M031BT Bluetooth Electric Toothbrush

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This sample code uses M031BT to do the Bluetooth electric toothbrush solution.
BSP Version	M031_Series_BSP_CMSIS_V3.05.000
Hardware	Nuvoton_M031BT bluetoothbrush solution board

1. Overview

This sample code uses M031BT to achieve the Bluetooth electric toothbrush solution.

The M031BT is an Arm Cortex-M0 based microcontroller (MCU) with a maximum clock speed of 48 MHz. It features two PWM modules and 16 12-bit ADC channels. Each PWM module has 6 channels and can be used to do the motor control to drive and adjust the vibration frequency of the electric toothbrush. The maximum sampling rate of ADC is up to 2 MSPS and can provide the accurate voltage measurement in time. Specifically, the M031BT uses the QFN48 package with only 5mm x 5mm area to facilitate the system hardware design.

1.1 Principle

The M031BT is a 2-in-1 microcontroller. It integrates the general MCU features and Bluetooth transparent data transmission function. To Combine with the APP developed by various manufactures, the electric Bluetooth toothbrush that embeds with the M031BT can record the personal health index information, oral health and brushing habits. Other than Bluetooth transmission, the M031BT that is integrated with rich peripherals can also provide the main functions in the electric toothbrush, including motor control, light display, battery detection and so on.

Therefore, the M031BT can simplify the hardware design to reduce the cost and speed up the APP development.

The development package provides the Android system mobile phone APP (NuToothbrush v1.0.7.apk) for developers to demonstrate, and provides the hardware PCB layout and circuit diagram of the solution. The PCB version is PADS9.5, and the circuit diagram version is orCAD 16.3.

There are 6 LED lights and 1 button on the electric toothbrush, and the main function is as follows:

- There is a set of 4 LED lights (corresponding to 4 gear displays);
- There is a set of 2 LED lights, one for power indication (flashing when battery power is low), the other for Bluetooth connection status (the light is On when the BT is connected, and the light is Off when the BT is disconnected);
- There is one button (used for power on-off, gear switching, and other controls);
- Set the brushing intensity and time of each gear through the APP, and after the setting is completed, it can run immediately, and the recent toothbrush usage records can also be restored.

1.2 ICP Download

Connect to the ICP tool and download the firmware to the chip.

Download M031BT_NuToothbrush.bin and configure the chip, as shown in Figure 1-1 and Figure 1-2.

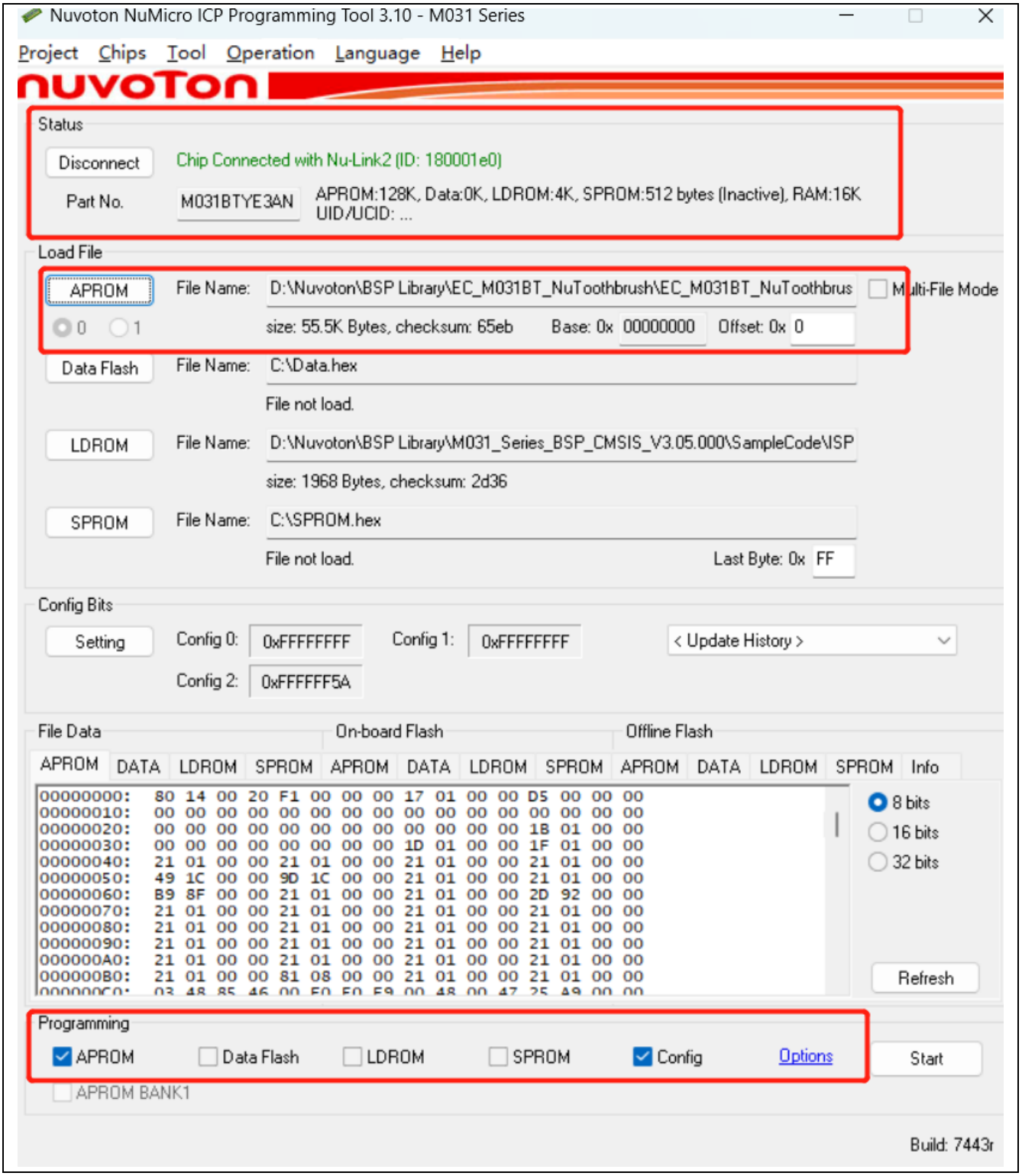

Figure 1-1 ICP Configuration

Chip settings are shown as follows:

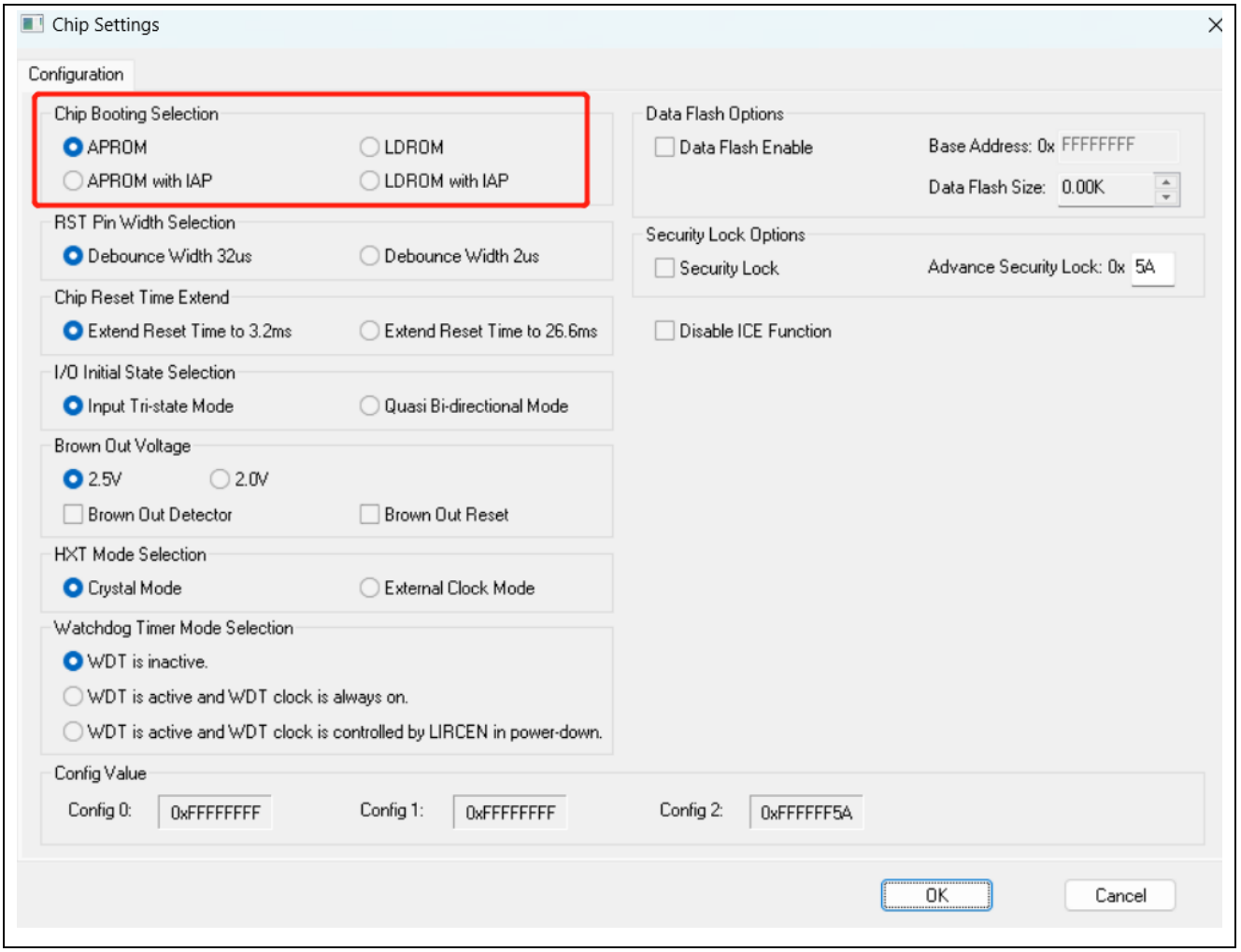

Figure 1-2 Chip Settings

1.3 Demo Result

The result of the linkage between the code and the APP in this example is shown in Figure 1-3 and the following is the status of the Bluetooth electric toothbrush and the Bluetooth APP of the mobile phone.

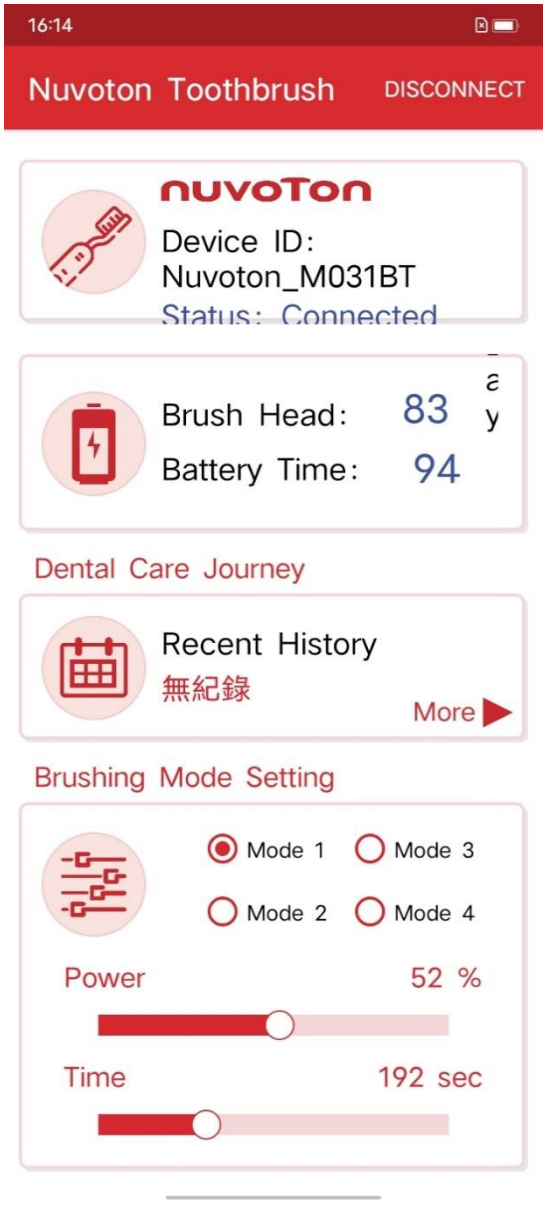

Figure 1-3 Bluetooth Electric Toothbrush and Mobile Phone Bluetooth APP Connected State

Figure 1-4 is the recent use record of the electric toothbrush. Click on the recent use record to view the recent use time record of the toothbrush.

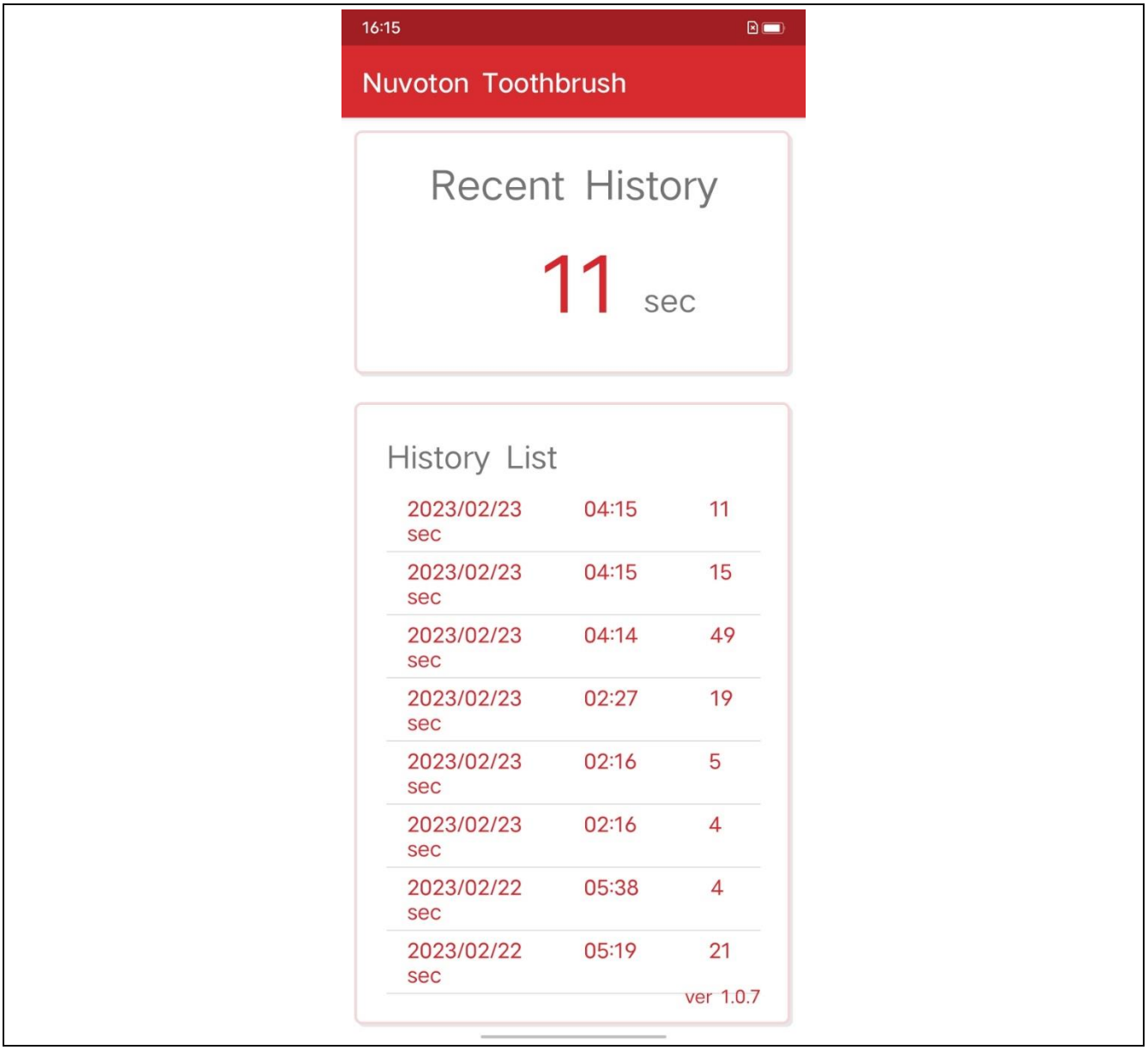

Figure 1-4 Recent Use of Electric Toothbrushes

2. Code Description

Key processing function:

```
void KeyPorc(void)
{
    if(KeyCode == KEY_CHANGE)
    {
         KeyCode = KEY_NONE;
         if(power)
         {
              if(sta<0x5)
              {
                   if(sta==0x4)sta=0;
                   sta++;
                   onlyflag=1;
              }
              BT_Switch_Channel();
         }
    }
    else if(KeyCode == KEY_VERSION)
    {
         KeyCode = KEY_NONE;
         PowerON();
    }
     Motor_Change();
}
```

Motor gear switching function:

```
void Motor_Change(void)
{
    if(power&&onlyflag)
    {
         switch(sta)
         {
              case 0x1:Display_Level_1();onlyflag=0;break;
              case 0x2:Display_Level_2();onlyflag=0;break;
              case 0x3:Display_Level_3();onlyflag=0;break;
              case 0x4:Display_Level_4();onlyflag=0;break;
              default:break;
         }
    }
}
```

Sleep and wake-up functions of BLE and MCU:

```
/* Run user application */
BleApp_Main();

if (dsleep_enabled)
{
    status = setRF_WakeUpFromDeepSleep();
    if (status == BLESTACK_STATUS_SUCCESS)
    {
```

```
            /* Restore BLE settings after wakeup */
            Ble_Slave_Init();
            dsleep_enabled = 0;
            atcmd_set_adven(0);
        }
    }
    if (((UART0->FIFOSTS & (UART_FIFOSTS_RXIDLE_Msk | UART_FIFOSTS_TXEMPTYF_Msk)) ==
    (UART_FIFOSTS_RXIDLE_Msk | UART_FIFOSTS_TXEMPTYF_Msk)) && (tx_data_transmit_enable ==
    0)&&(BAT_CHRG==1)&&(wakeflag==0))
    {
        wakeflag=1;
        /* System enter Power Down mode & wait interrupt event. */
        setMCU_SystemPowerDown();
    }
```

BLE data transmission:

```
//Send out RF data
void trspx_send(uint8_t *data, uint16_t len)
{
    int i = 0;

    tx_data_transmit_enable = 1;

    /* if there is data pending, still send old data and skip new data */
    if (notifyPending == 0)
    {
        txDataLength = len;

        //put UART data in txDataBuffer
        for (i = 0; i < len; i++)
        {
            txDataBuffer[i] = data[i];
        }
    }
}


void handle_AppLink0_TRSPP(void)
{

    BleStackStatus status;

    if (bleProfile_link0_info.bleState == STATE_BLE_STANDBY)
    {
        LED_BT = 1;
        power_on_flag = 1;
        // reset preferred MTU size
        setBLEGATT_PreferredMtuSize(CONN_TRSP_LINK_HOSTID, DEFAULT_MTU);

        // reset service data
        BleService_GATT_DataInit(&(bleProfile_link0_info.serviceGATT_info_s.data));
        BleService_UDF01S_DataInit(&(bleProfile_link0_info.serviceUDF01S_info_s.data));

        // enable advertisement
```

```
        status = Ble_AdvStart(CONN_TRSP_LINK_HOSTID);
        LED_BT = 1;

        if (status == BLESTACK_STATUS_SUCCESS)
        {
            bleProfile_link0_info.bleState = STATE_BLE_ADVERTISING;
            LED_BT = 1;
        }
    }
    else if (bleProfile_link0_info.bleState == STATE_BLE_CONNECTION)
    {
        if(power)
        {
            LED_BT = 0;

            if(power_on_flag > 0)
            {
                channel_count = 3;
                Report_BatteryLevel = 1;
                power_on_flag = 0;
            }
        }
        else
        {
            bleProfile_link0_info.bleState = STATE_BLE_STANDBY;
            power_on_flag = 1;
        }

        if ((tx_data_transmit_enable == 1) || (notifyPending == 1))
        {
            tx_data_transmit_enable = 0;

            // Send out data by RF
            /* Be careful that the client should enable NOTIFY, then the server can start
             sending notification data */
            status = setUDF01S_ServerDataSend(bleProfile_link0_info.hostId,

                      bleProfile_link0_info.serviceUDF01S_info_s.data.udatn01_cccd,

                      bleProfile_link0_info.serviceUDF01S_info_s.handles.hdl_udatn01,
                                                txDataBuffer,
                                                txDataLength);

            if (status == BLESTACK_STATUS_SUCCESS)
            {
                notifyPending = 0;

                if(Brush_Time_Record)
                    Brush_Time_Record = 0;
            }
            else
            {
                notifyPending = 1;
            }
        }
    }
```

```
    else
        power_on_flag = 1;
}
```

BLE AT command handler function:

```
AtCmdStatus atcmd_handler(char *data) /* AT command functions */
{
    AtCmdStatus ret = ATCMD_FAIL;
    uint8_t i;
    char pcCopy[12], pcCopy_t[3], pcCopy_p[3];
    strcpy(pcCopy, data);

    if (pcCopy[0] == 'M')
    {
        if (pcCopy[1] == '1')
        {
            if (pcCopy[2] == 'P')
            {
                pcCopy_p[0] = pcCopy[3];
                pcCopy_p[1] = pcCopy[4];
                pcCopy_p[2] = pcCopy[5];
            }
            else if(pcCopy[2] == 'R')
            {
                set_mode_p[0] = 0x127;
                set_mode_t[0] = 0x78;
            }

            if(pcCopy[6] == 'T')
            {
                pcCopy_t[0] = pcCopy[7];
                pcCopy_t[1] = pcCopy[8];
                pcCopy_t[2] = pcCopy[9];
                sscanf(pcCopy_p, "%d", &set_p[0]);
                sscanf(pcCopy_t, "%d", &set_t[0]);
                set_mode_p[0] = (set_p[0] * 3);
                set_mode_t[0] = set_t[0];
            }

            sta = 0x1;
            Display_Level_1();
        }
        else if (pcCopy[1] == '2')
        {
            if (pcCopy[2] == 'P')
            {
                pcCopy_p[0] = pcCopy[3];
                pcCopy_p[1] = pcCopy[4];
                pcCopy_p[2] = pcCopy[5];
            }
            else if(pcCopy[2] == 'R')
            {
                set_mode_p[1] = 0x116;
```

```
                set_mode_t[1] = 0x5A;
            }

            if(pcCopy[6] == 'T')
            {
                pcCopy_t[0] = pcCopy[7];
                pcCopy_t[1] = pcCopy[8];
                pcCopy_t[2] = pcCopy[9];
                sscanf(pcCopy_p, "%d", &set_p[0]);
                sscanf(pcCopy_t, "%d", &set_t[0]);
                set_mode_p[1] = (set_p[0] * 3);
                set_mode_t[1] = set_t[0];
            }

            sta = 0x2;
            Display_Level_2();
        }
        else if (pcCopy[1] == '3')
        {
            if (pcCopy[2] == 'P')
            {
                pcCopy_p[0] = pcCopy[3];
                pcCopy_p[1] = pcCopy[4];
                pcCopy_p[2] = pcCopy[5];
            }
            else if(pcCopy[2] == 'R')
            {
                set_mode_p[2] = 0x11F;
                set_mode_t[2] = 0x3C;
            }

            if(pcCopy[6] == 'T')
            {
                pcCopy_t[0] = pcCopy[7];
                pcCopy_t[1] = pcCopy[8];
                pcCopy_t[2] = pcCopy[9];
                sscanf(pcCopy_p, "%d", &set_p[0]);
                sscanf(pcCopy_t, "%d", &set_t[0]);
                set_mode_p[2] = (set_p[0] * 3);
                set_mode_t[2] = set_t[0];
            }

            sta = 0x3;
            Display_Level_3();
        }
        else if (pcCopy[1] == '4')
        {
            if (pcCopy[2] == 'P')
            {
                pcCopy_p[0] = pcCopy[3];
                pcCopy_p[1] = pcCopy[4];
                pcCopy_p[2] = pcCopy[5];
            }
            else if(pcCopy[2] == 'R')
            {
                set_mode_p[3] = 0x10F;
```

```
                set_mode_t[3] = 0x1E;
            }

            if(pcCopy[6] == 'T')
            {
                pcCopy_t[0] = pcCopy[7];
                pcCopy_t[1] = pcCopy[8];
                pcCopy_t[2] = pcCopy[9];
                sscanf(pcCopy_p, "%d", &set_p[0]);
                sscanf(pcCopy_t, "%d", &set_t[0]);
                set_mode_p[3] = (set_p[0] * 3);
                set_mode_t[3] = set_t[0];
            }

            sta = 0x4;
            Display_Level_4();
        }
    }
    else if ((pcCopy[0] == 'U') && (pcCopy[1] == 'T'))
    {
        sprintf((char*)use_record, "%d", Record_T[1]);

        if(Record_T[1] < 100)
        {
            use_record[2] = use_record[1];
            use_record[1] = use_record[0];
            use_record[0] = '0';

            if(Record_T[1] < 10)
            {
                use_record[2] = use_record[1];
                use_record[1] = '0';
            }
        }

        for(i = 0; i < 3; i++)
        {
            BT_record[i + 1] = use_record[i];
        }

        if(power)
            BT_record[0] = 'O';
        else
            BT_record[0] = 'D';

        trspx_send(BT_record, 0x4);
    }
    else if ((pcCopy[0] == 'I') && (pcCopy[1] == 'D'))
    {
        BLE_Addr_Param bleAddrParam;
        uint8_t i, j, k = 0;
        BT_ID_addr[0] = 'A';
        getBLE_BleDeviceAddr(&bleAddrParam);

        for (i = 0; i < 6; i++)
        {
```

```
            sprintf((char*)BT_ID, "%02x", bleAddrParam.addr[i]);

            for(j = 0; j < 2; j++)
            {
                k = k + 1;
                BT_ID_addr[k] = BT_ID[j];
            }
        }

        trspx_send(BT_ID_addr, 13);
    }

    return ret;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.05.000
- IDE version
  - Keil uVersion 5.28

3.2 Hardware Requirements

- Circuit components
  - Nuvoton_M031BT Bluetooth electric toothbrush solution board
- Pin Connect

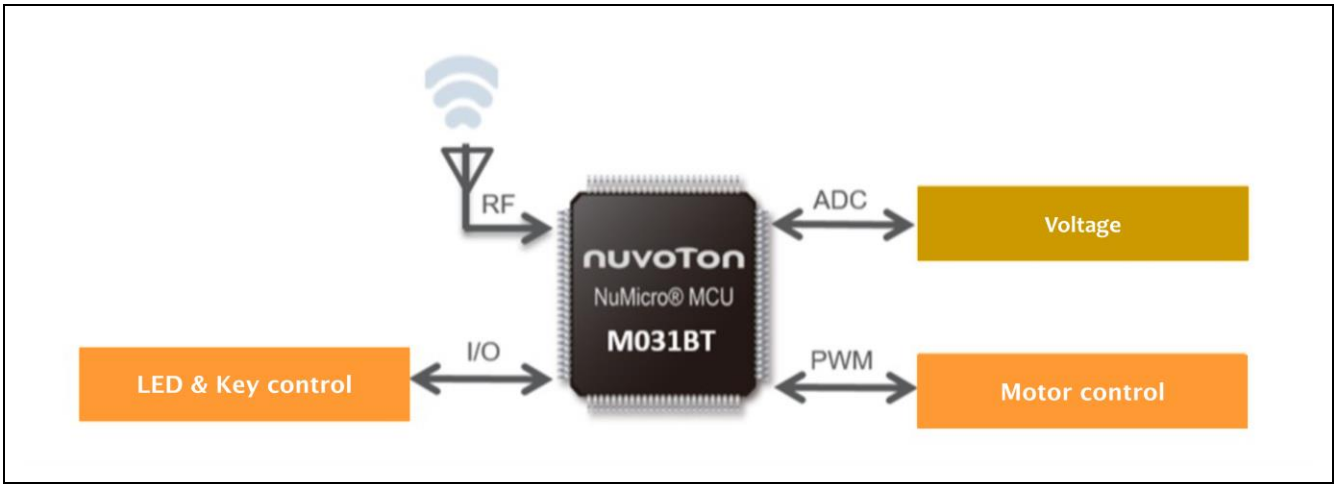

Figure 3-1 Nuvoton_M031BT Bluetooth Electric Toothbrush Schematic Diagram

4. Directory Information

The directory structure is shown below.

EC_M031BT_NuToothbrush_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code
App	Android APP file. Its website is: https://play.google.com/store/apps/details?id=com.nuvoton.nutoothbrush
Hardware	Hardware schematics with PCB files
M031BT_NuToothbrush	Keil project source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and *M031BT_NuToothbrush.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.02.21	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.