Using NUC131 CAN to Adjusting Baud Rate Automatically

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code receives data on the CAN bus by periodically changing the baud rate of NUC131 CAN to match the baud rate of the CAN bus.
BSP Version	NUC131_Series_BSP_CMSIS_V3.00.005
Hardware	NuTiny-EVB-NUC131-LQFP64 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 sample code contains two projects, "CAN_TX_RX" and "CAN_Auto_Adjust_BPS". The "CAN_TX_RX" is a CAN sending and receiving demo, which is written to facilitate testing the function of "CAN_Auto_Adjust_BPS" demo.

The "CAN_Auto_Adjust_BPS" demo receives data on the CAN bus by periodically changing the baud rate of NUC131 CAN until it can normally receive data on the CAN bus, so as to match the baud rate of the CAN bus. After matching the baud rate of the CAN bus, packets will be periodically sent to the CAN bus. This demo can detect whether the baud rate range is 50 Kbps~1 Mbps, and step is 50 Kbps.

1.1 Principle

In this application, the "CAN_Auto_Adjust_BPS" demo will first set the CAN of NUC131 to Silent mode to receive the communication data on the CAN bus. In Silent Mode, the CAN Core is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, Error Frames), the bit is rerouted internally so that the CAN core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits. Figure 1-1 shows the connection of signals CAN_TX and CAN_RX to the CAN core in Silent mode.

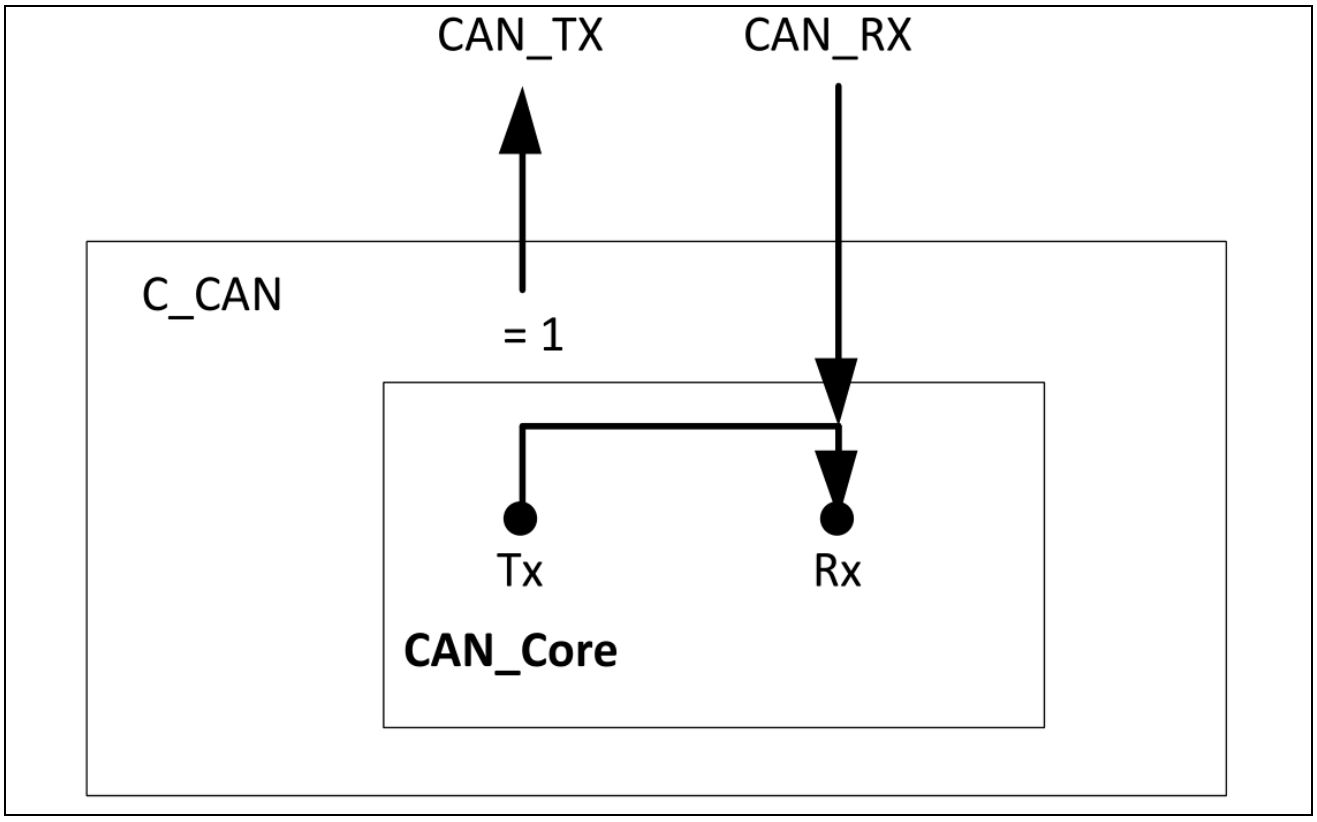

Figure 1-1 CAN Core in Silent Mode

1.2 Demo Result

This sample code requires two NUC131 NuTiny boards. One is used to download the “CAN_TX_RX” project, and the other is used to download the “CAN_Auto_Adjust_BPS” project. Then, connect the two NUC131 NuTiny boards as shown in Figure 3-1. Then, open the serial debugging tool and send the corresponding character as prompted.

The printing information of "CAN TX RX" function board is shown in Figure 1-2.

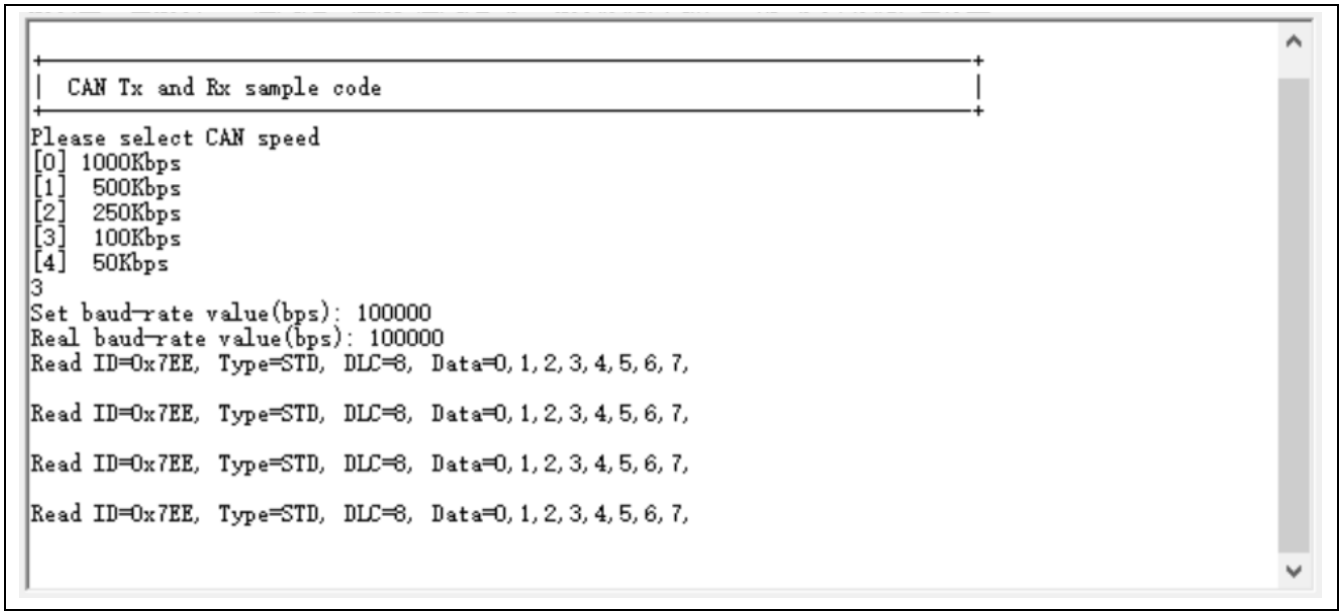

Figure 1-2 Printing Information of "CAN TX RX"

The printing information of "CAN_Auto_Adjust_BPS" function board is shown in Figure 1-3.

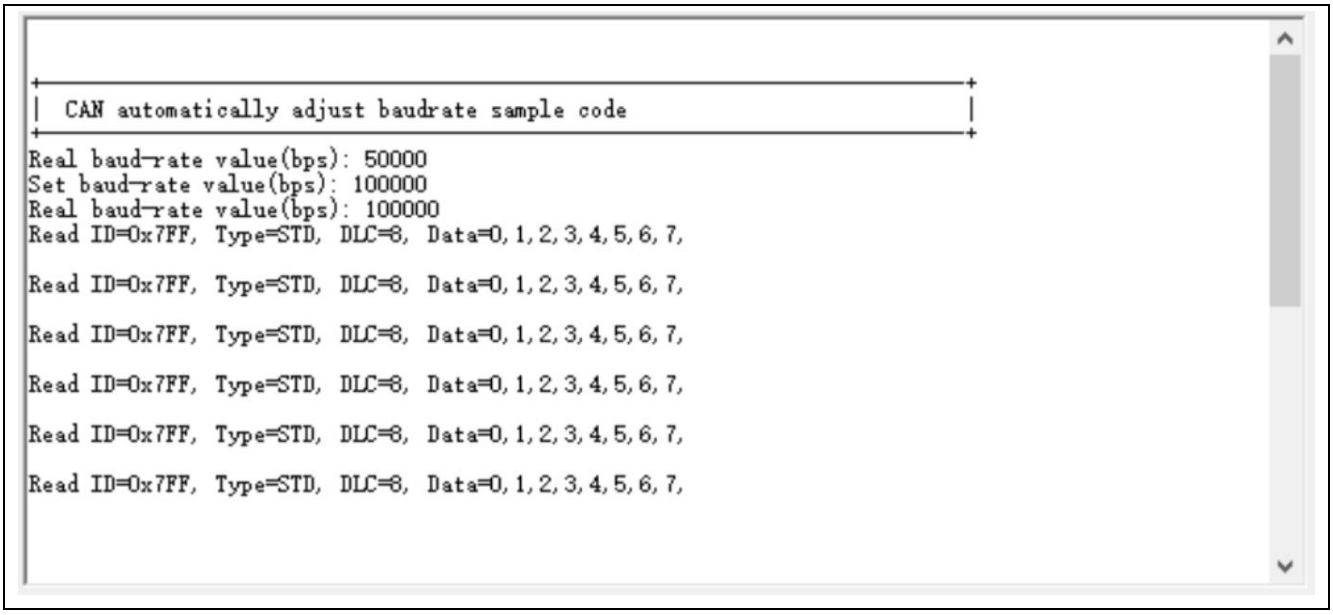

Figure 1-3 Printing Information of "CAN_Auto_Adjust_BPS"

2 Code Description

The following code description is based on the "CAN_Auto_Adjust_BPS" demo.

Baud rate detection range and step setting are described below.

```
#define BAUDRATE_MIN    50000
#define BAUDRATE_MAX    1000000
#define BAUDRATE_OFFSET 50000
```

Initialize the CAN.

```
void CAN_Init(uint32_t u32BaudRate)
{
    uint32_t u32RealBaudRate = 0;

    /* Reset CAN */
    SYS_ResetModule(CAN0_RST);

    u32RealBaudRate = CAN_SetBaudRate(CAN0, u32BaudRate);
    printf("Set baud-rate value(bps): %d\n", u32BaudRate);
    printf("Real baud-rate value(bps): %d\n", u32RealBaudRate);

    /* Enable CAN to Silent mode */
    CAN_EnterTestMode(CAN0, CAN_TEST_SILENT_Msk);

    /* Enable CAN interrupt and corresponding NVIC of CAN */
    CAN_EnableInt(CAN0, CAN_CON_IE_Msk | CAN_CON_SIE_Msk);
    NVIC_SetPriority(CAN0_IRQn, (1 << __NVIC_PRIO_BITS) - 2);
    NVIC_EnableIRQ(CAN0_IRQn);

    if(CAN_SetRxMsg(CAN0, MSG(0), CAN_STD_ID, 0x7FF) == FALSE)
    {
        printf("Set Rx Msg Object failed\n");
    }
}
```

Periodically change the CAN baud rate until the data on CAN bus can be received normally, and then exit Silent mode.

```
    if(g_u8RxOkFlag == 0)
    {
        if(g_u8RxOkCount == 0)
        {
            u32BaudRate = u32BaudRate + BAUDRATE_OFFSET;
            if(u32BaudRate > BAUDRATE_MAX)
            {
                u32BaudRate = BAUDRATE_MIN;
            }

            u32RealBaudRate = CAN_SetBaudRate(CAN0, u32BaudRate);
            printf("Set baud-rate value(bps): %d\n", u32BaudRate);
            printf("Real baud-rate value(bps): %d\n", u32RealBaudRate);

            CLK_SysTickLongDelay(100000);
        }
        else if(g_u8RxOkCount >= RX_OK_COUNT)
        {
            g_u8RxOkFlag = 1;
            CAN_LeaveTestMode(CAN0);
        }
    }
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - NUC131_Series_BSP_CMSIS_V3.00.005
- IDE version
  - Keil uVersion 4.74

3.2 Hardware Requirements

- Circuit components
  - NuTiny-EVB-NUC131-LQFP64 V1.0
  - CAN Transceiver
- Pin Connect

The hardware connection diagram for this demo code is shown in Figure 3-1.

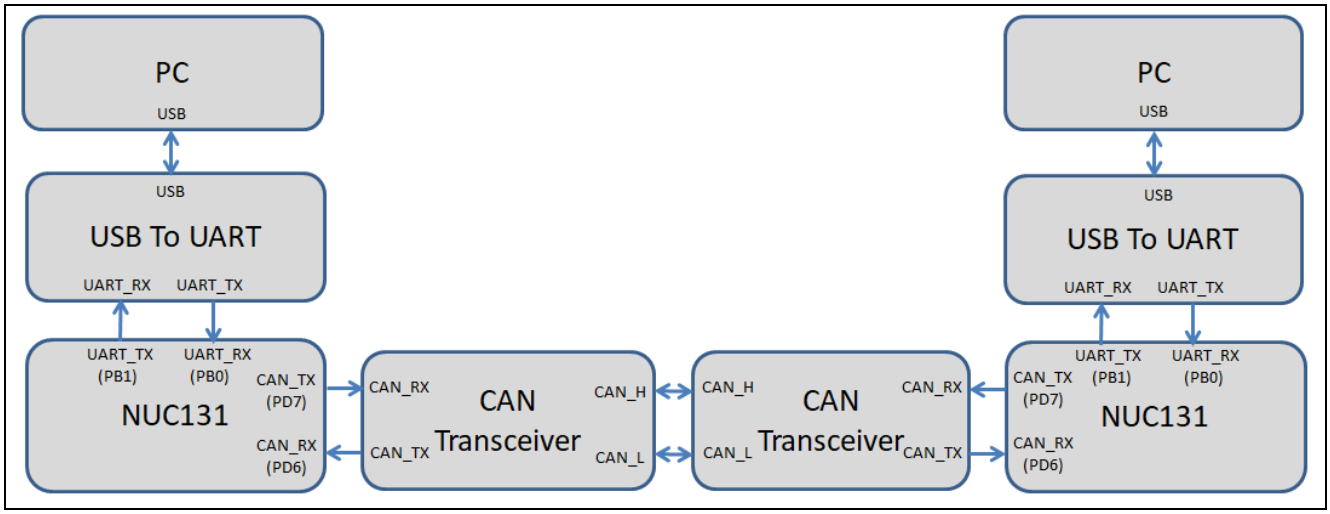

Figure 3-1 Hardware Connection Diagram

The CAN transceiver used in this demo is SN65HVD230D, and its connection with NUC131 is shown in Figure 3-2.

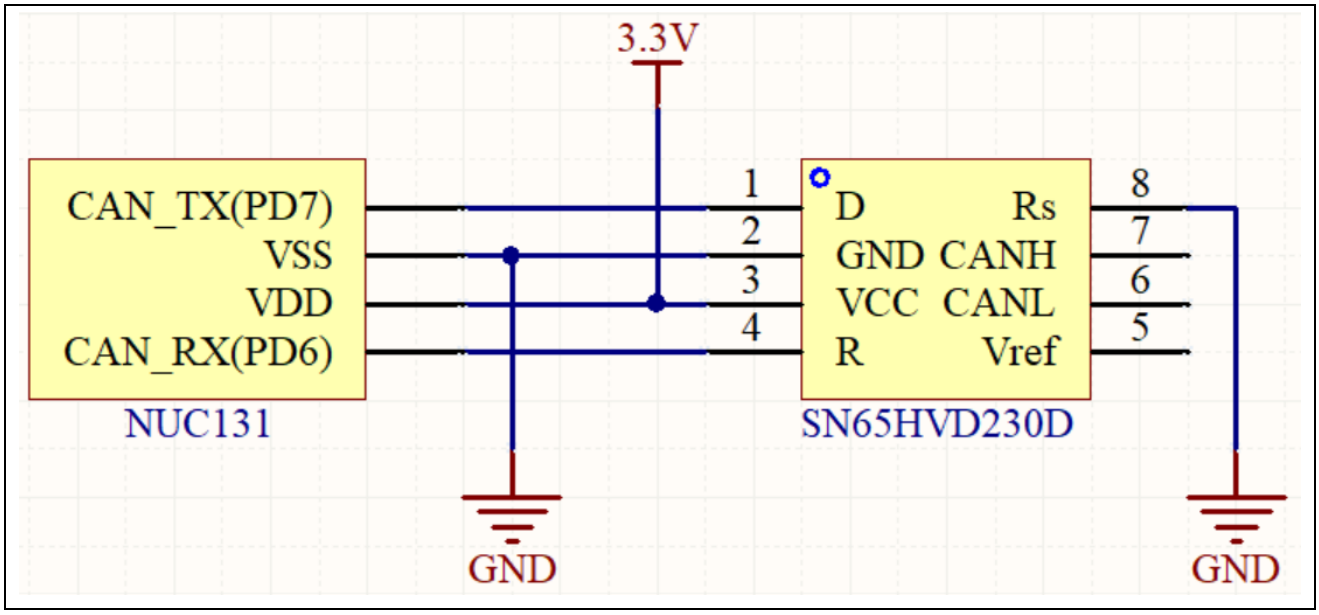

Figure 3-2 Connection Diagram between NUC131 and CAN Transceiver

If other CAN transceivers are used, the connection may be different. Please refer to the corresponding CAN transceiver datasheet for details.

4 Directory Information

The directory structure is shown below.

EC_NUC131_CAN_Auto_Adjust_BPS_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

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 CAN_TX_RX.uvproj and CAN_Auto_Adjust_BPS.uvproj.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.08.20	1.00	1. Initially issued.

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.