Using M251 to Drive MCP2515 CAN Module

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M251 to drive the MCP2515 CAN bus module through SPI, and demonstrates the sending and receiving of CAN data using the M253 with a CAN FD interface as the slave device.
BSP Version	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.00.006 M253_Series_BSP_CMSIS_V3.00.008
Hardware	NuMaker-M251SD V1.1 NuMaker-M253LEB 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

To implement CAN application on an microcontroller (MCU) without a built-in CAN FD interface, a stand-alone CAN controller can be used to connect the MCU via another communication interface to achieve CAN communication. In this example code, the widely available MCP2515 CAN bus module will be used, connecting to the NuMaker-M251SD via the SPI interface. The NuMaker-M253LEB will be used as the CAN Slave to complete the receiving and sending of CAN packages.

1.1 Principle

The MCP2515 CAN bus module, as shown in Figure 1-1, has two main components: the MCP2515 CAN controller, which uses SPI as the communication interface to handle CAN bus logic, and the TJA1050 CAN transceiver, which converts digital signals into differential signals. For the circuit diagram of this module, please refer to the link.

The module's supply voltage requirements, as shown in Figure 1-2 and Figure 1-3, are derived from the MCP2515 and TJA1050 datasheets. Due to the minimum voltage requirement of the TJA1050, at least 4.75V must be supplied to the module's V_{CC}. As for the SPI I/O high condition, as shown in Figure 1-2, it is necessary to refer to the MCP2515's V_{IH} specification. The V_{IH} needs to be at least 0.7*V_{DD} to adequately support the SPI I/O to High-Level Input Voltage.

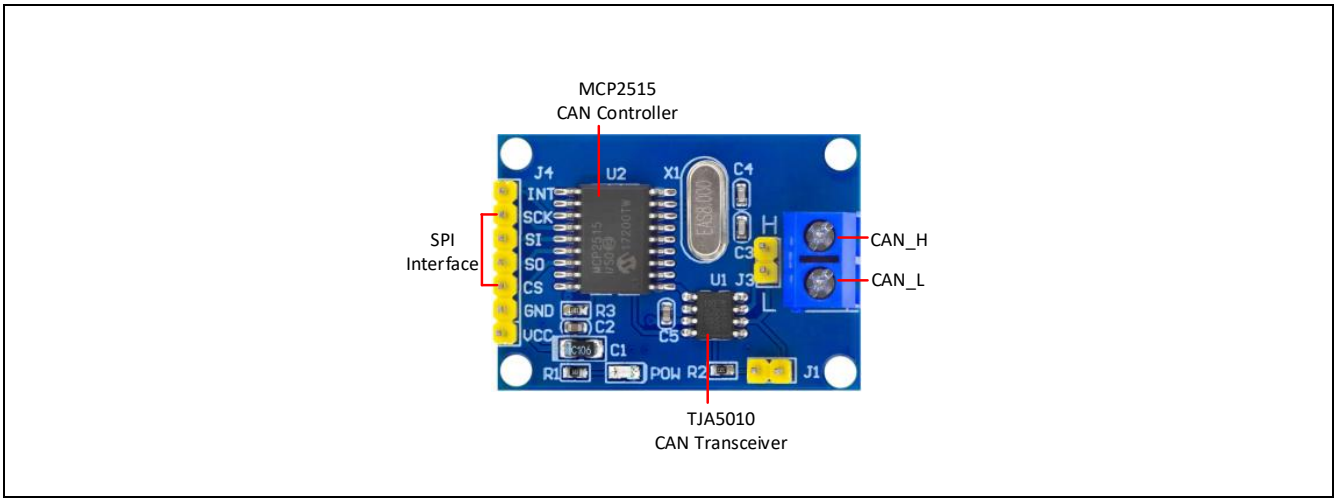

Figure 1-1 MCP2515 CAN Module

MCP2515

TABLE 13-1: DC CHARACTERISTICS

DC Characteristics			V_{DD} = 2.7V to 5.5V		Industrial (I): T_{AMB} = -40°C to +85°C Extended (E): T_{AMB} = -40°C to +125°C	
Param. No.	Sym	Characteristic	Min	Max	Units	Conditions
	V_{DD}	Supply Voltage	2.7	5.5	V	
	V_{RET}	Register Retention Voltage	2.4	—	V	
	V_{IH}	**High-Level Input Voltage**				
		RXCAN Pin	2	V_{DD} + 1	V	
		SCK, $\overline{CS}$, SI, $\overline{TXnRTS}$ Pins	0.7 V_{DD}	V_{DD} + 1	V	
		OSC1 Pin	0.85 V_{DD}	V_{DD}	V	
		$\overline{RESET}$ Pin	0.85 V_{DD}	V_{DD}	V	
	V_{IL}	**Low-Level Input Voltage**				
		RXCAN, $\overline{TXnRTS}$ Pins	-0.3	0.15 * V_{DD}	V	
		SCK, $\overline{CS}$, SI Pins	-0.3	0.4 * V_{DD}	V	
		OSC1 Pin	V_{SS}	0.3 * V_{DD}	V	
		$\overline{RESET}$ Pin	V_{SS}	0.15 * V_{DD}	V	
	V_{OL}	**Low-Level Output Voltage**				
		TXCAN Pin	—	0.6	V	I_{OL} = +6.0 mA, V_{DD} = 4.5V
		$\overline{RXnBF}$ Pin	—	0.6	V	I_{OL} = +8.5 mA, V_{DD} = 4.5V
		SO, CLKOUT Pins	—	0.6	V	I_{OL} = +2.1 mA, V_{DD} = 4.5V
		$\overline{INT}$ Pin	—	0.6	V	I_{OL} = +1.6 mA, V_{DD} = 4.5V
	V_{OH}	**High-Level Output Voltage**			V	
		TXCAN, $\overline{RXnBF}$ Pins	V_{DD} – 0.7	—	V	I_{OH} = -3.0 mA, V_{DD} = 4.5V
		SO, CLKOUT Pins	V_{DD} – 0.5	—	V	I_{OH} = -400 µA, V_{DD} = 4.5V
		$\overline{INT}$ Pin	V_{DD} – 0.7	—	V	I_{OH} = -1.0 mA, V_{DD} = 4.5V

Figure 1-2 MCP2515 Datasheet – DC Characteristics

SYMBOL	PARAMETER	CONDITIONS	MIN.	MAX.	UNIT
V_{CC}	supply voltage		4.75	5.25	V
V_{CANH}	DC voltage at pin CANH	0 < V_{CC} < 5.25 V; no time limit	–27	+40	V
V_{CANL}	DC voltage at pin CANL	0 < V_{CC} < 5.25 V; no time limit	–27	+40	V
$V_{I(dif)(bus)}$	differential bus input voltage	dominant	1.5	3	V
$t_{PD(TXD-RXD)}$	propagation delay TXD to RXD	V_S = 0 V; see Fig.7	–	250	ns
T_{vj}	virtual junction temperature		–40	+150	°C

Figure 1-3 TJA1050 Datasheet – DC Characteristics

The MCP2515 is controlled via an SPI interface connected to the MCU. All operations are performed by sending commands to and reading from registers. The complete list of registers can be found in the MCP2515 datasheet.

1.2 Demo Result

The result of compiling and programming this sample code shows that when the NuMaker-M251SD and NuMaker-M253LEB are designated as Slave or Master respectively, the successful transmission and reception of the specified CAN packets will be displayed respectively. The output is shown in Figure 1-4.

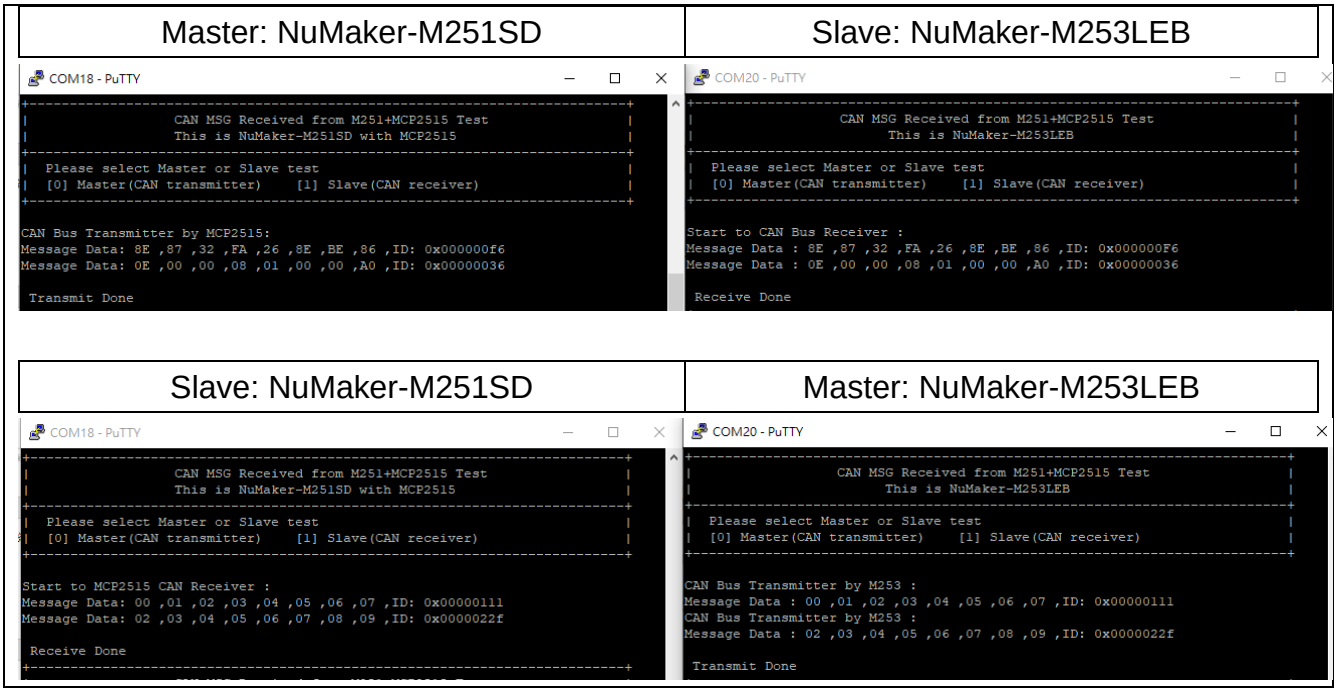

Figure 1-4 Compilation and Programming Results

2. Code Description

The M251 is used for handling CAN communication and transmission. The main program starts with *main.c* and in *MCP2515.c*, the corresponding SPI registers are set to initialize the MCP2515, configure its speed, and set the mode.

```
ERROR MCP2515_reset(void)
{
    startSPI();
    SPI_transfer(INSTRUCTION_RESET);

    endSPI();
    TIMER_Delay(TIMER0,500); //delay 500ms
    uint8_t zeros[14];
    memset(zeros, 0, sizeof(zeros));
    MCP2515_setRegisters(MCP_TXB0CTRL, zeros, 14);
    MCP2515_setRegisters(MCP_TXB1CTRL, zeros, 14);
    MCP2515_setRegisters(MCP_TXB2CTRL, zeros, 14);

    MCP2515_setRegister(MCP_RXB0CTRL, 0);
    MCP2515_setRegister(MCP_RXB1CTRL, 0);

    MCP2515_setRegister(MCP_CANINTE, CANINTF_RX0IF | CANINTF_RX1IF | CANINTF_ERRIF |
CANINTF_MERRF);

    // receives all valid messages using either Standard or Extended Identifiers that
    // meet filter criteria. RXF0 is applied for RXB0, RXF1 is applied for RXB1
    MCP2515_modifyRegister(MCP_RXB0CTRL,
                  RXBnCTRL_RXM_MASK | RXB0CTRL_BUKT | RXB0CTRL_FILHIT_MASK,
                  RXBnCTRL_RXM_STDEXT | RXB0CTRL_BUKT | RXB0CTRL_FILHIT);
    MCP2515_modifyRegister(MCP_RXB1CTRL,
                  RXBnCTRL_RXM_MASK | RXB1CTRL_FILHIT_MASK,
                  RXBnCTRL_RXM_STDEXT | RXB1CTRL_FILHIT);

    RXF filters[] = {RXF0, RXF1, RXF2, RXF3, RXF4, RXF5};
    for (int i=0; i<6; i++) {
        bool ext = (i == 1);
        ERROR result = MCP2515_setFilter(filters[i], ext, 0);
        if (result != ERROR_OK) {
            return result;
        }
    }

    MASK masks[] = {MASK0, MASK1};
    for (int i=0; i<2; i++) {
        ERROR result = setFilterMask(masks[i], true, 0);
        if (result != ERROR_OK) {
            return result;
        }
    }

    return ERROR_OK;
}
```

```
ERROR MCP2515_setBitrate(const CAN_SPEED canSpeed)
{
    return MCP2515_setBitrateWithClock(canSpeed, MCP_8MHZ);
}

ERROR MCP2515_setNormalMode()
{
    return MCP2515_setMode(CANCTRL_REQOP_NORMAL);
}
```

After configuring the MCP2515, enter the MCP2515_CAN_TxRxTest function to perform transmission or reception tests. The transmission and reception operations are completed through APIs in the *MCP2515.c* file.

```
void MCP2515_CAN_TxRxTest(void)
{
    uint8_t u8Item;
    uint8_t u8Cnt;
    uint8_t u8RxTestNum = 0;

    printf("+-------------------------------------------------------------------------+\n");
    printf("|                  CAN MSG Received from M251+MCP2515 Test                |\n");
    printf("|                  This is NuMaker-M251SD with MCP2515                    |\n");
    printf("+-------------------------------------------------------------------------+\n");
    printf("|  Please select Master or Slave test                                     |\n");
    printf("|  [0] Master(CAN transmitter)    [1] Slave(CAN receiver)                 |\n");
    printf("+-------------------------------------------------------------------------+\n\n");
    u8Item = getchar();
    u8RxTestNum = 0;
    bool result ;
    if (u8Item == '0')
    {
      printf("CAN Bus Transmitter by MCP2515:\n");
      MCP2515_sendMessage(&canMsg1);
      MCP2515_sendMessage(&canMsg2);
      TIMER_Delay(TIMER0, 100);//delay 100ms
      if((MCP2515_sendMessage(&canMsg1) || MCP2515_sendMessage(&canMsg2)) == 0) {
            printf("Message Data: ");
            for(u8Cnt =0; u8Cnt<canMsg1.can_dlc; u8Cnt++) {
                printf("%02X ,",canMsg1.data[u8Cnt]);
            }
            printf("ID: 0x%08lx \n",canMsg1.can_id);
            printf("Message Data: ");
            for(u8Cnt =0; u8Cnt<canMsg2.can_dlc; u8Cnt++) {
                printf("%02X ,",canMsg2.data[u8Cnt]);
            }
            printf("ID: 0x%08lx \n",canMsg2.can_id);

            printf("\n Transmit Done\n");
        } else {
            printf("\n Transmit Fail\n");
        }
    } else {
        printf("Start to MCP2515 CAN Receiver :\n");
        do {
            if(MCP2515_readMessage(&canMsg0) == ERROR_OK) {
```

```
                printf("Message Data: ");
                for(u8Cnt=0; u8Cnt<canMsg0.can_dlc; u8Cnt++) {
                    printf("%02X ,",canMsg0.data[u8Cnt]);
                }
                printf("ID: 0x%08lx \n",canMsg0.can_id);
                u8RxTestNum++;
            }
        } while(u8RxTestNum < 2);
        printf("\n Receive Done\n");
    }
}

int main(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Init System, IP clock and multi-function I/O. */
    SYS_Init();
    /* Lock protected registers */
    SYS_LockReg();

    /* Init SPI */
    SPI_Init();

    MCP2515_reset();
    MCP2515_setBitrate(CAN_125KBPS);
    MCP2515_setNormalMode();

    canMsg1.can_id  = 0x0F6;
    canMsg1.can_dlc = 8;
    canMsg1.data[0] = 0x8E;
    canMsg1.data[1] = 0x87;
    canMsg1.data[2] = 0x32;
    canMsg1.data[3] = 0xFA;
    canMsg1.data[4] = 0x26;
    canMsg1.data[5] = 0x8E;
    canMsg1.data[6] = 0xBE;
    canMsg1.data[7] = 0x86;

    canMsg2.can_id  = 0x036;
    canMsg2.can_dlc = 8;
    canMsg2.data[0] = 0x0E;
    canMsg2.data[1] = 0x00;
    canMsg2.data[2] = 0x00;
    canMsg2.data[3] = 0x08;
    canMsg2.data[4] = 0x01;
    canMsg2.data[5] = 0x00;
    canMsg2.data[6] = 0x00;
    canMsg2.data[7] = 0xA0;

    MCP2515_CAN_TxRxTest();

    while(1);
}
```

On the M253 side, the main program starts with *main.c*. The specified CAN packet transmission or reception is completed in the CAN_TxRxTest function.

```
void CAN_TxRxTest(void)
{
    uint8_t u8Item;
    uint8_t u8Cnt;
    CANFD_FD_MSG_T      sRxFrame;
    CANFD_FD_MSG_T      sTxMsgFrame;

    /* CAN interface initialization*/
    CAN_Init();

    printf("+-----------------------------------------------------------------+\n");
    printf("|              CAN MSG Received from M251+MCP2515 Test            |\n");
    printf("|                  This is NuMaker-M253LEB                        |\n");
    printf("+-----------------------------------------------------------------+\n");
    printf("|  Please select Master or Slave test                             |\n");
    printf("|  [0] Master(CAN transmitter)    [1] Slave(CAN receiver)         |\n");
    printf("+-----------------------------------------------------------------+\n\n");
    u8Item = getchar();

    uint8_t u8ErrFlag = 0;
    uint8_t u8RxTestNum = 0;

    if(u8Item == '0') {
        uint8_t u8TxTestNum = 0;

        /* Send 2 messages with different ID and data size */
        for(u8TxTestNum = 0; u8TxTestNum < 2 ; u8TxTestNum++) {
            printf("CAN Bus Transmitter by M253 :\n");

            /* Set the ID Number */
            if(u8TxTestNum == 0)      sTxMsgFrame.u32Id = 0x111;
            else if(u8TxTestNum == 1) sTxMsgFrame.u32Id = 0x22F;
            /*Set the ID type*/
            sTxMsgFrame.eIdType = eCANFD_SID;
            /*Set the frame type*/
            sTxMsgFrame.eFrmType = eCANFD_DATA_FRM;
            /*Set the bitrate switch */
            sTxMsgFrame.bBitRateSwitch = 0;
            /*Set the data lenght */
            sTxMsgFrame.u32DLC = 8;

            for(u8Cnt = 0; u8Cnt < sTxMsgFrame.u32DLC; u8Cnt++) sTxMsgFrame.au8Data[u8Cnt]
= u8Cnt + u8TxTestNum*2;
            printf("Message Data : ");
            for(u8Cnt = 0; u8Cnt < sTxMsgFrame.u32DLC; u8Cnt++) {
                printf("%02X ,", sTxMsgFrame.au8Data[u8Cnt]);
            }
            printf("ID: 0x%08x \n", sTxMsgFrame.u32Id);

            /* use message buffer 0 */
            if(CANFD_TransmitTxMsg(CANFD0, 0, &sTxMsgFrame) != eCANFD_TRANSMIT_SUCCESS) {
                printf("Failed to transmit message\n");
            }
        }
```

```
        printf("\n Transmit Done\n");
    } else {
        printf("Start to CAN Bus Receiver :\n");
        /* Receive 2 messages with different ID */
        do {
            /* check for any received messages on CAN FD0 message buffer 0 */
            if(CANFD_ReadRxBufMsg(CANFD0, 0, &sRxFrame) == eCANFD_RECEIVE_SUCCESS) {
                printf("Message Data : ");
                for(u8Cnt = 0; u8Cnt < sRxFrame.u32DLC; u8Cnt++) {
                    printf("%02X ,", sRxFrame.au8Data[u8Cnt]);
                }
                printf("ID: 0x%08X\n", sRxFrame.u32Id);

                /* Check Standard ID number and Data Length*/
                if(((sRxFrame.u32Id != 0x0F6) && (sRxFrame.u32Id != 0x036)) ||
                   (RX_TEMP_LEN != sRxFrame.u32DLC) || (sRxFrame.eIdType != eCANFD_SID)) {
                    u8ErrFlag = 1;
                }
                if(u8ErrFlag == 1) {
                    printf("CAN STD ID or Data Error \n");
                }

                u8RxTestNum++;
            }
        } while(u8RxTestNum < 2);
        printf("\n Receive Done\n");
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.00.006
  - M253_Series_BSP_CMSIS_V3.00.008
- IDE version
  - Keil uVersion 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M251SD V1.1
  - NuMaker-M253LEB V1.0
- Pin Connect
  - To ensure stable operating voltage, an external power supply is used to provide a 5V voltage to the MCP2515 module. For other wiring connections, please refer to the pin description in Table 3-1 and the connection in Figure 3-1.

Power Supply	MCP2515	NuMaker-M251SD	NuMaker-M253LEB
5V	VCC	-	-
-	SCK	SPI0_CLK Pin 42 (PD.2)	-
-	SI	SPI0_MOSI Pin 44 (PD.0)	-
-	SO	SPI0_MISO Pin 43 (PD.1)	-
-	CS	SPI0_SS Pin 41 (PD.3)	-
GND	GND	GND	GND
-	CAN_H	-	CAN_H
-	CAN_L	-	CAN_L

Table 3-1 Pin Description

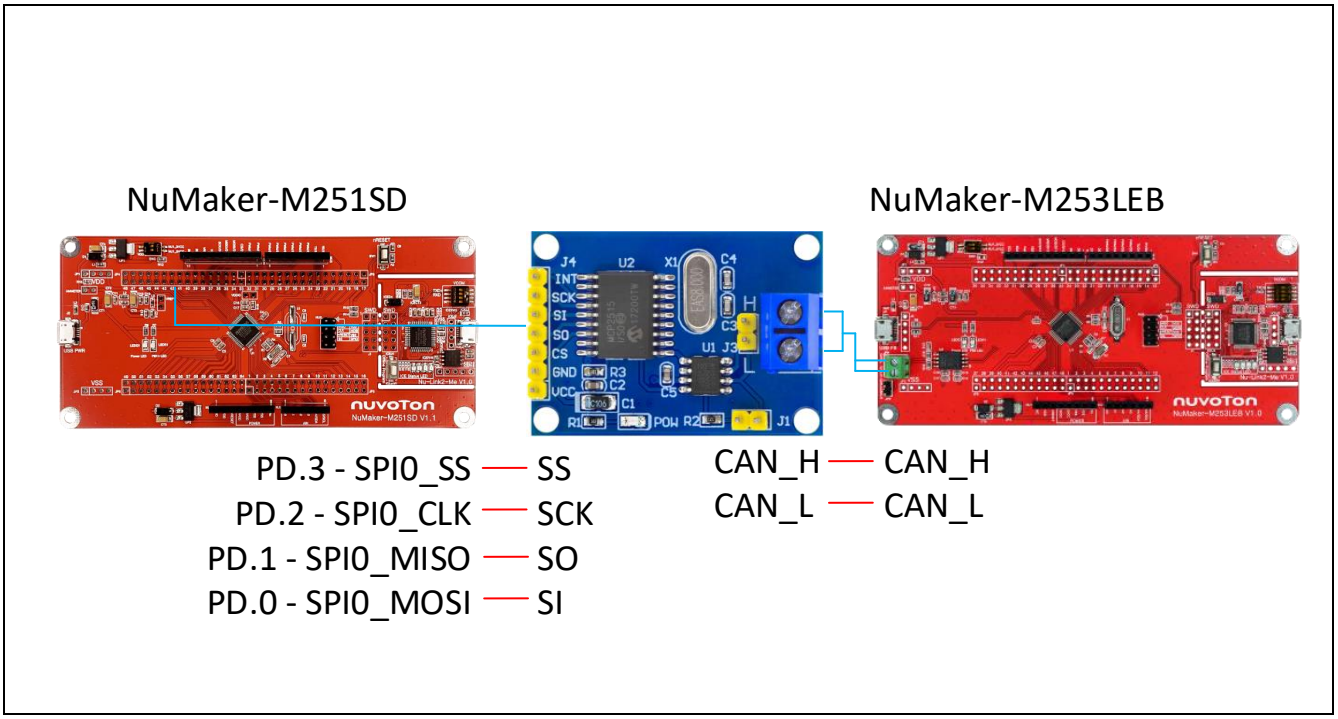

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M251_CAN_Controller_MCP2515_V1.00	
M251_SPI_MCP2515_Master	
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	
Project	Source file of example code
M253_CAN_Slave	
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	
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 double-click *M251_SPI_MCP2515_Master.uvprojx* and *M253_CAN_Slave.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2024.05.17	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.