**Bluetooth Module Description Document**

**1.Introduction**

Shanjing Bluetooth uses a main MCU (AP80) plus RF radio frequency chip configuration.

The currently supported RF chips include RDA5875, CSR BC6, Broadcom 20702, MTK6626, and Realtek 8761. Among these, the Realtek 8761 RF chip supports Bluetooth 4.0, while the others only support Bluetooth 2.1. The communication method between the MCU and the RF chip uses the BUART channel (for related BUART information, please refer to the BUART module manual) and the PCM channel. The PCM channel can be omitted in the Realtek 8761 chip, thereby freeing up 4 GPIO pins for other uses.

The protocol stack supports Bluetooth 4.0 protocols, including traditional Bluetooth 2.1 and Low Energy. Among them, traditional Bluetooth 2.1 supports HFP1.5 (HandsFree), A2DP1.2 (SINK), AVRCP1.5 (Advanced AVRCP), HID, SPP, and Apple's MFi.

In the protocol stack, traditional Bluetooth and BLE are two independent modules that only share BUART for data transmission. Logically, they can be understood as not interfering with each other.

**2.Bluetooth RF Initialization**

When using the Bluetooth module, you must first initialize the RF chip for later control. The RF initialization function is BTDeviceInit(uint8\_t BTDeviceType, int8\_t \*BdAdd); a non-zero return value indicates successful initialization. After that, the Bluetooth module will be taken over by the protocol stack for control.

**3.Traditional Bluetooth Protocol Stack**

**3.1 Introduction**

Traditional Bluetooth refers to the Bluetooth protocol stack that supports Bluetooth 2.1. Currently, Shansheng Bluetooth supports HFP 1.5 (HF end), A2DP1.2 (SINK end), AVRCP1.5 (CT end), HID (client end), SPP (DevB), OBEX (server end), and Apple's MFi.

**3.2 Initialization of the Traditional Bluetooth Protocol Stack**

The traditional Bluetooth protocol stack initialization uses BTStackRunInit(uint8\_t BTDeviceType, uint8\_t \*UserDefinedDevLocalName, uint8\_t BtFeatureID); returning a non-zero value indicates that the protocol stack initialization was successful.

**3.3 Traditional Bluetooth Operation**

The operation of the Bluetooth protocol stack relies on the function BTStackRun(). This function must be continuously called within a loop to ensure the execution of the protocol stack's internal logic and data transmission between the MCU and RF. Each call to BTStackRun() causes the MCU to check the BUART for data transmitted from the RF and whether there is any data that needs to be transmitted to the RF via the BUART. At the same time, the internal logic of the protocol stack runs once to check whether there is any parsed data to be passed to the upper-layer application and whether there is any data that needs to be passed from the upper-layer application to the lower-layer protocol stack.

Note: BTStackRun() not only provides transmission and protocol stack logic processing for traditional Bluetooth, but also provides transmission and BLE protocol stack logic processing for BLE.

**3.4 The End of Traditional Bluetooth**

When the Bluetooth protocol stack no longer needs to run, you need to exit BTStackRun() and call BTStackRunEnd() to clean up the memory inside the protocol stack.

**4.BLE Protocol Stack**

**4.1 Introduction**

BLE stands for Bluetooth Low Energy. It is a supplement to traditional Bluetooth in the low-power consumption field and can exist independently or coexist with traditional Bluetooth. RF chips that coexist with traditional Bluetooth are called dual-mode chips. Realtek 8761 is such a dual-mode chip.

**4.2 Initialization of the BLE Protocol Stack**

The function InitBleStack() is the initialization function of the BLE protocol stack. BLE roles are generally divided into two types: Central and Peripheral. Peripheral generally plays the role of the GATT server end, so during initialization, it is necessary to specify the BLE role and its profile.

**4.3 Operation of the BLE Protocol Stack**

The operation of the BLE protocol stack is the same as traditional Bluetooth, both of which require BTStackRun() to be called in a loop to enable data transmission and reception as well as logical processing.

**4.4 End of BLE Protocol Stack**

When BLE is no longer needed to continue running, call UninitBleStack() to terminate the operation and release the memory space allocated by the BLE protocol stack.

**5.Bluetooth Operation Diagram**

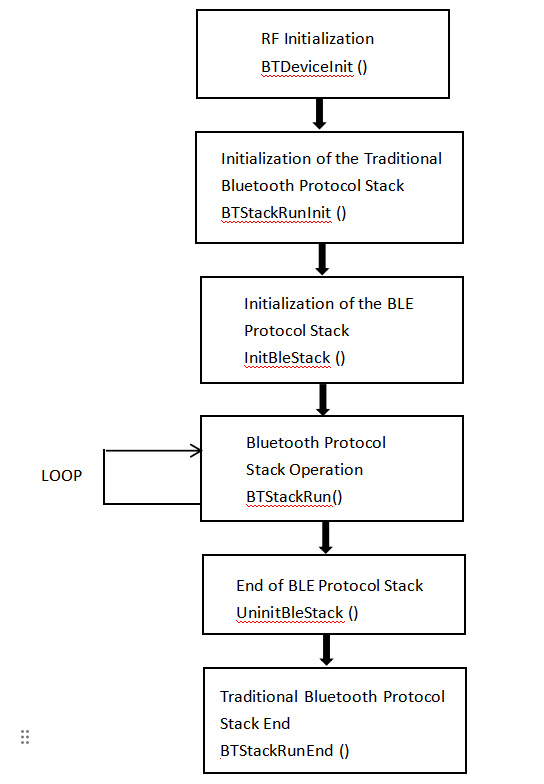


Figure 1 Bluetooth Operation Flowchart

**6.Bluetooth Reconnection Mechanism**

**6.1 Introduction**

Bluetooth reconnection refers to the SDK connecting to the most recently connected Bluetooth (mobile phone) device. If there has been no previous connection or the Flash memory has been erased, reconnection will not be possible.

**6.2 Functions Related to the Reconnection Mechanism:**

void StartBtReConnection(uint8\_t reconnection\_count);

This function initiates the reconnection mechanism, with the parameter reconnection\_count specifying the number of connection attempts.

void StopBtReConnection(void);

This function stops the reconnection mechanism regardless of whether the number of attempts has been reached.

bool IsBtReConnectionFinish(void);

This function determines whether the reconnection mechanism has ended or whether reconnection is in progress.

**6.3 Reconnection Interval Time(RECONNECTION\_PERIOD\_TIME)**

When the number of attempts to reconnect exceeds one, there is a problem of the interval between two reconnections. The interval time is to allow other devices to pair and connect with the Bluetooth of the SDK during this interval time. It avoids the problem that other devices cannot connect to the SDK throughout the entire reconnection mechanism.

**7.Bluetooth Call**

**7.1 Introduction**

Shanjing Bluetooth currently supports HFP1.5 (HF end), that is, the Bluetooth headset end. Its functions include voice calls and control, as well as phonebook reading. Bluetooth call voice data can be sent and received either through the PCM path or by using BURAT (eliminating the need for the PCM path method). Among them, the method of omitting the PCM path is currently only suitable for Realtek 8761.

**7.2 HFP Operating Mechanism**

HFP uses a mechanism that runs based on conversions between event notifications and state machines. The conversions between event notifications and state machines are shown in the figure below.

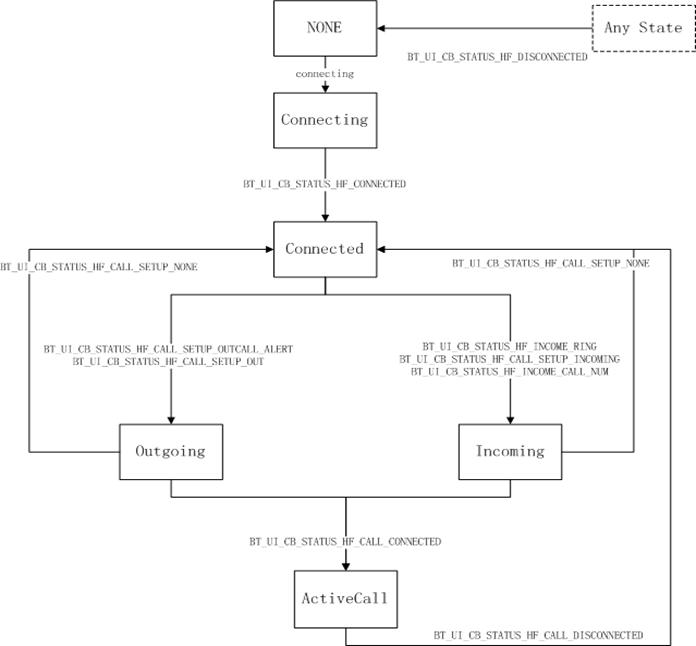


Figure 2 HFP operation mechanism

**7.3 HFP Control Commands**

See bt\_control\_api.h for details

**7.4 HFP PCM Bypass Mode**

This mode is controlled by the macro FUNC\_BT\_HF\_UART. When this macro is enabled, the SCO data of HFP will be transferred to the MCU via BUART. The voice data contained in it is in PCM format and is processed by the function RTL8761ScoProcessPCM() in pcm\_sync.c.

**7.5 HFP Events**

HFP events are defined in bt\_app\_ui\_callback.h under BT\_UI\_CB\_STATUS\_HF\_XXXX. For detailed descriptions, please refer to the comments.

1. **Bluetooth Music Playback**

**8.1 Introduction**

Bluetooth music playback includes A2DP and AVRCP. A2DP is for music data stream transmission, while AVRCP is for music control commands. When connecting to Bluetooth A2DP, the system will automatically connect to AVRCP.

**8.2 A2DP**

Shanjing Bluetooth A2DP only supports the Sink end, that is, the end receiving the data stream, which is passive reception. Whether the A2DP has a data stream is determined by the Source end (i.e., the mobile phone end), so the A2DP only acquires the status.

The state that can be obtained through GetA2dpState():

BT\_A2DP\_STATE\_CONNECTED indicates that A2DP is connected but there is no audio stream data.

BT\_A2DP\_STATE\_STREAMING indicates that A2DP is transmitting data streams (but this does not necessarily mean that a song is playing; it could be an empty data stream).

**8.3 AVRCP**

The role of Shanjing Bluetooth AVRCP is the Control end, which is the command initiation end. Commands can be initiated locally to the Target end (i.e., the mobile phone end). The control commands of AVRCP are all asynchronous. The protocol stack can only execute one command at a time. The application needs to ensure that the previous command has completed (the protocol stack accepts the command) before executing the next command. When the command interface function of AVRCP is executed and returns a non-zero value, it indicates that the command has been accepted by the protocol stack. The application layer generally does not need to wait for the response from the Target end to continue sending the next command, but it cannot be guaranteed whether the Target end accepts the protocol stack to execute the corresponding operation. The protocol stack can only ensure that the corresponding command is delivered. If the Target side does not support a certain command, the protocol stack will have relevant event reports.

The command of AVRCP can be referred to bt\_control\_api.h

**8.4 Advanced AVRCP**

Advanced AVRCP functionality is only available after receiving the BT\_UI\_CB\_STATUS\_ADV\_AVRCP\_SUPPORTED event. If the event is not received or the [BT\_UI\_CB\_STATUS\_ADV\_AVRCP\_NOT\_SUPPORTED] event is received, it indicates that the connected device does not support advanced AVRCP functionality.

If the function DisableAdvancedAvrcpFeature() is called before the protocol stack is initialized, the advanced AVRCP function will be disabled.

**9. SPP**

**9.1 SPP Introduction**

Shanjing Bluetooth SPP function only serves as the DevB, that is, the Server side. It only supports active connection of Master devices such as mobile phones, computers and tablets. It does not support the power-on reconnection of SPP, nor does it support the interconnection of SPPS between two devices. The SPP function can only be used after being configured as enabled during the protocol stack initialization; otherwise, it is in an unenabled state.

The Bluetooth SPP itself is merely a data path and does not involve more data interaction protocols. If the user wishes to apply SPP to a specific scenario, they need to pre-define the relevant data interaction protocol by themselves and implement the corresponding functions on the host end and the speaker end (the device where MV BT runs). This document introduces the data processing methods and precautions for the SPP at the speaker end. The implementation methods on the host side are not within the scope described in this document. The SDK released by the Mountain View standard usually includes a functional sample APK that can run on the android system and match the standard SDK. Users can make any appropriate cuts and expansions on this basis.

**9.2 SPP Usage Method**

The entry point of the SPP code is defined as a function in the SDK release standard file spp\_app\_proc.c

DWORD UserProcSppData(VOID)；

This function is the sole entry point for SPP interaction. All related interface functions of SPP must be called under this entry point and cannot be called anywhere else. This function is handled by the protocol stack for callback processing. Therefore, in principle, all processing within this function must not be implemented in a way that obviously blocks for a long time. The return value of the function must comply with the following two rules:

1. When data is consumed (or retrieved) within the SPP, return 1 (non-zero).
2. Return a value of 0 when no data is consumed (or retrieved) within the SPP.

All interface functions of SPP are defined in the file [spp\_app\_proc.h], mainly including the following functions

void     SppFlushRecBuf(int16\_t Size);

uint16\_t  SppSendData(uint8\_t\* Buf, uint16\_t Size);

int16\_t   SppGetSendBufFreeSize(void);

uint16\_t  SppReadData(uint8\_t\* Buf, uint16\_t Len);

uint16\_t  SppGetRecBufDataSize(void);

Executing the function SppFlushRecBuf (int16\_t Size) will clear the data of the specified length in the receive buffer. If the data in the buffer is less than the expected value, the situation buffer will be returned. If the input parameter -1 indicates clearing all buffer data.

Executing the function SppSendData (uint8\_t\* Buf, uint16\_t Size) will send data to the send buffer. If the remaining space in the send buffer is less than the given amount of data, the function returns when the buffer is filled, but does not return the actual size of the sent data. This function only returns a value indicating whether the act of filling the buffer has been executed (regardless of the actual amount of data filled). Therefore, it is recommended to call the function SppGetSendBufFreeSize () before sending to obtain the current size of the sending buffer. Note: It is meaningless to wait in place for changes in the available amount of the current send buffer.

The execution function SppGetSendBufFreeSize (void) returns the current available size of the send buffer.

Execute the function SppReadData (uint8\_t\* Buf, uint16\_t Len) to read the contents of the specified data volume from the receive buffer. The return value represents the actual amount of data read. It is recommended that the amount of data in the current buffer be obtained by calling the function SppGetRecBufDataSize () before reading.

Note: It is meaningless to wait in place in the hope of reading more data.

Executing the function SppGetRecBufDataSize (void) can obtain the amount of data in the current receiving buffer.