Bridging Transmission of UART and CAN

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code demonstrates the bridging transmission between UART and CAN using the M0A23 series microcontroller.
BSP Version	M0A21_Series_BSP_CMSIS_V3.03.000
Hardware	NuMaker-M0A23EC V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This example code demonstrates how to achieve bridging transmission between UART and CAN on the M0A23 series platform.

A relevant application scenario is in the automotive electronics field, where the system needs to integrate UART sensors but must transmit data using the CAN bus. In such cases, UART-to-CAN bridging technology can be employed to enable data communication and ensure stable transmission.

1.1 Principle

Since the transmission principles and data formats of UART and CAN are different, this section focuses on the conversion of transmission formats and the data flow between UART and CAN.

CAN transmits data in “frame”, while UART transmits data in bytes. To ensure compatibility, the data received via UART must be formatted to match the CAN frame structure. The mapping relationship is defined in Table 1-1. Unlike CAN, which has clearly defined frame boundaries, UART lacks an explicit framing mechanism. Therefore, a data header is added to prevent potential data parsing errors.

UART Bytes	UART Data	Contents	Example (in Hex)
0-1	0x55, 0xAA	Start header	55, AA
2-5	ID1, ID2, ID3, ID4	CAN ID	00, 00, 01, 33
6	Length	Data length Note: The length cannot be 0.	04
7-N	DataN	CAN Data	11, 22, 33, 44

Table 1-1 Data format

Once UART receives the data and verifies that it has been correctly parsed, it encapsulates the data into a CAN frame and transmits it using CAN_Transmit. The process and decision flow are illustrated in Figure 1-1.

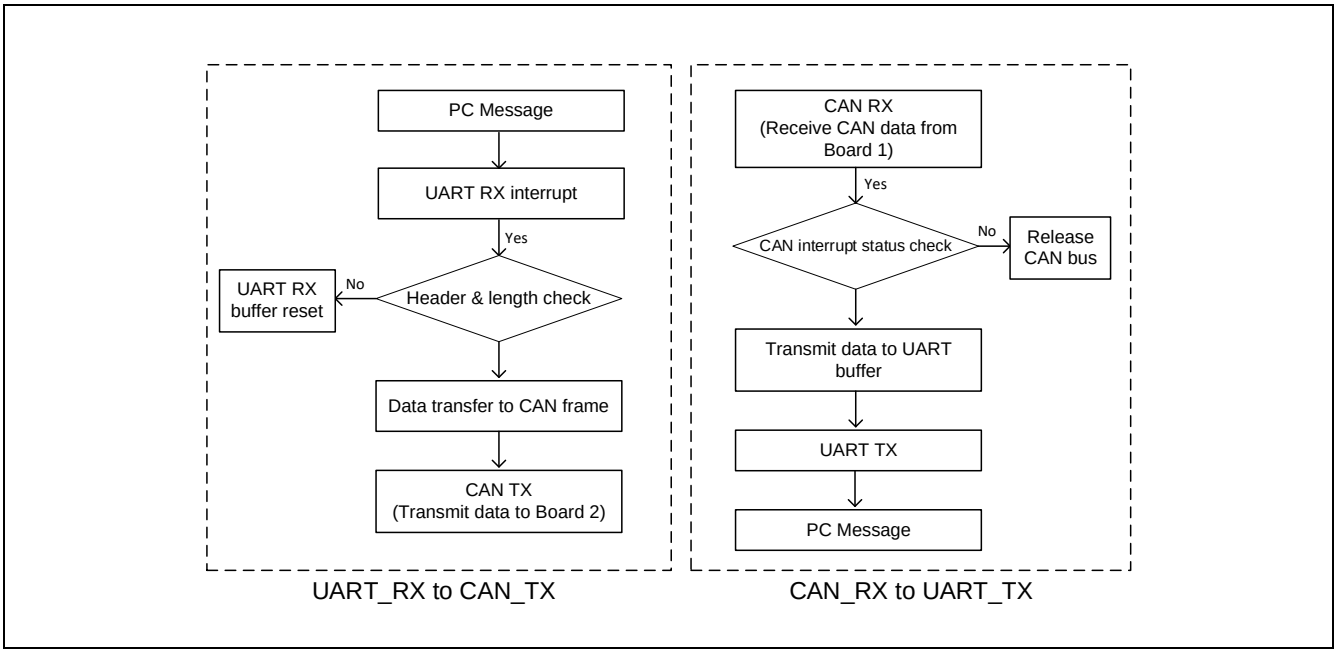

Figure 1-1 Data Reception and Conversion Process

For example, if the data received by the UART as shown in the "Example" column in Table 1-1 is converted into a CAN packet, the transmitted CAN data will be formatted as shown in Table 1-2.

Frame Type	ID Type	ID	DLC	Data
CAN_Data_FRAME	CAN_STD_ID CAN_EXT_ID	0x00000133	4	11,22,33,44

Table 1-2 CAN Package

1.2 Demo Result

The compilation and programming results of this example code are shown in Figure 1-2.

Please refer to the circuit diagram in Section 3.2 for the hardware connection method. For software tools, a serial port monitoring software that supports Hex transmission and reception is required. Use two NuMaker-M0A23EC boards, and enter the appropriate Hex string in the serial port monitoring software on one of the boards. As shown in the example string in Table 1-1, the other serial port should receive the specified string successfully.

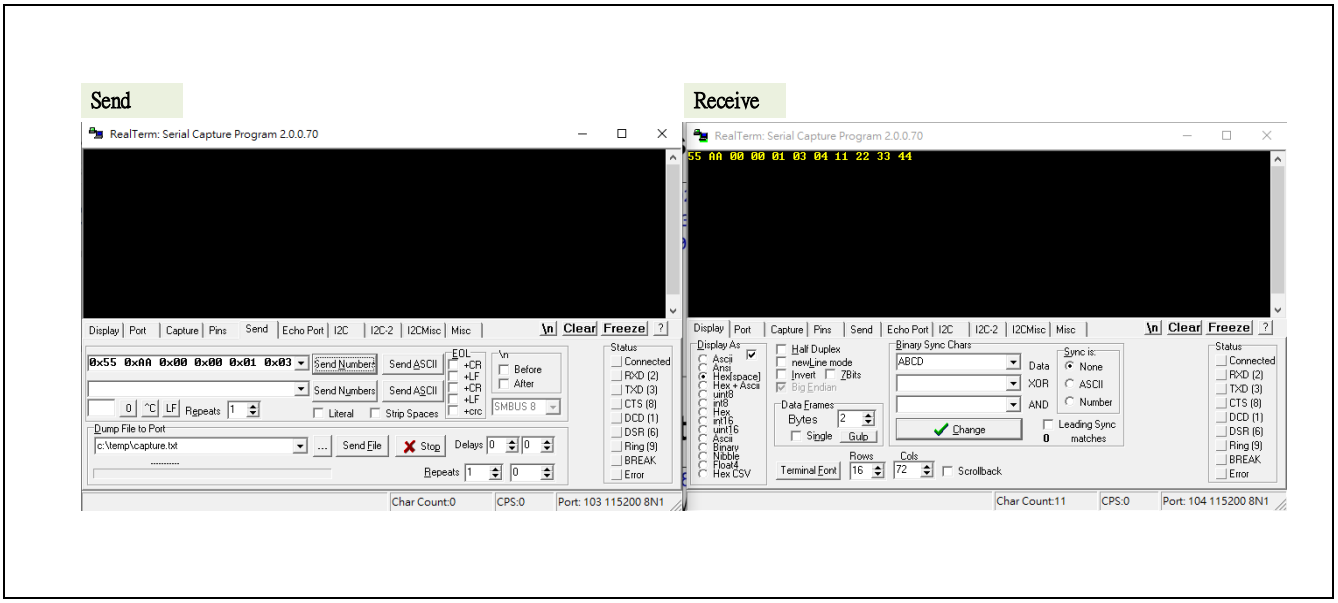

Figure 1-2 Compilation and Programming Results

2. Code Description

UART receives data using the RDA interrupt and stores it in the UART ring buffer.

```
void UART0_IRQHandler(void)
{
    uint32_t u32IntStatus;
    int32_t size;
    uint8_t loc_data[1];
    u32IntStatus = UART0->INTSTS;

    if((u32IntStatus & UART_INTSTS_RDAINT_Msk))
    {
        /* Get all the input characters */
        while(UART_GET_RX_EMPTY(UART0) == 0)
        {
            loc_data[0] = UART_READ(UART0);
            lwrb_write(&lw_uart_rx_buff, loc_data, 1);
        }
    }

    /* Check if Tx is working */
    if((UART0->INTEN & UART_INTEN_THREIEN_Msk) != 0)
    {
        if(u32IntStatus & UART_INTSTS_THREIF_Msk)
        {
            if((lw_uart_tx_buff.w - lw_uart_tx_buff.r) != 0)
            {
                /* Fill the Tx FIFO */
                size = (lw_uart_tx_buff.w - lw_uart_tx_buff.r);
                if(size >= TX_FIFO_SIZE)
                {
                    size = TX_FIFO_SIZE;
                }
                while(size)
                {
                    lwrb_read(&lw_uart_tx_buff, loc_data, 1);
                    while(UART0->FIFOSTS & UART_FIFOSTS_TXFULL_Msk) {};
                    UART0->DAT = loc_data[0];
                    size--;
                }
            }
            else
            {
                if((lw_uart_tx_buff.w - lw_uart_tx_buff.r) == 0)
                {
                    /* No more data, just stop Tx (Stop work) */
                    UART_DISABLE_INT(UART0, UART_INTEN_THREIEN_Msk);
                    lwrb_reset(&lw_uart_tx_buff);
                }
            }
        }
    }
}
```

In the while loop of *main.c*, the program checks whether there is data in the UART buffer and analyzes whether the received UART data format meets the defined specifications.

```
        /* analyze UART data */
        if((lw_uart_rx_buff.w - lw_uart_rx_buff.r) > 0)
        {
            __disable_irq();
            lwrb_read(&lw_uart_rx_buff, loc_data_m, 1);
            gUartReciveGroup[u8UART0_RX_Count] = loc_data_m[0];

            /* compare start byte, and the start byte should be 0x55 */
            if(u8UART0_RX_Count == 0 && gUartReciveGroup[u8UART0_RX_Count] != 0x55)
            {
                dbg_printf("The first byte is not 0x55.\n");
            }
            else
            {
                u8UART0_RX_Count++;
                /*check the format
                1. The first 2 bytes are 0x55 0xAA
                2. The data length should not be 0
                3. Data received finish, when the length is equal to gUartReciveGroup[6]+7
                */
                if( (gUartReciveGroup[0] == 0x55) && (gUartReciveGroup[1] == 0xaa) &&
                    (gUartReciveGroup[6] != 0) &&
                    ((u8UART0_RX_Count >= (gUartReciveGroup[6] + 7))) )
                {
                    u8UART0_RX_Flag = 1;
                }
            }
            if((lw_uart_rx_buff.w - lw_uart_rx_buff.r) == 0)
            {
                lwrb_reset(&lw_uart_rx_buff);
            }
            __enable_irq();
        }
```

If the data meets the defined specifications, it will be encapsulated into the CAN frame format and transmission will begin.

```
        /* Start CAN transmit */
        if(u8UART0_RX_Flag == 1)
        {
            /* The format of UartReceiveGroup is
                55 AA ID1 ID2 ID3 ID4 Length Data1 Data2 ...
            */
            uint32_t id = 0;
            uint32_t count = 0;

            /* Get device address */
            id = gUartReciveGroup[2] << 24;
            id += gUartReciveGroup[3] << 16;
            id += gUartReciveGroup[4] << 8;
            id += gUartReciveGroup[5];
            STR_CANMSG_T tMsg;

            /* Send a 11-bits message */
            tMsg.FrameType = CAN_DATA_FRAME;
```

```
#ifdef CAN_STD_FRAME
            tMsg.IdType = CAN_STD_ID;
#else
            tMsg.IdType = CAN_EXT_ID;
#endif
            tMsg.Id = id;
            tMsg.DLC = gUartReciveGroup[6];

            for(count = 0; count < gUartReciveGroup[6]; count++)
                tMsg.Data[count] = gUartReciveGroup[7 + count];

            /* Configure Msg RAM and send the Msg in the RAM */
            if(CAN_Transmit(tCAN, MSG(1), &tMsg) == FALSE)
            {
                dbg_printf("Set Tx Msg Object failed\n");
            }

            /* clear buffer */
            for(count = 0; count < 7; count++)
                gUartReciveGroup[count] = 0;

            u8UART0_RX_Count = 0;

            /* clear flag */
            u8UART0_RX_Flag = 0;
    }
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M0A21_Series_BSP_CMSIS_V3.03.000
- IDE version
  - Keil uVersion 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M0A23EC V1.0
- Pin Connect
  - Connect the CAN_H and CAN_L pins of the two NuMaker-M0A23EC evaluation boards together. Then, use UART to receive and transmit data through the PC to display the execution results of the example code. On the NuMaker-M0A23EC, you can directly connect to the PC via the Nu-Link2-Me USB port (ICEJ3) on the board and set the VCOM switch's RX and TX to the "ON" position.

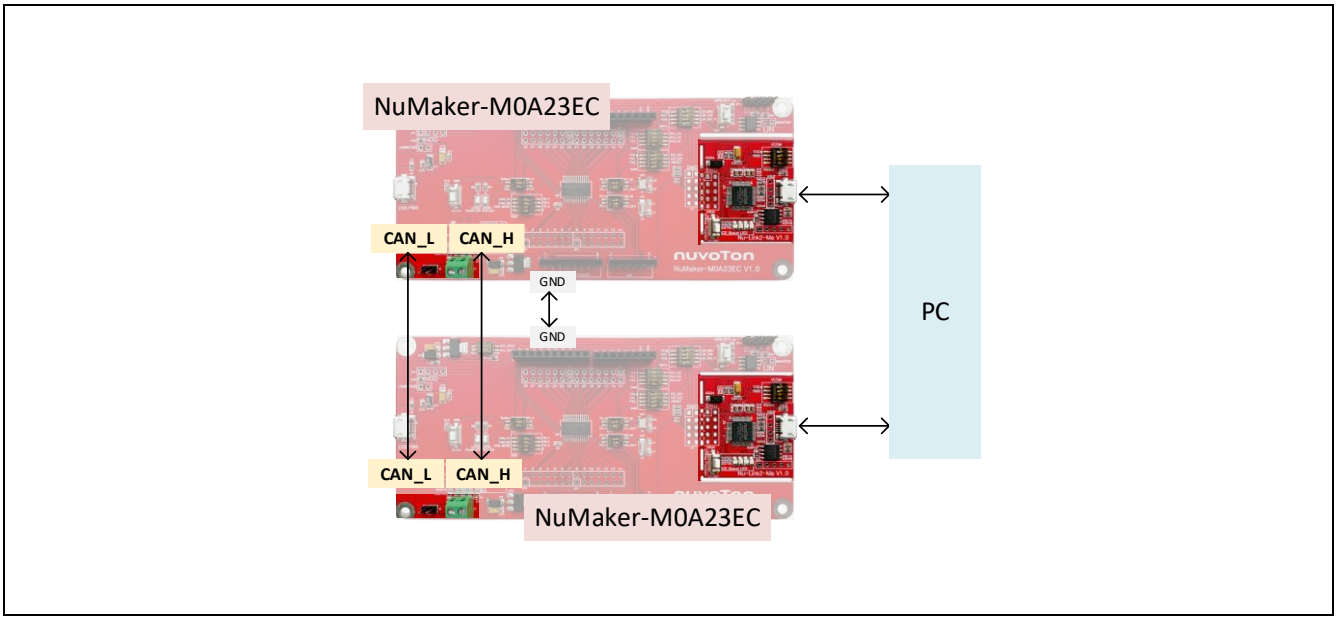

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

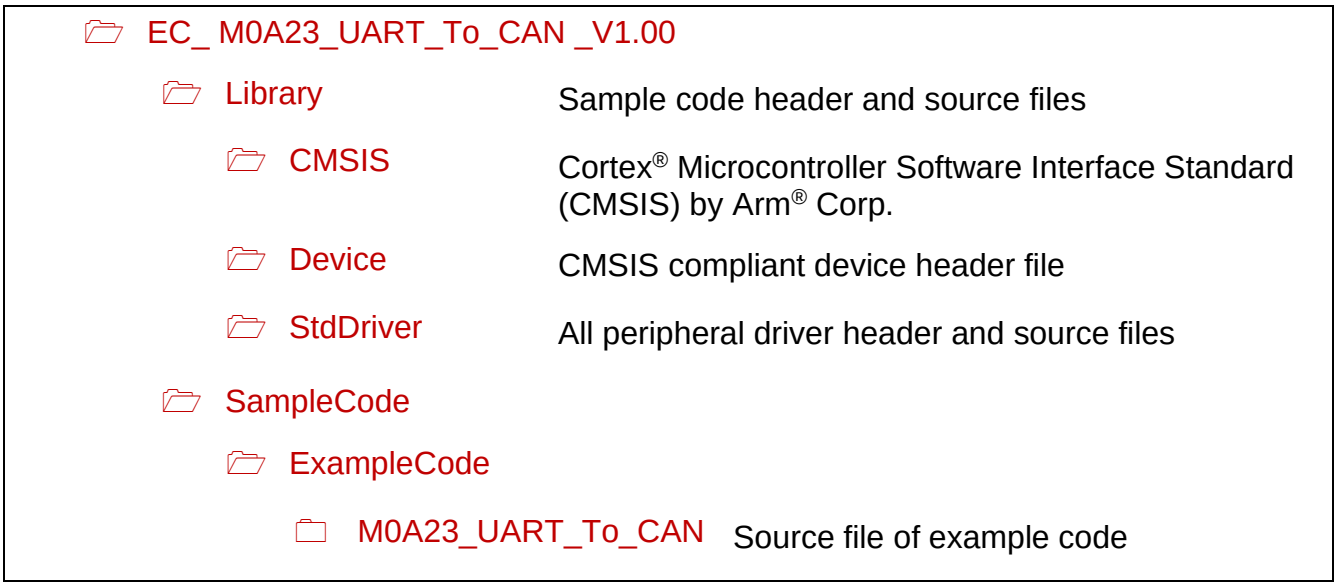

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M0A23_UART_To_CAN.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.02.28	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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.