX and PRX PIPE0 pipe.<br>2. The Input Parameter 1 PIPE1 is equivalent to 1, and the Input Parameter 2 is the address for setting the PIPE1 receive pipe.<br>3. The Input Parameter 1 PIPE2 is equivalent to 2, and the Input Parameter 2 is the address for setting the PIPE2 receive pipe.<br>4. The Input Parameter 1 PIPE3 is equivalent to 3, and the Input Parameter 2 is the address for setting the PIPE3 receive pipe.<br>5. The Input Parameter 1 PIPE4 is equivalent to 4, and the Input Parameter 2 is the address for setting the PIPE4 receive pipe.<br>6. The Input Parameter 1 PIPE5 is equivalent to 5, and the Input Parameter 2 is the address for setting the PIPE5 receive pipe. |
| **API Name** | **BC5602GetAddress(const u8 adr,u8 *adrbuf)** |
| API Function | Read the PTX, PRX PIPE0, PIPE1, PIPE2, PIPE3, PIPE4 and PIPE5 addresses |
| Input Parameter 1 | TRXADDR / PIPE1 / PIPE2 /PIPE3 / PIPE4 / PIPE5 |
| Input Parameter 2 | Data for storing address |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, it can read the pipe (Input Parameter 1) to the corresponding storage area (Input Parameter 2). The Input Parameter 1 can directly use TRXADDR / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 to operate this API.<br>1. The Input Parameter 1 TRXADDR is equivalent to 0, and the Input Parameter 2 is the data for storing the address of the PTX and PRX PIPE0 receive pipes.<br>2. The Input Parameter 1 PIPE1 is equivalent to 1, and the Input Parameter 2 is the data for storing the address of the PIPE1 receive pipe.<br>3. The Input Parameter 1 PIPE2 is equivalent to 2, and the Input Parameter 2 is the data for storing the address of the PIPE2 receive pipe.<br>4. The Input Parameter 1 PIPE3 is equivalent to 3, and the Input Parameter 2 is the data for storing the address of the PIPE3 receive pipe.<br>5. The Input Parameter 1 PIPE4 is equivalent to 4, and the Input Parameter 2 is the data for storing the address of the PIPE4 receive pipe.<br>6. The Input Parameter 1 PIPE5 is equivalent to 5, and the Input Parameter 2 is the data for storing the address of the PIPE5 receive pipe. |
| **API Name** | **BC5602SetDataRate(u8 datarate)** |
| API Function | Set the BC5602 data rate |
| Input Parameter | DR500KBPS / DR250KBPS / DR125KBPS |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use DR500KBPS / DR250KBPS / DR125KBPS to operate this API.<br>1. DR500KBPS is equivalent to 0. After use, the BC5602 data rate will be set to 500Kbps.<br>2. DR250KBPS is equivalent to 1. After use, the BC5602 data rate will be set to 250Kbps.<br>3. DR125KBPS is equivalent to 2. After use, the BC5602 data rate will be set to 125Kbps. |

| API Name | BC5602OpenPipe(u8 pipe_num, u8 auto_ack) |
|---|---|
| API Function | Control a certain pipe open and Auto-ACK enable/disable |
| Input Parameter 1 | PIPE0 / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 / PIPEALL |
| Input Parameter 2 | ENABLE / DISABLE |
| Output Parameter | — |
| Program Description | The Input Parameter 1 can directly use PIPE0 / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 / PIPEALL, and the Input Parameter 2 can directly use ENABLE / DISABLE to operate this API.<br>1. The Input Parameter 1 PIPE0 is equivalent to 0, at which point PIPE0 will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of PIPE0 will be disabled; otherwise the Auto-ACK function will be enabled.<br>2. The Input Parameter 1 PIPE1 is equivalent to 1, at which point PIPE1 will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of PIPE1 will be disabled; otherwise the Auto-ACK function will be enabled.<br>3. The Input Parameter 1 PIPE2 is equivalent to 2, at which point PIPE2 will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of PIPE2 will be disabled; otherwise the Auto-ACK function will be enabled.<br>4. The Input Parameter 1 PIPE3 is equivalent to 3, at which point PIPE3 will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of PIPE3 will be disabled; otherwise the Auto-ACK function will be enabled.<br>5. The Input Parameter 1 PIPE4 is equivalent to 4, at which point PIPE4 will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of PIPE4 will be disabled; otherwise the Auto-ACK function will be enabled.<br>6. The Input Parameter 1 PIPE5 is equivalent to 5, at which point PIPE5 will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of PIPE5 will be disabled; otherwise the Auto-ACK function will be enabled.<br>7. The Input Parameter 1 PIPEALL is equivalent to 0xFF, at which point all pipes will be opened. If the Input Parameter 2 is DISABLE, the Auto-ACK function of all pipes will be disabled; otherwise, the Auto-ACK function will be enabled. |

| API Name | BC5602ClosePipe(u8 pipe_num) |
|---|---|
| API Function | Control a certain pipe close |
| Input Parameter | PIPE0 / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 / PIPEALL |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use PIPE0 / PIPE1 / PIPE2 / PIPE3 / PIPE4 / PIPE5 / PIPEALL to operate this API.<br>1. PIPE0 is equivalent to 0. After execution, PTX and PRX PIPE0 will be closed.<br>2. PIPE1 is equivalent to 1. After execution, PIPE1 will be closed.<br>3. PIPE2 is equivalent to 2. After execution, PIPE2 will be closed.<br>4. PIPE3 is equivalent to 3. After execution, PIPE3 will be closed.<br>5. PIPE4 is equivalent to 4. After execution, PIPE4 will be closed.<br>6. PIPE5 is equivalent to 5. After execution, PIPE5 will be closed.<br>7. PIPEALL is equivalent to 0xFF. After execution, all pipes will be closed. |

| API Name | BC5602SetAutoRetr(u8 retr, u8 delay) |
|---|---|
| API Function | Set the number of BC5602 auto retransmissions and the retransmission delay time |
| Input Parameter 1 | 0~15 |

| | |
|---|---|
| Input Parameter 2 | ARD_DLY250 / ARD_DLY500 / ARD_DLY750 / ARD_DLY1000 / ARD_DLY1250 / ARD_DLY1500 / ARD_DLY1750 / ARD_DLY2000 / ARD_DLY2250 / ARD_DLY2500 / ARD_DLY2750 / ARD_DLY3000 / ARD_DLY3250 / ARD_DLY3500 / ARD_DLY3750 / ARD_DLY4000 |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter 1 can directly use 0~15 to set the number of auto retransmissions, and the Input Parameter 2 can directly use ARD_DLY250~ARD_DLY4000 to select the retransmission delay time ranging from 250μs~4000μs. Refer to the PTX Retransmission Control Register (13H) description in the Datasheet for more details. |
| **API Name** | **BC5602SetRFChannel(channel)** |
| API Function | Set the BC5602 communication frequency |
| Input Parameter | 2~85 |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use 2~85 to set the communication frequency. After execution, the BC5602 communication frequency will be set to (2400+channel)MHz. |
| **API Name** | **BC5602SetOutputPower(u8 power)** |
| API Function | Set the BC5602 output power |
| Input Parameter | N5dBm / P0dBm / P5dBm / P6dBm |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can directly use N5dBm / P0dBm / P5dBm / P6dBm to operate this API.<br>1. N5dBm is equivalent to 0. After execution, the BC5602 output power will be set to -5dBm.<br>2. P0dBm is equivalent to 1. After execution, the BC5602 output power will be set to 0dBm.<br>3. P5dBm is equivalent to 2. After execution, the BC5602 output power will be set to 5dBm.<br>4. P6dBm is equivalent to 3. After execution, the BC5602 output power will be set to 6dBm. |
| **API Name** | **BC5602GetIRQFlags(void)** |
| API Function | Read the BC5602 interrupt control register 1 Bit 4 ~ Bit 6 |
| Input Parameter | — |
| Output Parameter | 10H / 20H / 40H |
| Program Description | In the program example, after this API is executed, the current status of the interrupt control register 1 Bit 4 ~ Bit 6 can be read.<br>1. If the Output Parameter is 10H, this shows that the MAX_RT interrupt occurs.<br>2. If the Output Parameter is 20H, this shows the TX_DS interrupt occurs.<br>3. If the Output Parameter is 40H, this shows that the RX_DR interrupt occurs and the data can be read from RX FIFO. |
| **API Name** | **BC5602ClearIRQFlags(source)** |
| API Function | Clear the BC5602 interrupt control register 1 status flags |
| Input Parameter | _MASK_MAXRT_ / _MASK_TXDS_ / _MASK_RXDR_ |
| Output Parameter | — |
| Program Description | In the program example, after this API is executed, input the corresponding parameter to clear the corresponding interrupt.<br>1. _MASK_MAXRT_ is equivalent to 10H, this shows that the MAX_RT interrupt will be cleared. |

| | |
|---|---|
| | 2. _MASK_TXDS_ is equivalent to 20H, this shows that the TX_DS interrupt will be cleared.<br>3. _MASK_RXDR_ is equivalent to 40H, this shows that the RX_DR interrupt will be cleared. |
| **API Name** | **BC5602SetRxPayloadWidth(pipe_num,width)** |
| API Function | Set the payload length of a certain BC5602 receive pipe |
| Input Parameter 1 | RX_PIPE0 / RX_PIPE1 / RX_PIPE2 / RX_PIPE3 / RX_PIPE4 / RX_PIPE5 |
| Input Parameter 2 | 0~32 |
| Output Parameter | — |
| Program Description | Note: This set length will not be used when the corresponding pipe dynamic length function is disabled. |
| **API Name** | **BC5602GetRxPayloadWidth(pipe_num)** |
| API Function | Read the payload length of a certain BC5602 receive pipe |
| Input Parameter | RX_PIPE0 / RX_PIPE1 / RX_PIPE2 / RX_PIPE3 / RX_PIPE4 / RX_PIPE5 |
| Output Parameter | Value |
| Program Description | In the program example, after this API is executed, it will return a value, which is the payload length of the setting input communication pipe. |
| **API Name** | **BC5602GetAutoRetrStatus(void)** |
| API Function | Read the retransmission control register |
| Input Parameter | — |
| Output Parameter | Value |
| Program Description | In the program example, after this API is executed, the retransmission control register value can be read. The high nibble represent the number of packet lost times in the same frequency and the low nibble represent the number of packet retransmission times. |
| **API Name** | **BC5602GetPacketLostCtr(void)** |
| API Function | Read the number of packet lost times in the same frequency |
| Input Parameter | — |
| Output Parameter | The number of packet lost times |
| Program Description | In the program example, after this API is executed, the number of packet lost times in the same frequency can be read from the retransmission control register. |
| **API Name** | **BC5602GetFIFOStatus(void)** |
| API Function | Read the TX FIFO and RX FIFO status |
| Input Parameter | — |
| Output Parameter | Value |
| Program Description | In the program example, after this API is executed, the FIFO register value can be read. Only Bit 0, Bit 1, Bit 4 and Bit 5 are valid. The specific meaning of these four bits is as follows:<br>1. Bit 0: If it is 1, RX FIFO is empty; otherwise, RX FIFO has data.<br>2. Bit 1: If it is 1, RX FIFO is full; otherwise, RX FIFO is not full.<br>3. Bit 4: If it is 1, TX FIFO is empty; otherwise, TX FIFO has data.<br>4. Bit 5: If it is 1, TX FIFO is full; otherwise, TX FIFO is not full. |
| **API Name** | **BC5602SetupFeature(setup)** |
| API Function | Select the BC5602 Feature |
| Input Parameter | Value |
| Output Parameter | — |

| | |
|---|---|
| Program Description | In the program example, the Input Parameter only requires to set Bit 0, Bit 1, Bit 2 and Bit 7 when this API is executed. Each bit function is as follows:<br>1. Bit 0: PTX "write TX FIFO with No-Auto-ACK" command enable<br>2. Bit 1: PRX acknowledge with payload function enable<br>3. Bit 2: Dynamic payload length enable<br>4. Bit 7: NO_ACK bit function control |
| **API Name** | **BC5602SetupDynamicPayload(setup)** |
| API Function | Enable the dynamic payload length function of a certain pipe |
| Input Parameter | 0x01 / 0x02 / 0x04 / 0x08 / 0x10 / 0x20 which correspond to pipe0 / pipe1 / pipe2 / pipe3 / pipe4 / pipe5 |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter has 6 bits of data. Bit 0 ~ Bit 5 represent pipe0 ~ pipe5. If it is set to 1, the dynamic length Payload function of the corresponding pipe will be enabled; otherwise, the function will be disabled. |
| **API Name** | **BC5602ReadRxPayloadWidth(void)** |
| API Function | Read the current received data length from RX FIFO |
| Input Parameter | — |
| Output Parameter | Data length |
| Program Description | In the program example, after this API is executed, it will output the current received data length from the RX FIFO. |
| **API Name** | **BC5602SetupCE(setup)** |
| API Function | Control the BC5602 enable/disable |
| Input Parameter | CE / ~CE |
| Output Parameter | — |
| Program Description | In the program example, the Input Parameter can use CE / ~CE to operate this API.<br>1. CE is equivalent to 1. If the Input Parameter is CE, it will enable the BC5602.<br>2. ~CE is equivalent to 0. If the Input Parameter is ~CE, it will disable the BC5602. |
| **API Name** | **BC5602GetAddressRSSI(void)** |
| API Function | Read the received signal strength value when addresses match |
| Input Parameter | — |
| Output Parameter | Signal strength value |
| Program Description | In the program example, after this API is executed, it will return a value, which is the received signal strength at the instant the BC5602 has read an address match. |
| **API Name** | **BC5602GetMeasurementRSSI(void)** |
| API Function | Read the current received signal strength value |
| Input Parameter | — |
| Output Parameter | Signal strength value |
| Program Description | In the program example, after this API is executed, it will return a value, which is the received signal strength when the BC5602 has read the data. |
| **API Name** | **BC5602StrobeCommand(u8 cmd)** |
| API Function | Send the Strobe Command |
| Input Parameter | Strobe Command |
| Output Parameter 1 | — |

| | |
|---|---|
| Program Description | Refer to the Strobe Command Table in the Datasheet for more details. |
| **API Name** | **BC5602ReadRegister(u8 regs)** |
| API Function | Read the value from a certain register |
| Input Parameter | Register address |
| Output Parameter | Value read from the register |
| Program Description | The Input Parameter is the address of the register to be read. For the exact register address, refer to the corresponding register description in the Datasheet. After this API is executed, the input register value can be read using the function. |
| **API Name** | **BC5602WriteRegister(u8 regs,u8 data)** |
| API Function | Write a value to a certain register |
| Input Parameter | Register address |
| Input Parameter | Value to be written |
| Output Parameter | — |
| Program Description | The Input Parameter is the address of the register to be written. For the exact register address, refer to the corresponding register description in the Datasheet. |
| **API Name** | **BC5602ReadMultibyteRegister(u8 cmd,u8 *pbuf,u16 length)** |
| API Function | Read n bytes of data from a certain register |
| Input Parameter 1 | Register address |
| Input Parameter 2 | Area for storing data |
| Input Parameter 3 | Length to be read |
| Output Parameter | — |
| Program Description | The Input Parameter 1 is the address of the register to be written. For the exact register address, refer to the corresponding register description in the Datasheet. |
| **API Name** | **BC5602WriteMultibyteRegister(u8 cmd,u8 *pbuf,u16 length)** |
| API Function | Write n bytes of data to a certain register |
| Input Parameter 1 | Register address |
| Input Parameter 2 | Area for storing data |
| Input Parameter 3 | Length to be written |
| Output Parameter | — |
| Program Description | The Input Parameter 1 is the address of the register to be written. For the exact register address, refer to the corresponding register description in the Datasheet. |

# Conclusion

This user guide has introduced how to use the seven major applications related to the BC5602, and users in subsequent program editing can also obtain program examples through Bestcomm.

# Reference Material

Consult the BC5602 datasheet.

For more information consult the Holtek website www.holtek.com.

# Version and Modification Information

| Date | Author | Issue and Modification Information |
|---|---|---|
| 2020.04.23 | 徐鴻文(Harry Hsu) | First Version |