Automatic Baud Rate Adjustment with M253 CAN

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code receives data packets on the CAN bus, uses the detection of protocol error interrupts to match the baud rate of CAN bus communication, and dynamically adjusts the baud rate of M253 CAN.
BSP Version	M253 Series CMSIS BSP V3.00.003
Hardware	NuMaker–M253LE V1.1, ADM3050E CAN transceiver

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

There are two codes in this example: “CANFD_CAN_TxRx” and “CANFD_CAN_Auto_Adjust”. The “CANFD_CAN_TxRx” is an example code of CAN transceiver, which is convenient for testing “CANFD_CAN_Auto_Adjust”. The “CANFD_CAN_Auto_Adjust” dynamically adjusts the baud rate of M253 CAN by receiving data packets on the CAN bus, and adopts protocol error detection and interruption methods until the data on the CAN bus can be received normally and the communication baud rate of the CAN bus is matched. When the communication baud rate of the CAN bus is matched, data packets will be sent to the CAN bus periodically. The detectable baud rate range in the example is 50 Kbps ~ 1 Mbps, and the interval is 50 Kbps

1.1 Principle

In this application, the “CANFD_CAN_Auto_Adjust” routine will first set the M253 CAN to Bus Monitoring mode and receive the communication data on the CAN bus. In Bus Monitoring mode (see ISO 11898-1:2015, 10.14 Bus monitoring), the CAN core is able to receive valid data frames and valid remote frames, but cannot start a transmission. If the CAN core needs to send a dominant bit (ACK bit, error frame), this bit is directly connected to RX internally. Thus, although the CAN bus remains in a recessive state, the CAN core can detect this dominant bit. The Bus Monitoring mode can be used to analyze the transmission data of the CAN bus without disturbing the data transmission on the bus. In Bus Monitoring mode, the connection of CAN_TX and CAN_RX of the CAN core is shown in Figure 1-1.

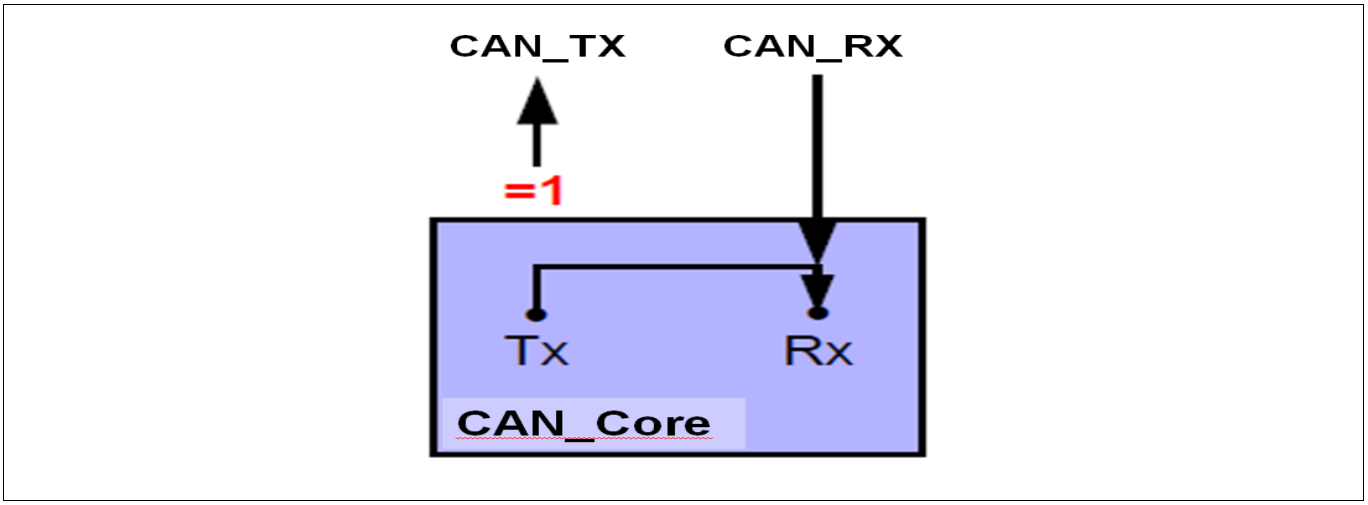

Figure 1-1 Bus Monitoring Mode

1.2 Demo Result

Use two M253 boards, one to program the “CANFD_CAN_TX_RX”, the other to program the “CANFD_CAN_Auto_Adjust”, and then connect the two M253 boards as shown in Figure 3-1. Then open the RS232 transmission interface and send the corresponding instructions according to the prompts.

For the “CANFD_CAN_TX_RX” example, set the CAN test baud rate, and test package ID Type and package message. The execution result is shown in Figure 1-2.

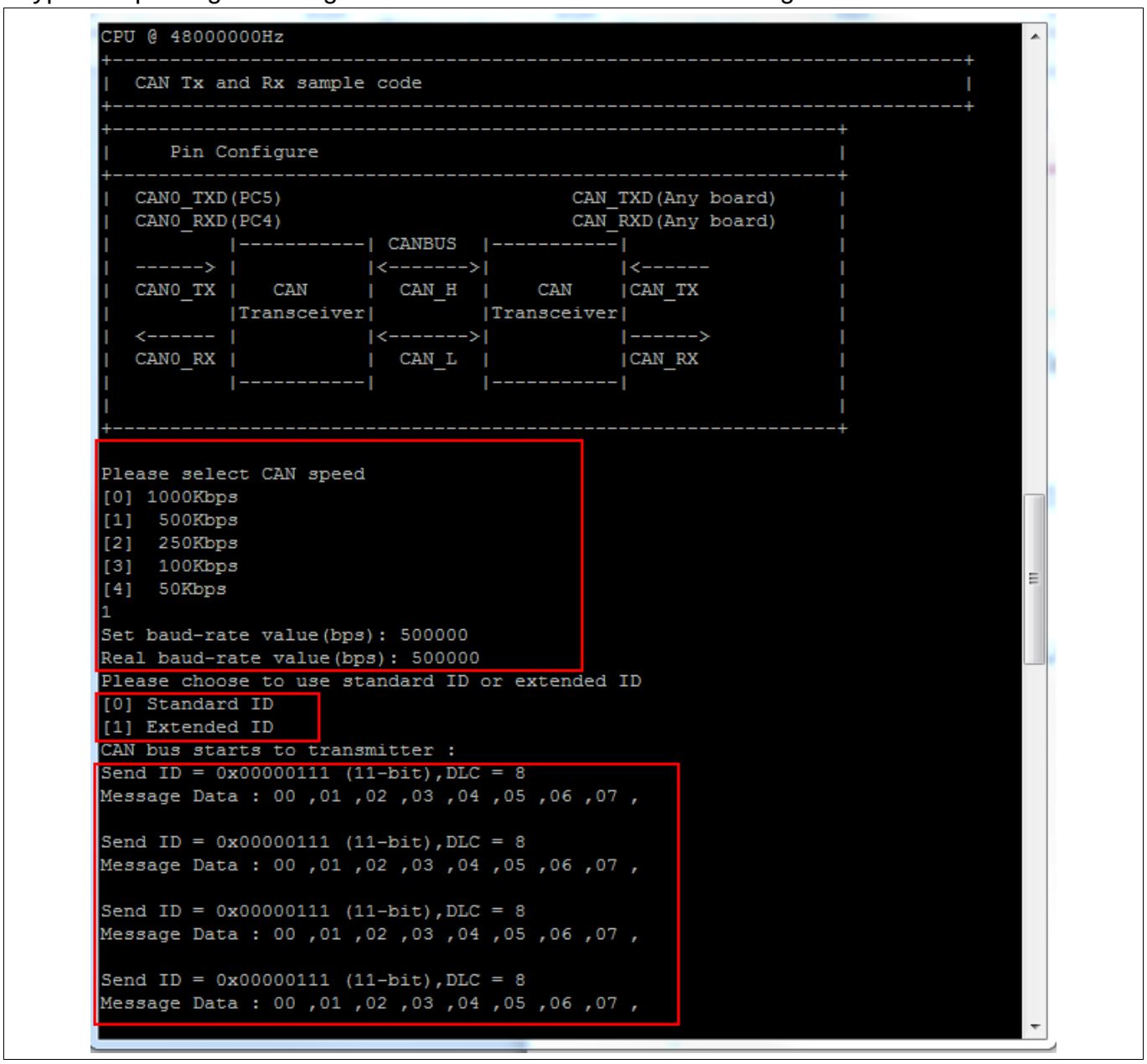

Figure 1-2 CANFD_CAN_TX_RX Execution Result

The “CANFD_CAN_Auto_Adjust” dynamically adjusts the CAN baud rate until the CAN bus data can be received normally. The execution result is shown in Figure 1-3.

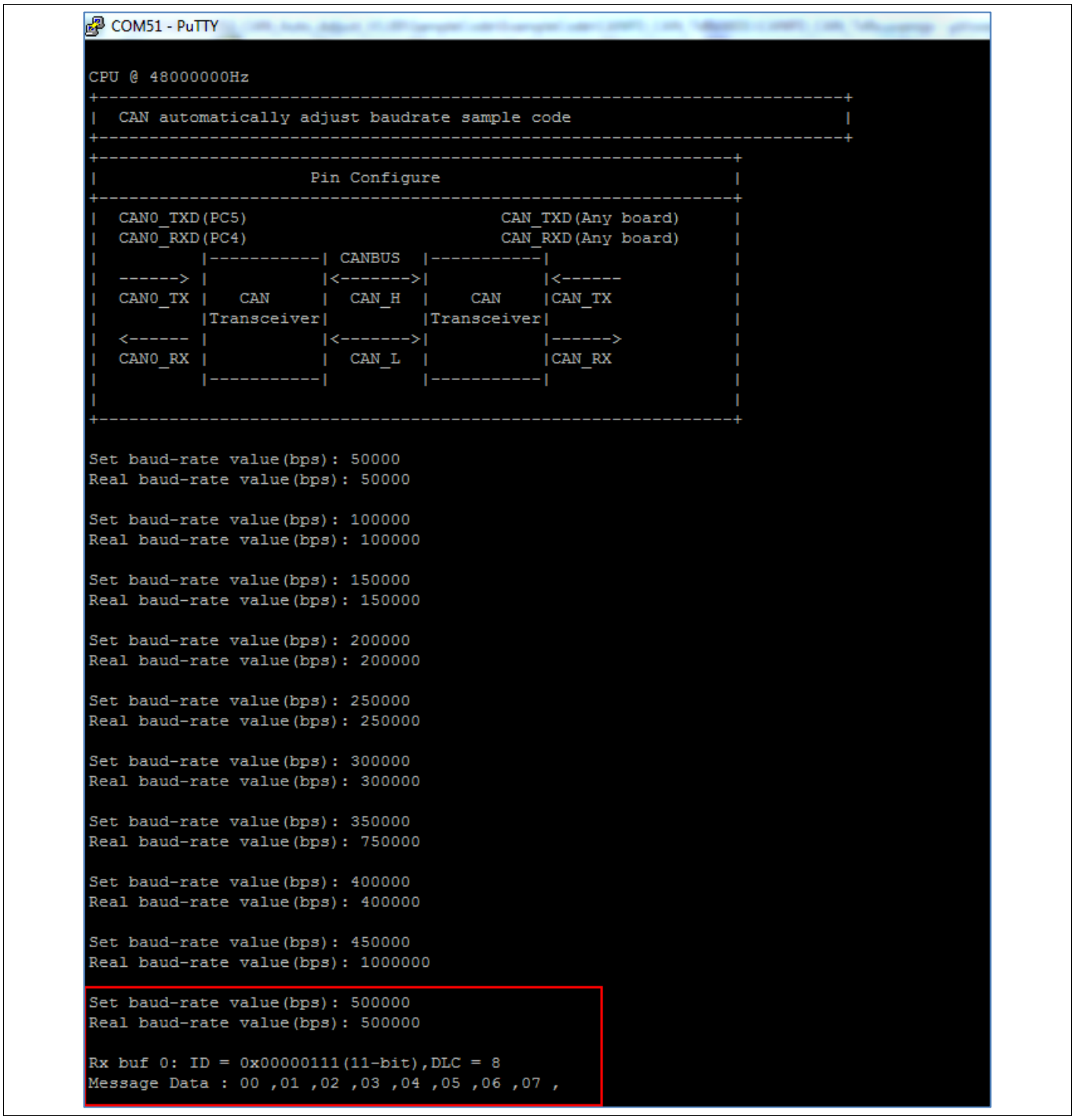

Figure 1-3 CANFD_CAN_Auto_Adjust Execution Result

Wait for the "CANFD_CAN_Auto_Adjust" to match the CAN communication baud rate, and then start to transmit with the "CANFD_CAN_TX_RX" message. The execution result is shown in Figure 1-4

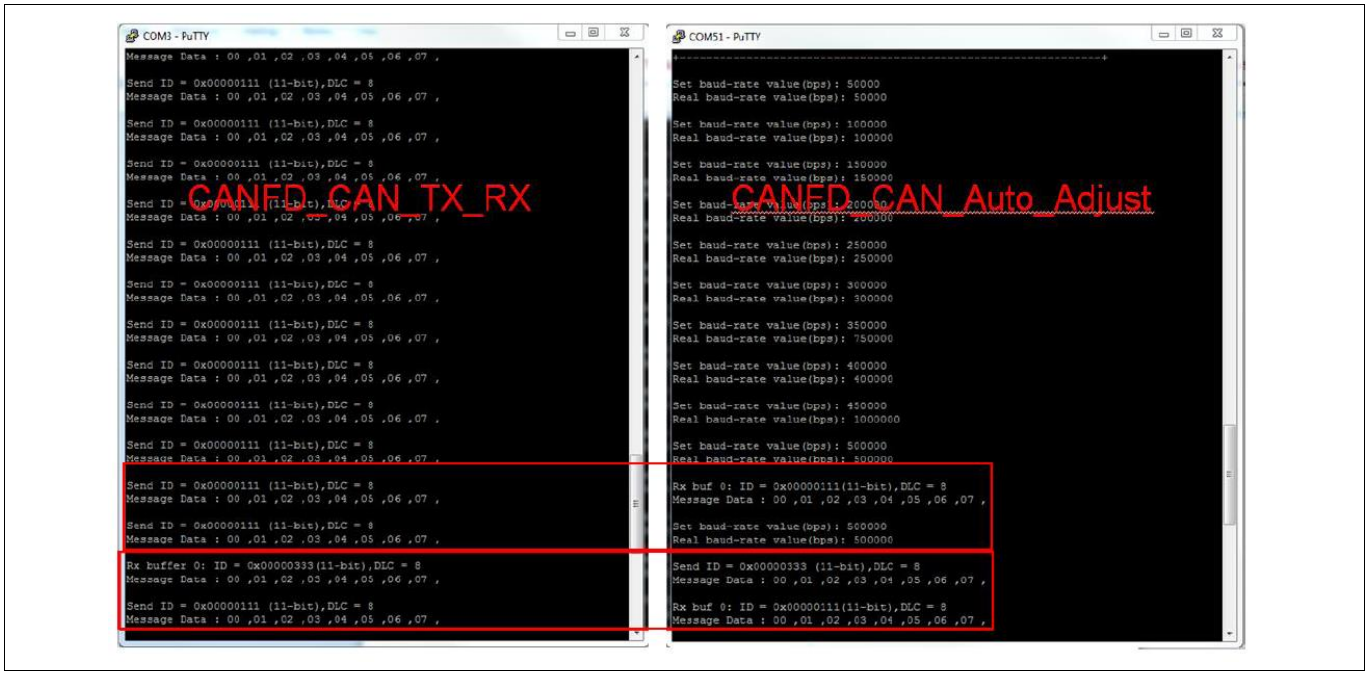

Figure 1-4 CANFD_CAN_Auto_Adjust and CANFD_CAN_TX_RX Communication Transmission Execution Result

2 Code Description

The following introduces the main program of the “CANFD_CAN_Auto_Adjust” example code.

Baud rate detection range and step size setting:

```
#define BAUDRATE_MIN        50000
#define BAUDRATE_MAX        1000000
#define BAUDRATE_OFFSET     50000
#define CAN_MON_DISABLE     0
#define CAN_MON_ENABLE      1
```

CAN initialization function:

```
void CAN_Init(uint32_t u32BitRate,uint8_t u8EnMonMode)
{
    CANFD_FD_T sCANFD_Config;
    CANFD_GetDefaultConfig(&sCANFD_Config, CANFD_OP_CAN_MODE);
    sCANFD_Config.sBtConfig.sNormBitRate.u32BitRate = u32BitRate;
    sCANFD_Config.sBtConfig.sDataBitRate.u32BitRate = 0;
    CANFD_Open(CANFD0, &sCANFD_Config);
    printf("Set baud-rate value(bps): %d\n",sCANFD_Config.sBtConfig.sNormBitRate.u32BitRate);
    printf("Real baud-rate value(bps): %d\n\n", Get_CAN_BitRate(CANFD0));
    /* receive 0x111 in CAN rx message buffer 0 by setting mask 0 */
    CANFD_SetSIDFltr(CANFD0, 0, CANFD_RX_BUFFER_STD(0x111, 0));


    /* receive 0x222 (29-bit id) in CAN rx message buffer 1 by setting mask 3 */
    CANFD_SetXIDFltr(CANFD0, 0, CANFD_RX_BUFFER_EXT_LOW(0x222, 1), CANFD_RX_BUFFER_EXT_HIGH(0x222,
1));
    /*Enable the Bus Monitoring Mode */
    if(u8EnMonMode == 1)
       CANFD0->CCCR |= CANFD_CCCR_MON_Msk;
    else
       CANFD0->CCCR &= ~CANFD_CCCR_MON_Msk;


/* Enable RX buffer new message ,Timeout,Protocol Error,Bus_Off interrupt using interrupt line 0. */
    CANFD_EnableInt(CANFD0, (CANFD_IE_TOOE_Msk | CANFD_IE_DRXE_Msk | CANFD_IE_BOE_Msk
                    |CANFD_IE_PEAE_Msk ), 0, 0, 0);
    /* CAN FD0 Run to Normal mode  */
    CANFD_RunToNormal(CANFD0, TRUE);
}
```

CAN sending message packet function:

```
void CAN_SendMessage(CANFD_FD_MSG_T *psTxMsg, E_CANFD_ID_TYPE eIdType, uint32_t u32Id, uint8_t
u8Len)
{
    uint8_t u8Cnt;
    psTxMsg->u32Id = u32Id;
    psTxMsg->eIdType = eIdType;
    psTxMsg->eFrmType = eCANFD_DATA_FRM;
    psTxMsg->bBitRateSwitch = 0;
    psTxMsg->u32DLC = u8Len;

    /* use message buffer 0 */
    if (eIdType == eCANFD_SID)
        printf("Send ID = 0x%08x (11-bit),DLC = %d\n", psTxMsg->u32Id,psTxMsg->u32DLC);
    else
        printf("Send ID = 0x%08x (29-bit),DLC = %d\n", psTxMsg->u32Id,psTxMsg->u32DLC);

    printf("Message Data : ");
    for (u8Cnt = 0; u8Cnt < psTxMsg->u32DLC; u8Cnt++)
    {
      psTxMsg->au8Data[u8Cnt] = u8Cnt;
      printf("%02X ,", psTxMsg->au8Data[u8Cnt]);
    }
     printf("\n\n");
    if (CANFD_TransmitTxMsg(CANFD0, 0, psTxMsg) != 1)
    {
        printf("Failed to transmit message\n");
    }
}
```

Dynamically adjust the CAN baud rate until the CAN bus data can be received correctly, exit the Bus Monitoring mode, and then start sending data packets:

```
while(1)
    {
        /* Start automatic bit rate adjustment */
        if(g_u8AutoAdjustOkFlag == 0)
        {
            /* Protocol Error in Arbitration Phase */
            if(g_u8CanBitRateAdjust == 20)
            {
                u32CanBitRate = u32CanBitRate + BAUDRATE_OFFSET;
```

```
                if(u32CanBitRate > BAUDRATE_MAX)
                {
                    u32CanBitRate = BAUDRATE_MIN;
                }
                /* CAN FD0 run to INIT mode  */
                CANFD_RunToNormal(CANFD0, FALSE);
                /* Can bus bit rate adjustment */
                CAN_Init(u32CanBitRate,CAN_MON_ENABLE);
                g_u8CanBitRateAdjust = 0;
            }
           if(g_u8RxBufRcvOk == 1)
           {
                g_u8AutoAdjustOkFlag = 1;
                /* CAN FD0 run to INIT mode  */
                CANFD_RunToNormal(CANFD0, FALSE);
                CAN_Init(u32CanBitRate,CAN_MON_DISABLE);
           }

       }
       /*Start data transmission communication*/
       if(g_u8AutoAdjustOkFlag == 1)
       {
         CAN_SendMessage(&sTxMsgFrame,eCANFD_SID,0x333,8);
         CLK_SysTickLongDelay(1000000);
         CAN_SendMessage(&sTxMsgFrame,eCANFD_XID,0x444,8);
         CLK_SysTickLongDelay(1000000);
       }
   }
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M253 Series CMSIS BSP V3.00.003
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M253LE V1.1
  - ADM3050E CAN transceiver
- Pin Connect

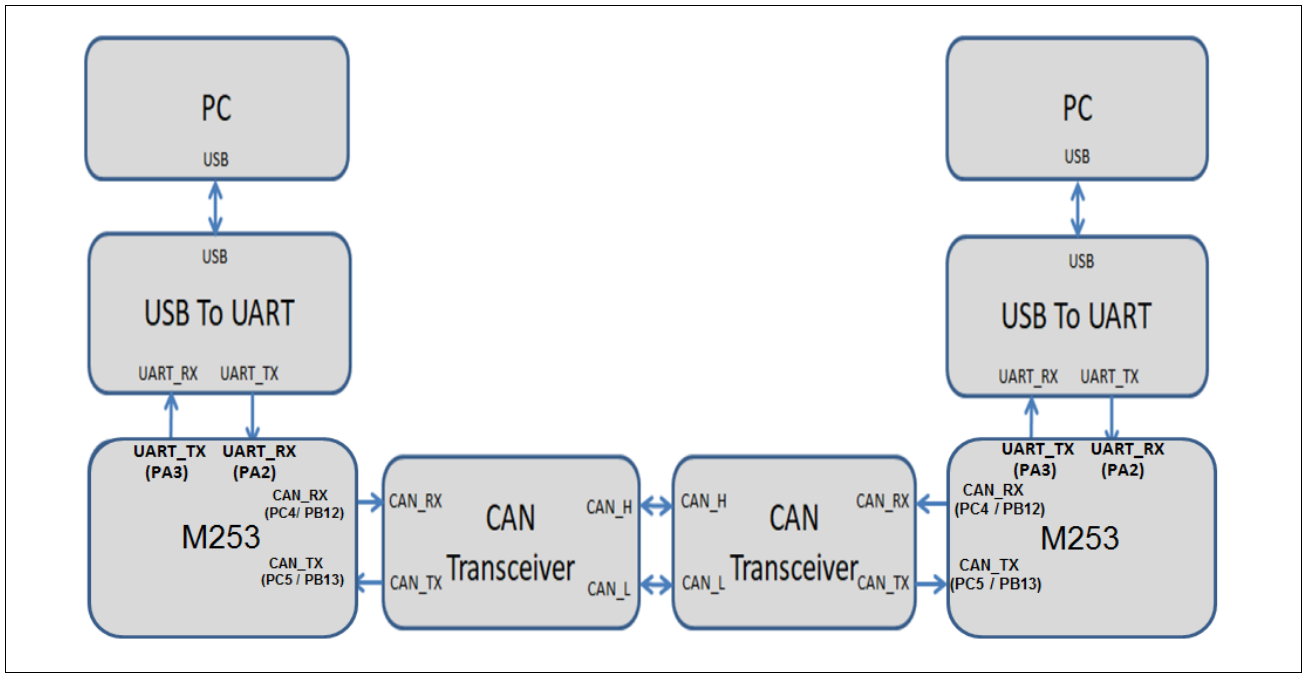

Figure 3-1 Schematic Diagram of Hardware Circuit Connection

In this example, the CAN transceiver used is ADM3050E, and its connection with M253 is shown in Figure 3-2.

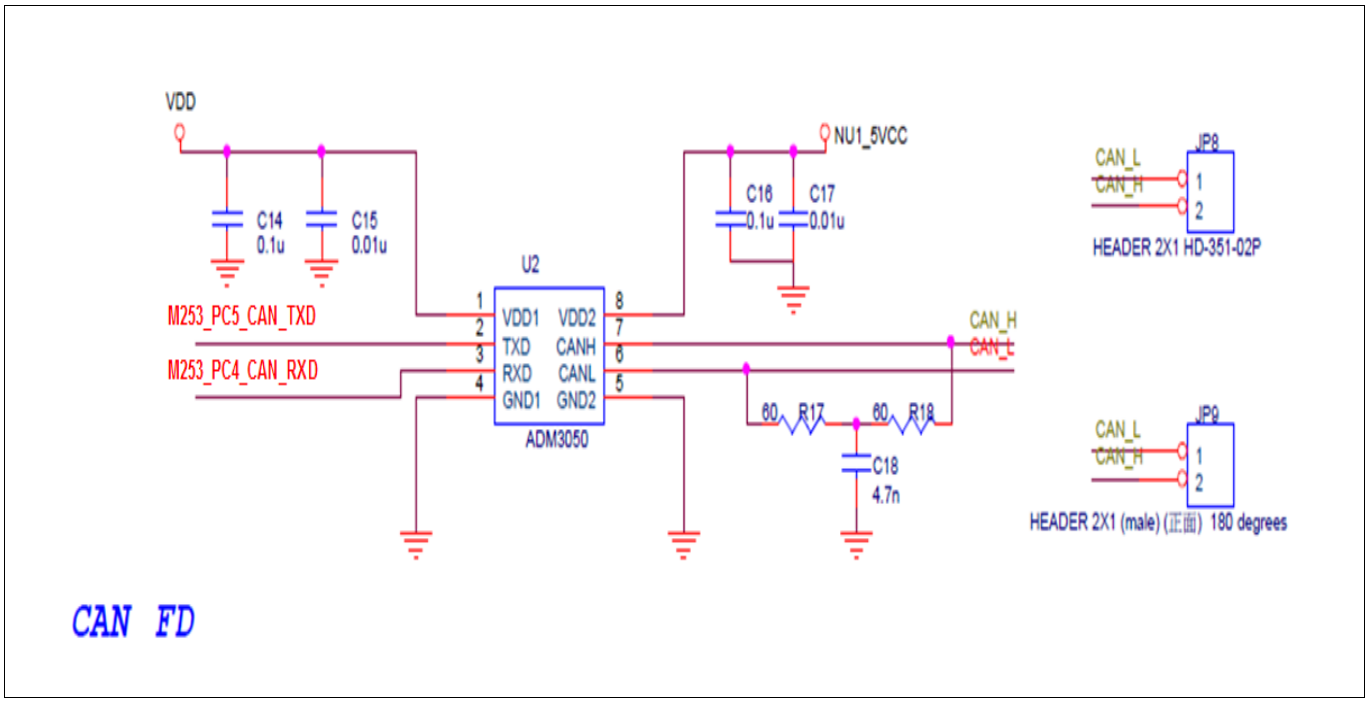

Figure 3-2 M253 and CAN Transceiver Connection Diagram

If other CAN transceivers are used, the connection may be different. For details, please refer to the corresponding CAN transceiver manual.

4 Directory Information

The directory structure is shown below.

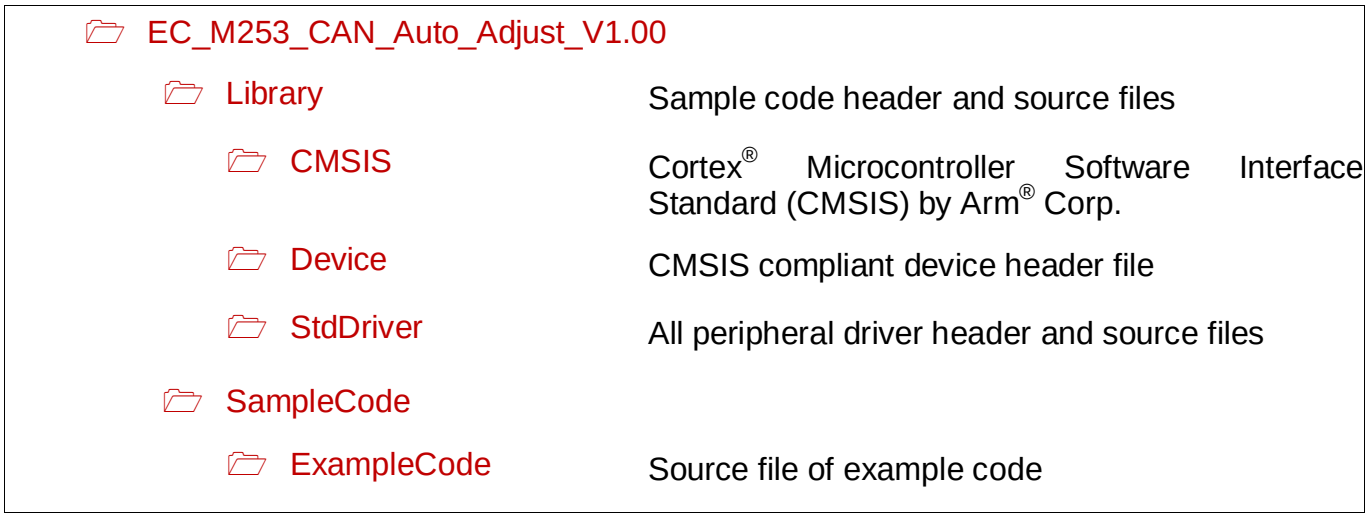

Figure 4-1 Directory Structure

5 Example Code Execution

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

6 Revision History

Date	Revision	Description
2021.11.30	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.