4-CANFD Transmit-Receive Code in M460 uCOSii

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M460 series microcontroller (MCU) to port uCOSii and execute 4-CANFD transmission and reception.
BSP Version	M460_Series_BSP_CMSIS_V3.00.002
Hardware	NuMaker-M467HJ V1.0

1. Overview

This example code uses the M460 series microcontroller (MCU) to port uCOSii, and respectively establishes tasks for 4 CANFDs that are responsible for sending and receiving messages.

1.1 Principle

The M460 CANFD has a powerful function. A maximum of 32 message buffers can be configured to wait for sending. As shown in Figure 1-1, the number of message buffers and the first address of buffers are configured by the TXBC register, which is in the function CANFD_Tx_Init(). In a multitasking system, the task that needs to send CANFD messages can exclusively use several buffers to send messages, so that CANFD does not need to be managed with semaphores.

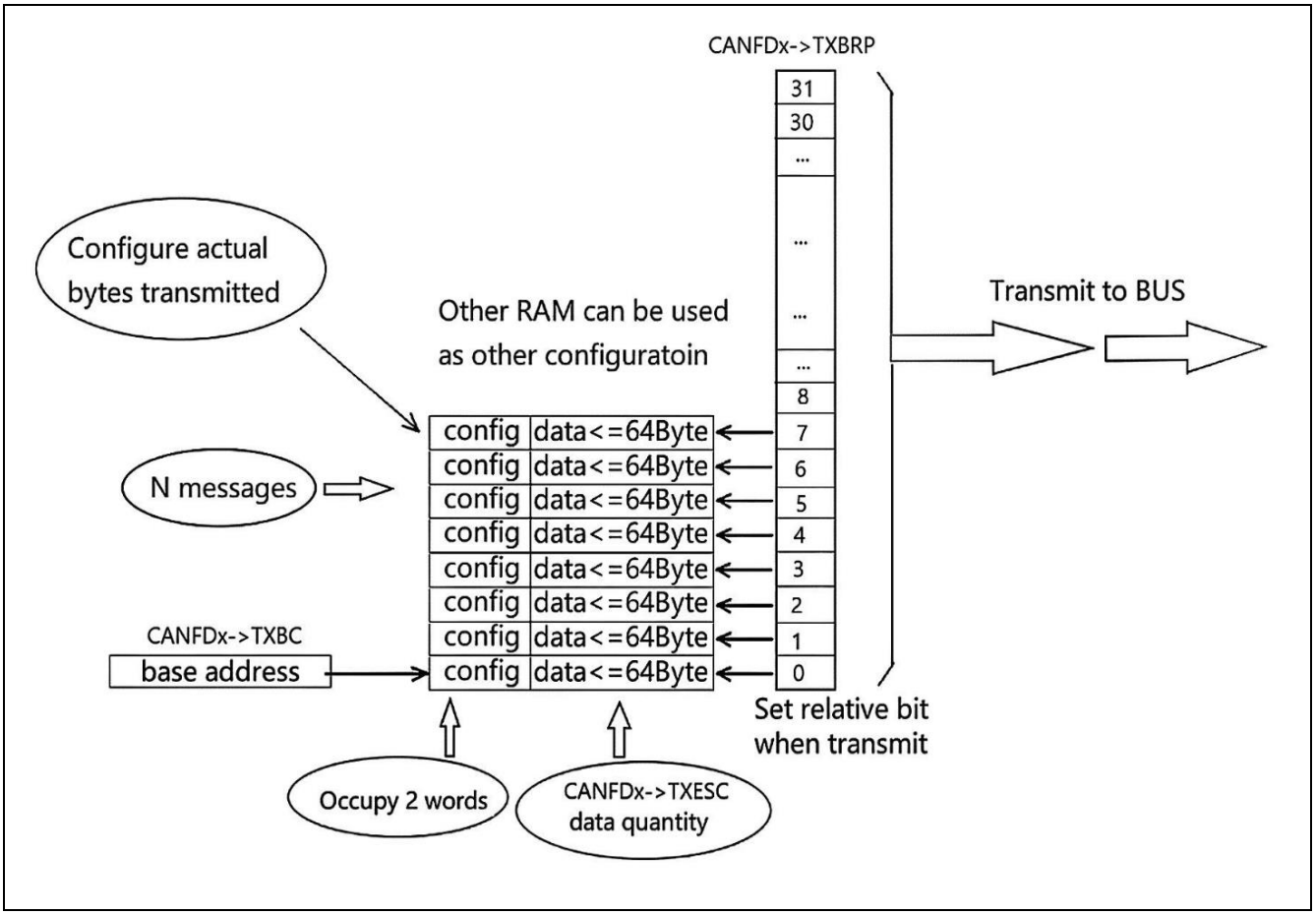

Figure 1-1 M460 CANFD Tx Register

The data structure of each message to be sent is composed of “Configuration + Data (maximum 64 bytes)”. Figure 1-2 shows the data structure of the sending messages.

Bit	31	24	23	16	15	8	7	0
T0	ESI	XTD	RTR	ID[28:0]				
T1	MM[7:0]		EFC	-	FDF	BRS	DLC[3:0]	MM[15:8]	-
T2	DB3[7:0]		DB2[7:0]		DB1[7:0]		DB0[7:0]	
T3	DB7[7:0]		DB6[7:0]		DB5[7:0]		DB4[7:0]	

Figure 1-2 Data Structure of Sending Messages

When receiving messages, the messages on the CANFD bus are filtered by up to 128 groups of standard SID and up to 64 groups of extended XID. Then, the received messages can be placed in a specified space of the 64 buffers. A task in the multitasking system can monopolizes a receive buffer. In this way, the receiving of CANFD messages can be managed without semaphore. The received messages can also be placed in FIFO0 or FIFO1 with a depth of 64 levels as shown in Figure 1-3. When there are a plenty of messages on bus, FIFO can reduce the process time of CPU. The received messages of this code are put into the respective FIFO1 of CANFD0~CANFD3.

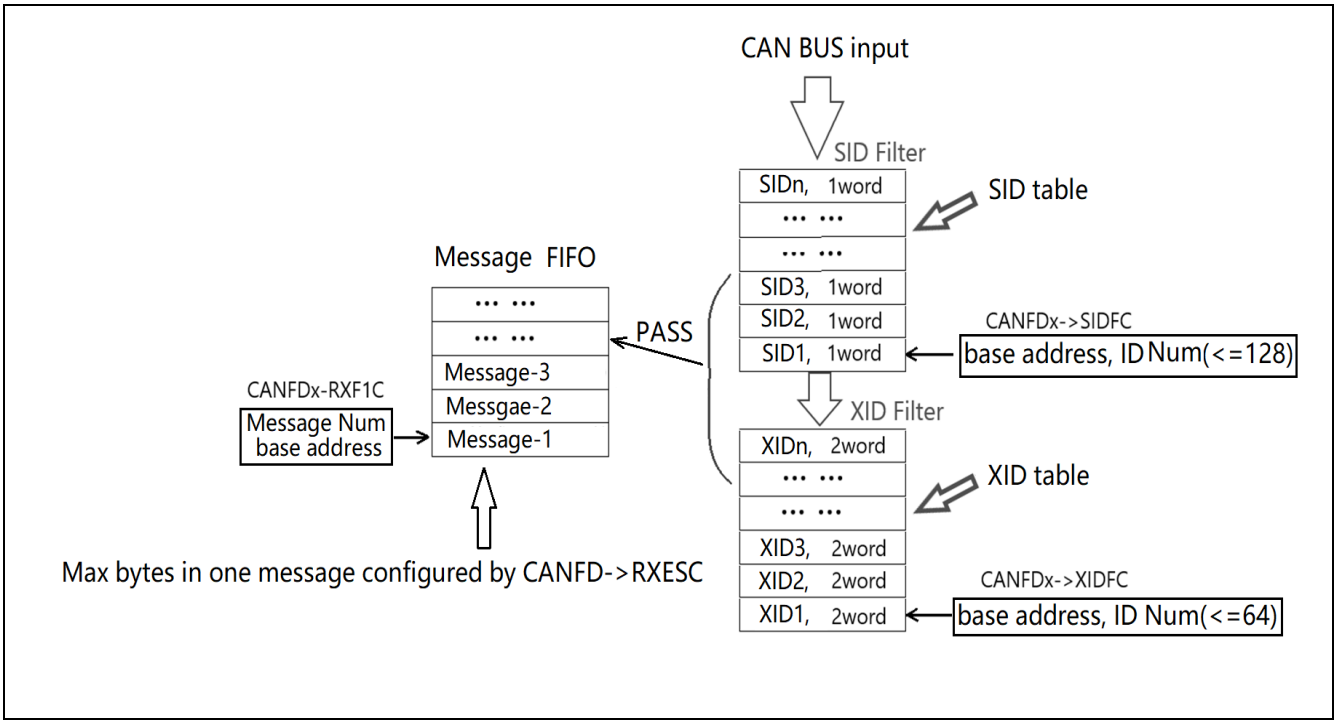

Figure 1-3 Diagram of CANFD Reception

Figure 1-4 shows the data structure of the SID used to filter messages when received.

Bit	31		24	23	16	15	8	7	0
S0	SFT [1:0]	SFEC [2:0]	SID1[10:0]			-	SID2(or Mask)[10:0]		

Figure 1-4 SID Structure

Figure 1-5 shows the data structure of the XID used to filter messages when reception.

Bit	31		24	23	16	15	8	7	0
F0	EFECV [2:0]		XID1[28:0]						
F0	EFT [1:0]	-	XID2(or MASK)[28:0]						

Figure 1-5 XID Structure

1.2 Demo Result

Figure1-6 shows the execution result of this code on the NuMAKE-M467HJ V1.0 board which only has one CANFD transceiver.

```
  Task A

Task 0
CAN0 SID=0x0116(04 Bytes): 139,05,00,00,
Task 1
CAN0 SID=0x0116(04 Bytes): 140,05,00,00,
Task 2
Task 3
Task 0
CAN0 XID=0x0118(08 Bytes): 00,00,00,00,225,255,255,255,
CAN0 SID=0x0116(04 Bytes): 141,05,00,00,
Task 1

  Task A

Task 0
Task 2
Task 3
CAN0 SID=0x0116(04 Bytes): 142,05,00,00,
Task 0
Task 1
Task 0
CAN0 XID=0x0118(08 Bytes): 00,00,00,00,226,255,255,255,
Task 2
Task 3
CAN0 SID=0x0116(04 Bytes): 143,05,00,00,
Task 0
CAN0 SID=0x0116(04 Bytes): 144,05,00,00,
```

Figure1-6 Programming Results

If connecting interfaces CANFD0~CANFD3 to one CANFD bus and run this code, you will obtain the information shown in Figure 1-7. Task0~Task3 indicates that tasks are being

executed and CANFD0~CANFD3 indicates that tasks are being sent. The CAN0~CAN3 indicates the messages received by the corresponding CANFD interface. After each CANFD sends messages, the other three CANFDs will receive messages.

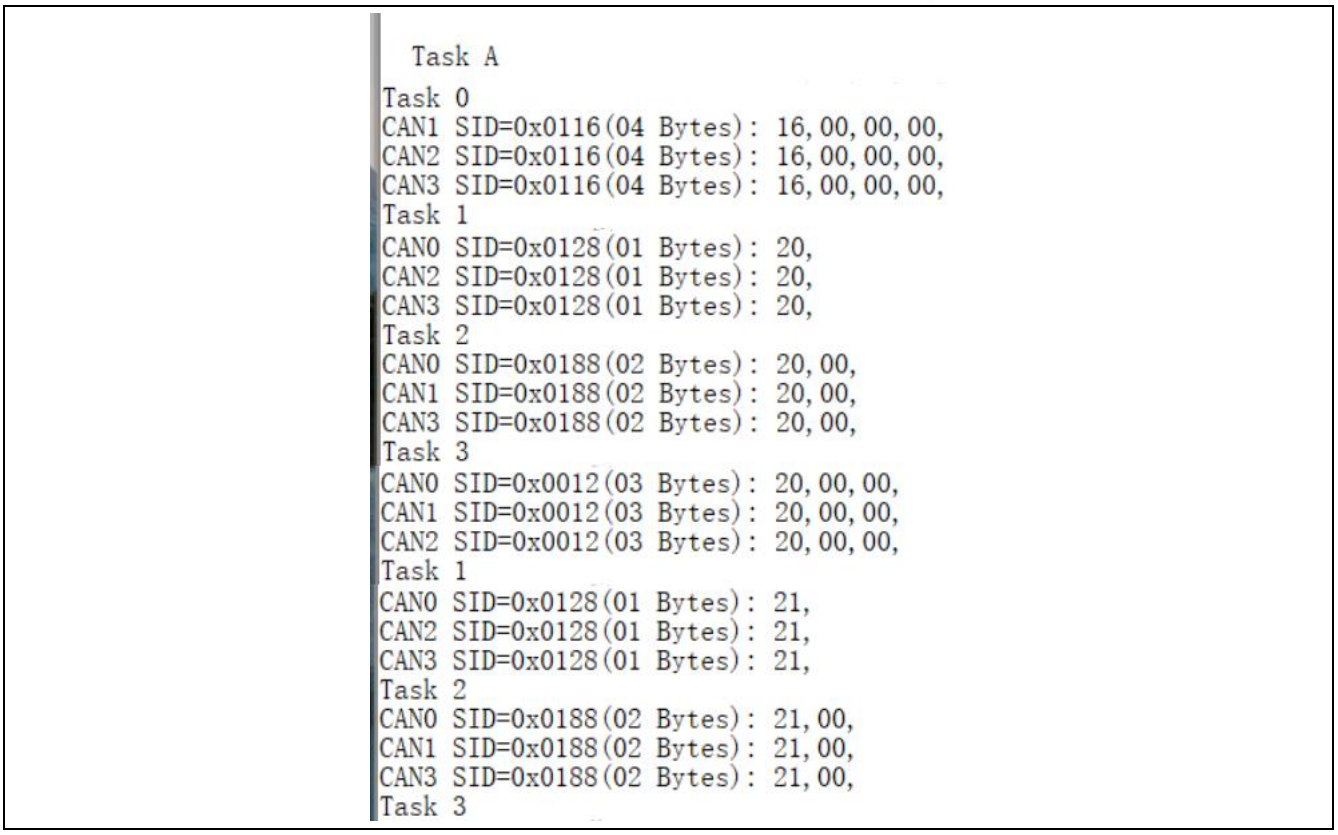

Figure 1-7 Printed Result of 4-CANFD Transceivers Linked Together

2. Code Description

This example code has ported uCOSii. For the contents, please refer to EC_M480_uCOS_II_Porting_Readme that can be downloaded at https://www.nuvoton.com.cn/resource-download.jsp?tp_GUID=EC012022101106314002.
This code establishes 6 tasks. task_Highest() is the task with the highest priority in which system beat timer must be configured and started at the beginning. UART0 printing is also in this task. If other tasks need UART0 printing information, the data to be printed must be sent through queue Q to avoid conflicts in accessing UART0.

The task0~task3 control CANFD0~CANFD3 to send messages to the CANFD bus.

The CANFD0~CANFD3 will send a semaphore (semaphore plus 1) in the corresponding interrupt as soon as a message is received. Outside the interrupt, the semaphore will be checked and the message will be read out from the corresponding FIFO1. In order to get a simple routine, all printing messages will be printed in task_A. For the messages to be printed in time, task_A generally has a higher priority than other tasks that need to print information through UART0.

The pin function configuration, communication rate, and send/receive initialization code for CANFD are configured at the beginning of the main() function. Whether to send CAN packets or CANFD packets is also set here.

```
//== Configure CANFD0 ====================================
    CANFD0->CCCR = CANFD_CCCR_CCE_Msk | CANFD_CCCR_INIT_Msk | CANFD_CCCR_BRSE_Msk |
CANFD_CCCR_FDOE_Msk;

#ifndef  __M467SJHAE            // If test this project on NuMaker-M467HJ board
    SET_CAN0_RXD_PJ11();        // Set PJ multi-function pins for CAN FD0 RXD and TXD
    SET_CAN0_TXD_PJ10();
#else                           // If test this project on M467SJHAE board
    SET_CAN0_RXD_PB10();
    SET_CAN0_TXD_PB11();
#endif
    CANFD_BitRate_Init(CANFD0);
                        // 32-Txbuf,StartAddress=0xE0 Please reference CANFD_TxBuff[32]
    CANFD_Tx_Init(CANFD0, 32, 0xE0);
    CANFD0_RX_Initial(CANFD0);
//    CANFD0->CCCR = CANFD_CCCR_BRSE_Msk | CANFD_CCCR_FDOE_Msk ;            // CANFD frame
    CANFD0->CCCR = 0 ;                                 // CAN frame,Write is OK after 2 CLKs
```

Correctly configuring the sampling point for reception is the key to improve the reliability of CAN or CANFD communication. Considering the transmission time, DTSG1 is generally longer than DTSG2. This is configured by the function CANFD_BitRate_Init(). DTSG1 is configured to occupy 75% of a bit-time in this code as shown Figure 2-1.

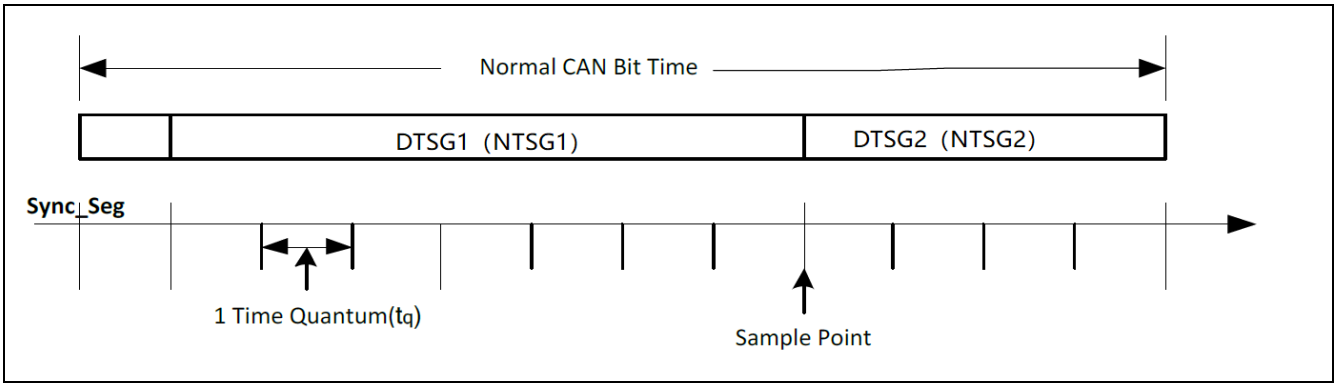

Figure 2-1 Sample Point of One Bit

```
void CANFD_BitRate_Init(CANFD_T *psCanfd)
{
    // CANFD_CLk = 200M/10 = 20MHz       Baud Rate = 20M/20 = 1Mbps

    psCanfd->NBTP = (3 << 25) +        // NSJW  = 3+1 =4 CLK
                    (0 << 16) +        // NBRP  = 0+1 =1       // prescaler =1
                    (13 << 8) +        // NTSG1 = 13+1 =14 CLK
                    (5 - 1) ;          // NTSG2 = 5 CLK       // One bit =1+14+5 =20 CLK

// If CANFD frame and permit data-rate change, define date rate as following.

//    psCanfd->DBTP = ((1 - 1) << 16) +      // DBRP  = 1 prescaler
//                    ((15 - 1) << 8) +      // DTSG1 = 15 CLK
//                    ((5 - 1) << 4) +       // DTSG2 = 5 CLK   // One bit = 15+5 = 20 CLK
//                    (4 - 1) ;              // DSJW  = 4 CLK
}
```

The CANFD_Tx_Init() function sets the first address of the sending buffer. Each buffer configured can store a maximum of 64 bytes. The actual number of bytes to be sent is defined when the packet is sent.

```
void CANFD_Tx_Init(CANFD_T *psCanfd, int16_t TxBuff_Quantity, uint16_t TxBuff_StartAddr)
{
    psCanfd->TXBC = (TxBuff_Quantity << CANFD_TXBC_NDTB_Pos)
                    + TxBuff_StartAddr ;            // Start address of messages RAM

    psCanfd->TXESC = 7 << CANFD_TXESC_TBDS_Pos;     // 7 mean 64 Bytes
}
```

The function CANFD0_RX_Initial() is the CANFD0 reception configuration. At first, the message buffer is configured to hold 64 bytes. Then the offset address of the SID and XID are configured in the dedicated RAM area. The code in ”#if 1” is the configuration code to store reception messages in FIFO1 and refuse remote frames. The #else code is a few examples to only accept some SID or XID values message.

```
/*---------------------------------------------------------------------------*/
/*                        Init CANFD0 Rx                                     */
/*---------------------------------------------------------------------------*/
```

```
void CANFD0_RX_Initial(CANFD_T *psCanfd)
{
    // 0=>8Byte, 1=>12Byte, 2=>16Byte, 3=>20Byte,4=>24Byte,5=>32Byte,6=>48Byte, 7=>64Byte
    psCanfd->RXESC = (psCanfd->RXESC & (~CANFD_RXESC_F1DS_Msk)) | (7 <<
CANFD_RXESC_F1DS_Pos);                                          // 7 mean MAX 64 Bytes

    psCanfd->RXF1C = (30 << CANFD_RXF1C_F1WM_Pos)          // watermark = 60 messages
                   | (32 << CANFD_RXF1C_F1S_Pos)           // FIFO1 can hold 64 messages
                   | 0x9E0 ;                  // FIFO1 start address =0x40020000 +0x9E0

    psCanfd->SIDFC = (24 << 16) + 0 ;                             // 24-SID at 0  address
    psCanfd->XIDFC = (16 << 16) + 0x60 ;                         // 16-XID at 0x60 address

#if 1
    CANFD0_SID_Buff[0].VALUE = CANFD_RX_FIFO1_STD_MASK(0, 0) ; // receive all SID messages

    CANFD0_XID_Buff[0].LOWVALUE  =CANFD_RX_FIFO1_EXT_MASK_LOW(0) ;
    CANFD0_XID_Buff[0].HIGHVALUE=CANFD_RX_FIFO1_EXT_MASK_HIGH(0);  //receive all XID

    CANFD0->GFC = 3 ;                                          // reject remote frames

#else
    CANFD0_SID_Buff[0].VALUE = CANFD_RX_FIFO1_STD_MASK(0x110, 0x7F0) ;   // SID, Mask
    CANFD0_SID_Buff[1].VALUE = CANFD_RX_FIFO1_STD_MASK(0x22F, 0x7FF) ;   // SID, Mask
    CANFD0_SID_Buff[2].VALUE = CANFD_RX_FIFO1_STD_MASK(0x333, 0x7FF) ;   // SID, Mask

    CANFD0_XID_Buff[0].LOWVALUE  = CANFD_RX_FIFO1_EXT_MASK_LOW(0x220) ;       // XID
    CANFD0_XID_Buff[0].HIGHVALUE = CANFD_RX_FIFO1_EXT_MASK_HIGH(0x1FFFFFF0) ; // MASK

    CANFD0_XID_Buff[1].LOWVALUE  = CANFD_RX_FIFO1_EXT_MASK_LOW(0x3333) ;      // XID
    CANFD0_XID_Buff[1].HIGHVALUE = CANFD_RX_FIFO1_EXT_MASK_HIGH(0x1FFFFFFF) ; // MASK

    CANFD0_XID_Buff[2].LOWVALUE  = CANFD_RX_FIFO1_EXT_MASK_LOW(0x44444) ;     // XID
    CANFD0_XID_Buff[2].HIGHVALUE = CANFD_RX_FIFO1_EXT_MASK_HIGH(0x1FFFFFFF) ; // MASK

    CANFD0->GFC = 0x20 + 0x08 + 3 ;  // reject all no-match SID &XID, reject remote frames

    //  CANFD0->GFC = 0x10 + 0x04 + 3 ;     //no-match ID into FIFO1. reject remote frames

#endif

    CANFD_EnableInt(CANFD0, (CANFD_IE_TOOE_Msk | CANFD_IE_RF1NE_Msk), 0, 0, 0);
    NVIC_EnableIRQ(CANFD00_IRQn);
}
```

This code directly organizes the message data in the dedicated RAM area of CANFD which can only be "written by words" but not by bytes. Therefore, the address of the dedicated RAM area should be configured in IRAM2 of the compiler Option as shown in Figure 2-2.

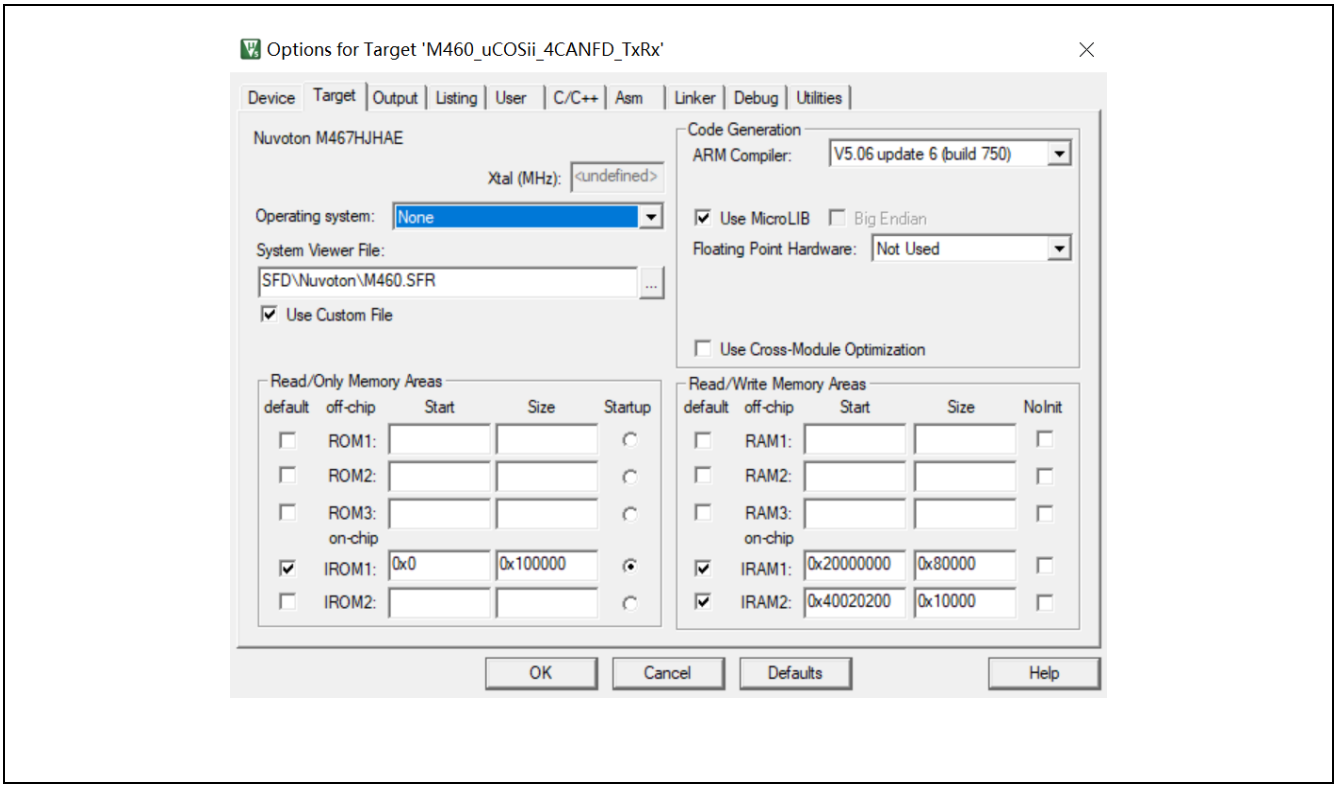

Figure 2-2 Option of MDK

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M460_Series_BSP_CMSIS_V3.00.002
- IDE version
  - Keil uVersion 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M467HJ V1.0
- Pin Connect
  - Connect the CANFD bus of two boards together to display the result of the example code execution as Figure 3-1.

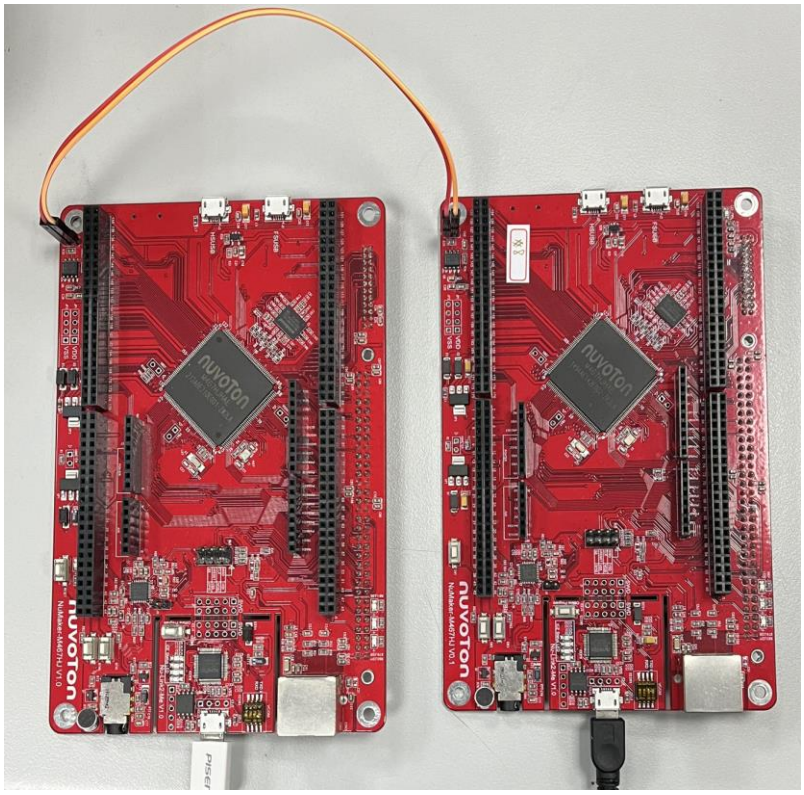

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M460_uCOSii_4CANFD_TxRx_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
uCOS_II	
Config	Configuration files of uCOSii for this project
Ports	Porting files of uCOSii for M460 series MCU
Source	Source file of uCOSii 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 *M460_uCOSii_4CANFD_TxRx.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.11.28	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.