User Guide

N32WB03x Data Transmission Routine Guide

Introduction

The purpose of this document is to enable users to quickly get familiar with the data transmission routine (rdtss) of the N32WB03x series Bluetooth SOC chips, in order to reduce the preparation time for development and lower the difficulty of development.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Table of Contents

1 Rountine Test Steps

1.1 Quick Test Steps：

1) Use the N32WB031_STB_V1.0 development board for testing.
2) Connect the SWD and USART1 on J10.
3) Connect the power supply to the MCU and NS-Link on J7 and J8.
4) Open the project: N32WB03x_SDK\projects\n32wb03x_EVAL\ble\rtdss\MDK-ARM\rtdss.uvprojx
5) Compile the project, connect to the USB port on the development board and download it.
6) Use the APP to scan for BLE devices and connect to the device "NS_RDTSS".
7) Sending data on the NS_Link virtual serial port will allow you to receive data in the APP.
8) Sending data on the APP will allow you to receive data on the NS_Link virtual serial port.

Note: You need to enable notifications for characteristic

0x02002EC78a0E73900xE111C20860270000 to receive data. Sending data is writing data to characteristic 0x01002EC78a0E73900xE111C20860270000.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

2 Data Transmission Service（RDTSS）

2.1 GATT Server

The Data Transmission Service (RDTSS) is used to receive and transmit BLE data. Its services and characteristics use 128-bit UUIDs as shown in the table below. It receives data sent from the APP through "write with no respond", and transmits data to the APP through "Notify".

RDTSS Service and Characteristic Table

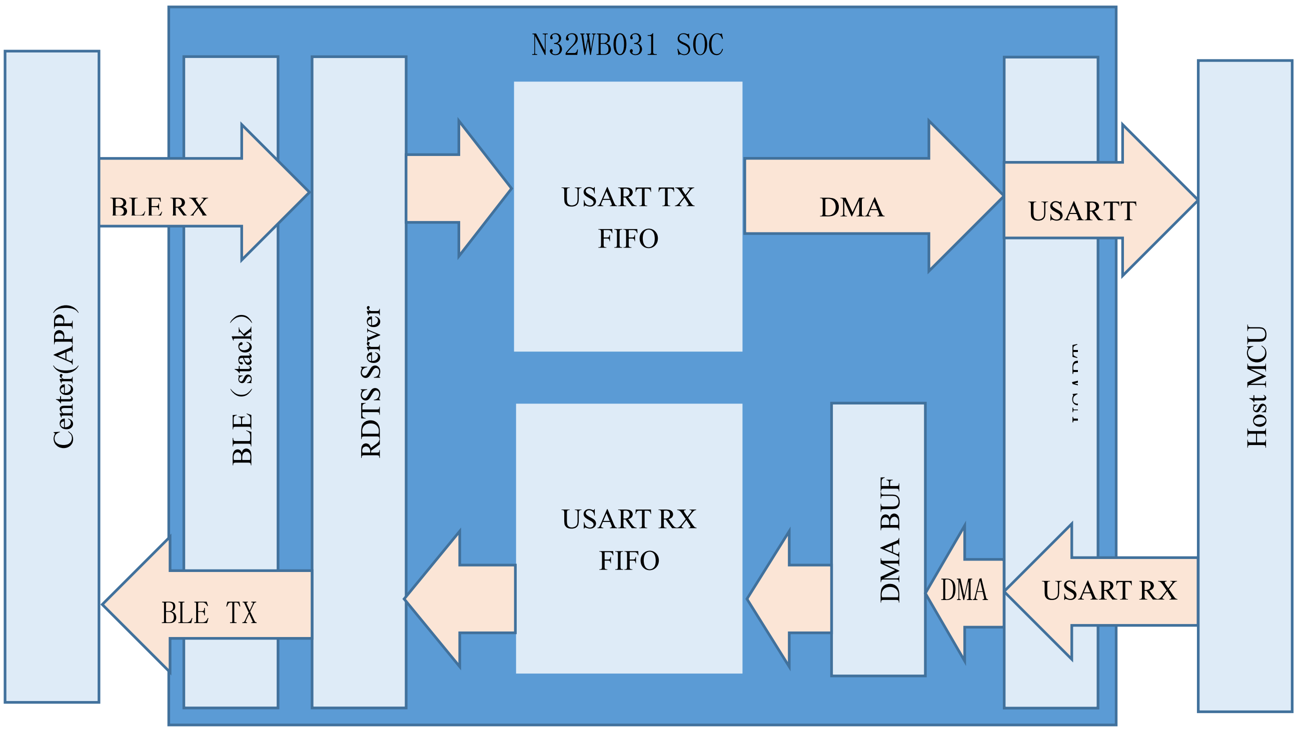
Service/Characteristic	UUID	Properties
RDTSS server	0x01102EC78a0E73900xE111C20860270000	ATT_CHAR_PROP_RD
RDTSS_RX	0x01002EC78a0E73900xE111C20860270000	ATT_CHAR_PROP_WR_NO_RESP
RDTSS_TX	0x02002EC78a0E73900xE111C20860270000	ATT_CHAR_PROP_NTF

2.2 Data Stream

It communicates with an external host via the serial port (USART). The USART uses hardware DMA and serial port interrupt service to receive and transmit data. In addition, the application layer also uses two FIFOs for serial port reception and BLE reception, respectively.

- Serial port reception: The serial port receives data through loopback mode DMA and uses three interrupt service functions (DMA_TC, DMA_HT, USART_IDLE) to call the usart_rx_check_in_irq function to check the data that the DMA has finished transferring, and transfers the received data to the FIFO (usart_rx_fifo_buf) through the app_usart_rx_data_fifo_enter function. See the BLE sending description for subsequent steps.
- Serial port sending: After the BLE data stream stops (no new data for 30ms), the app_usart_tx_process function is called to send the data in the FIFO (usart_tx_fifo_buf) out through the serial port DMA. After receiving the interrupt indicating DMA transmission completion, check the FIFO again to see if there is still data, and send it again if so, until all data has been sent.
- BLE receiving: Get the data sent from BLE in the callback function rdtss_val_write_ind_handler under the RDTSS_IDX_WRITE_VAL message, and call the app_usart_tx_fifo_enter function here to store the data in the FIFO (usart_tx_fifo_buf). See the serial port sending description for subsequent steps.
- BLE sending: After the data stream from the serial port stops (no new data for 10ms), the usart_forward_to_ble_loop function is called to check the FIFO (usart_rx_fifo_buf) and send the data to BLE through the rdtss_send_notify function. In the callback function rdtss_val_ntf_cfm_handler after each packet of data is sent, usart_forward_to_ble_loop is called again until all data is sent.

N32WB03x RDTSS Application



3 RDTSS Routine Project Directory Structure

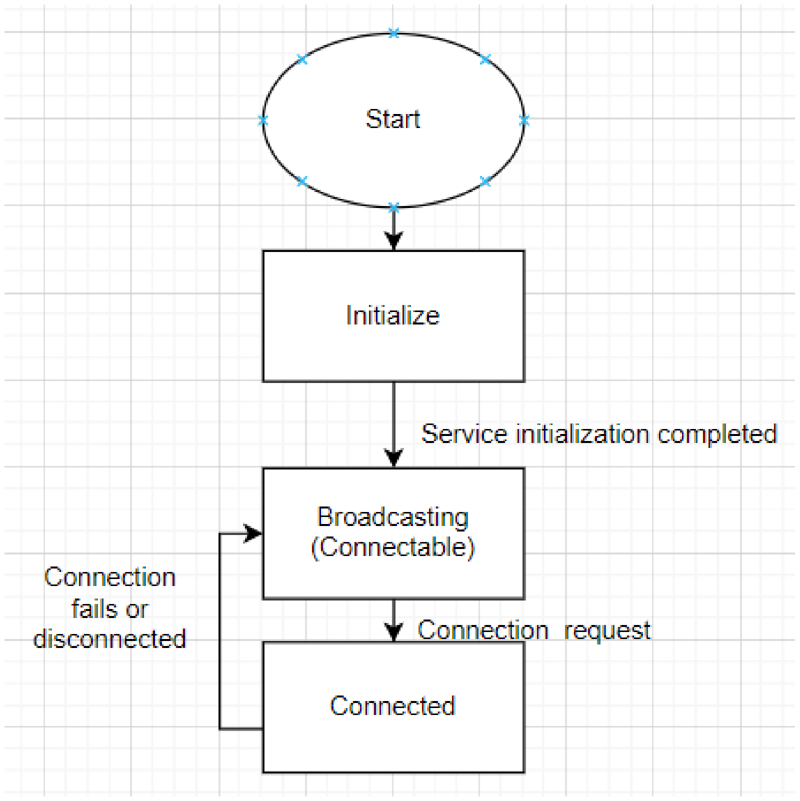
3.1 Project Target

- N32WB03x: Bluetooth project, without DFU configuration, generally only this target for ble projects.
- OTA_IMG_1: Bluetooth OTA project with configuration for Bank1 address
- OTA_IMG_2: Bluetooth OTA project with configuration for Bank2 address

3.2 Directory Structure

- STARTUP：Chip startup file
- CMSIS：Chip core configuration
- FWLB：Chip peripheral driver library
- BLE_STACK：Bluetooth BLE protocol stack
- BLE_PROFILE：Bluetooth BLE profile
- NS_DUF（可选）：Bluetooth OTA firmware upgrade related libraries
- Crypto (optional): Encryption related libraries used for Bluetooth OTA firmware upgrades
- NS_LIB: Bluetooth application related libraries
- BLE_APP: Bluetooth application code
    - app.c Bluetooth application code, such as broadcast and disconnect function implementation.
    - app_task.c Bluetooth message callback processing code, such as broadcast, connection and disconnect status callback messages.
    - app_dis.c Device information service application code
    - app_batt.c Battery level information service application code
    - app_rdtss.c Data transmission service application code
    - app_ns_ius.c OTA over-the-air upgrade service application code
    - app_sec_handlers.c Secure connection application code
    - app_bond_handlers.c Bonding application code
    - app_fsm.c Asynchronous state operation application code (such as storing bound devices to flash after disconnection)
- USER：User application code
    - main.c main function, user application code
    - app_usart.c Serial port driver and data cache application code
    - app_spi.c SPI driver and application code
- CONFIG：Configuration files
    - app_user_coonfig.h User configuration
    - app_user_callback_config.h User handler function configuration for Bluetooth message callbacks (currently implements the secure connection part)
- DOC：Documentation
    - readme.txt

4 Code State Machine


Start
Initialize
Service initialization completed
Broadcasting (Connectable)
Connection request
Connected
Connection fails or disconnected

Status	Description
Start	Chip system initialization, such as clocks, then enter the main function.
Initialize	Initialize the Bluetooth protocol stack, initialize the Bluetooth application task (app_task), initialize Bluetooth services (profiles), initialize broadcasting, and initialize user code (such as peripherals).
Broadcasting	Broadcasting status, can be discovered by mobile phones and other central devices. Since there are no high-speed peripherals that need to run continuously, the chip will enter sleep mode when there are no tasks (no events for the Bluetooth protocol stack and user code to handle) to reduce power consumption.
Connected	Connection completed, can receive and transmit Bluetooth data. Since the high-speed peripheral USART needs to work continuously to receive data, we have disabled the chip from entering sleep mode.

5 Key Code Description

5.1 app_user_config.h

The following are defined in macro form in this file:

- Bluetooth name
- Bluetooth MAC address
- Broadcast interval
- Broadcast timeout
- Broadcast data packet
- Broadcast response data packet
- Secure encryption connection parameters
- GATT service enable
- ns_log library enable and configuration
- ns_timer library enable configuration

5.2 gapc_connection_req_ind_handler

Bluetooth connection completion message processing, such as connection status indication.

5.3 gapc_disconnect_ind_handler

Bluetooth disconnection message processing . Broadcasting will be re-enabled here.

5.4 rdtss_val_write_ind_handler

Data transmission service write callback function, including notification enable message callback and BLE downlink data callback, i.e. chip receives Bluetooth data callback.

5.5 rdtss_send_notify

Data transmission service notify data sending function. The chip sends data to the APP end through this function

5.6 rdtss_val_ntf_cfm_handler

Dara transmission service notify data sending confirmation callback indicating that the application layer has correctly sent the data to the Bluetooth protocol stack, and the Bluetooth protocol statck will send the data to the APP end

5.7 app_sleep_prepare_proc

Processing function before entering sleep. For example, high-speed peripherals can be disabled and IOs can be set to analog input to achieve the lowest sleep power consumption. Note that time-consuming code cannot be implemented in this function, because the chip frequently enters and exits sleep mode. Too much time here will affect functionality.

5.8 app_sleep_resume_proc

Processing function after sleep wake-up. For example, peripherals can be initialized. After sleep wake-up, peripherals using HCLK, APB1, and APB2 need to be reinitialized. Do not occupy too much time, and if interrupts are pending during sleep, the interrupt service function will be entered first before coming here.

5.9 ns_sleep_lock_acquire

Sleep mode lock request. After calling, the chip entering sleep mode will be disabled.

5.10 ns_sleep_lock_release

Sleep mode lock release. When all locks are released, the chip will enter sleep mode at the appropriate time.

5.11 app_usart_dma_enable

Serial port enable or disable. Enabling includes initializing the serial port and starting DMA reception. Since the serial port needs to use a high frequency clock, we need the serial port to stay in a state where it can continuously receive data. Therefore, after enabling the serial port, ns_sleep_lock_acquire will be called to prohibit the chip from entering sleep mode.

5.12 app_usart_rx_data_fifo_enter

Store data in the serial receive FIFO cache area. Called in the interrupt service function to store DMA data in the serial receive FIFO cache area.

5.13 app_usart_tx_fifo_enter

Store data in the serial transmit FIFO cache area. Called in the Bluetooth receive data callback to store Bluetooth data in the serial transmit FIFO cache area.

5.14 ModuleIrqRemoval

Interrupt service function removal.

5.15 ModuleIrqRegister

Interrupt service function registration. Note that in the case of using the Bluetooth protocol stack, the interrupt service function can only be added in this way instead of directly using the interrupt service function declared in startup_n32wb03x.s. Before registration, first call the ModuleIrqRemoval function to clear this interrupt service function and then register it.

6 Bluetooth Program Writing Suggestions

- Do not continuously execute time-consuming code, which will hinder Bluetooth message processing. It is recommended to minimize user code as much as possible, with each task message or polling only occupying a small amount of time.
- Do not execute interrupt service functions for too long to avoid hindering Bluetooth message processing. It is recommended to implement logical code in tasks or polling through pending messages, timers or flags.
- Do not call hardware peripherals in interrupt service functions. If an interrupt event is pending after sleep, the interrupt service function will be called before app_sleep_resume_proc. The peripheral has not been initialized after wake-up, so calling hardware peripherals in interrupt service functions can cause errors because the peripherals are not initialized.
- For projects containing Bluetooth, interrupt service functions need to be registered through the ModuleIrqRegister function, and the interrupt priorities of user code can only use 2 and 3 (the Bluetooth protocol stack uses 0 and 1).
- Note that when entering the low power sleep mode, high-speed peripherals (such as USART) will be turned off and need to be initialized by the program after wake-up before they can be used.
- It is recommended that the user's logic code be implemented in the task callback functions, which can be message event callbacks or timed task callbacks.

7 Version History

Version	Date	Changes
V1.0	2020.08.05	Initial version

8 Disclaimer

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg