M460 CAN FD Transceiver Delay Compensation

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code describes how to use the M460 CAN FD transceiver delay compensation function.
BSP version	M460_Series_BSP_CMSIS_V3.00.002
Hardware	NuMaker-M476HJ_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 describes the transceiver delay compensation function of CAN FD. To ensure that CAN FD is sending messages at high speed, the transmitter is prevented from entering the BUSOFF.

Transceiver Delay Compensation (TDC) mainly solves the problem of the physical delay of the transceiver loop on the reliability verification of signal read back.

To facilitate the understanding of the principle of TDC, this example first introduces the TDC function from the following points, and then introduces the code configuration method:

- What is a delay in sending and receiving?
- Why to use transceiver delay compensation?
- How to set the TDC?
- Case Analysis

This example code needs to be provided with a CAN FD debugging tool to observe the data sent and received. In this example, there is already a capture diagram of the communication signal, and if you need to verify it, it is recommended to have an oscilloscope for observation.

1.1 What is a Delay in Sending and Receiving?

When digital communication is proceeded, the signal is transmitted through the TTL level of the MCU, and the single data is not suitable for long-distance and high-reliability transmission. Therefore, it is necessary to do some transformation of the signal to make the converted signal suitable for long-distance reliable communication.

In this document, CAN differential signal is used for symmetrical transmission. Figure 1-1 is a schematic representing the internal structure and transmission delay of the CAN converter chip in this document:

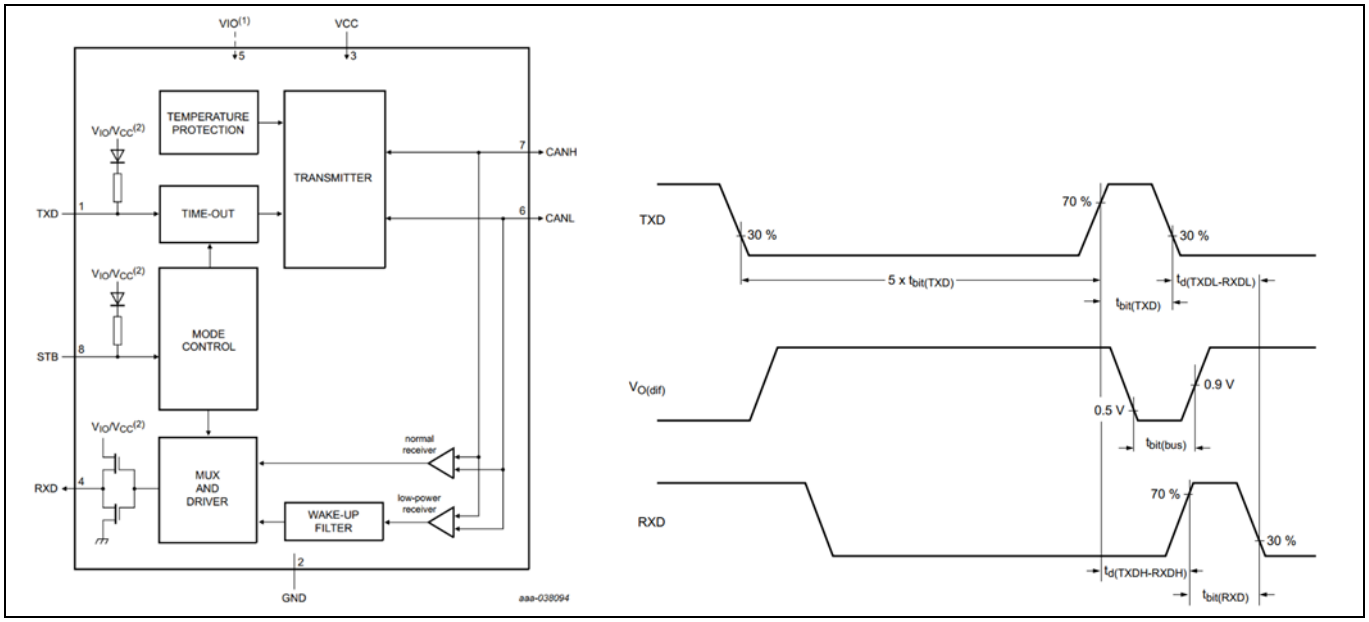


Figure 1-1 Internal Structure and Transmission Delay

TXD is the signal sender, Vo (dif) is the physical signal of the external bus, and RXD is the signal receiver.

Figure 1-1 is only the delay of the converter, and in the harsh communication environment, a signal isolator will be added to protect the controller, so as not to interfere with the operation of the controller with external noise. However, it will also bring a longer signal delay.

Figure 1-2 is the propagation delay time for the isolator used in this example:

Parameter	Symbol	Min	Typ	Max	Unit	Test Conditions
ADuM120xBR						
Minimum Pulse Width[2]	PW			100	ns	C_L = 15 pF, CMOS signal levels
Maximum Data Rate[3]		10			Mbps	C_L = 15 pF, CMOS signal levels
Propagation Delay[4]	t_{PHL}, t_{PLH}	20		60	ns	C_L = 15 pF, CMOS signal levels
Pulse-Width Distortion, $\|t_{PLH} - t_{PHL}\|$[4]	PWD			3	ns	C_L = 15 pF, CMOS signal levels
Change Versus Temperature			5		ps/°C	C_L = 15 pF, CMOS signal levels
Propagation Delay Skew[5]	t_{PSK}			22	ns	C_L = 15 pF, CMOS signal levels
Channel-to-Channel Matching, Codirectional Channels[6]	t_{PSKCD}			3	ns	C_L = 15 pF, CMOS signal levels
Channel-to-Channel Matching, Opposing Directional Channels[6]	t_{PSKOD}			22	ns	C_L = 15 pF, CMOS signal levels
Output Rise/Fall Time (10% to 90%)	t_R/t_F		3.0		ns	C_L = 15 pF, CMOS signal levels
ADuM120xAR						
Minimum Pulse Width[2]	PW			1000	ns	C_L = 15 pF, CMOS signal levels
Maximum Data Rate[3]		1			Mbps	C_L = 15 pF, CMOS signal levels
Propagation Delay[4]	t_{PHL}, t_{PLH}	50		150	ns	C_L = 15 pF, CMOS signal levels
Pulse-Width Distortion, $\|t_{PLH} - t_{PHL}\|$[4]	PWD			40	ns	C_L = 15 pF, CMOS signal levels
Propagation Delay Skew[5]	t_{PSK}			100	ns	C_L = 15 pF, CMOS signal levels
Channel-to-Channel Matching[6]	$t_{PSKCD/OD}$			50	ns	C_L = 15 pF, CMOS signal levels
Output Rise/Fall Time (10% to 90%)	t_R/t_F		10		ns	C_L = 15 pF, CMOS signal levels

Figure 1-2 Transmission Delay of the Isolator

It can be seen from Figure 1-2 that 10 Mbps (20~60 ns) nearly doubled the time of the isolator delay.

In summary, there are three types of latency in practical applications:

- Transceiver loop delay (the transceiver receives the transmitted data and then passes it back to the controller)
- Sending Delay (Sending Isolation)
- Receive Delay (Receive Isolation)

1.2 Why to Use Transceiver Delay Compensation?

At the beginning of the design of the CAN bus, it puts forward the requirements of high performance and reliability for its communication, so it is stipulated in the CAN protocol that when the sender sends data, it needs to detect whether the data sent back is consistent with the transmission, and if it is inconsistent, an error frame will be generated, which is to judge whether the signal is successfully sent by read back.

After the CAN detects that the backtracking of the sent information is incorrect and reaches a certain number, the controller will consider the bus abnormal and enter the BUSOFF state, and suspend itself without affecting the communication of other controllers on the bus.

In the white paper of the classic CAN ISO11898-5, the total loop delay is specified to be 255 ns. This rule ensures that the loop delay does not affect the data read back in low-speed mode. However, when the CAN FD mode communicates in the 5Mbps data segment, the actual bit transmission time is only 200 ns, and if the transmission and reception delay is not considered, the bus will not be able to send and receive data normally. Also in the analysis of section 1.1, the stacking of several types of latency is likely to exceed 255 ns in real applications.

To solve the above problems, BOSCH has designed a function called Transceiver Delay Compensation (TDC), which can automatically measure the delay time in the circuit and reset the signal sampling point position of the data segment through the delay time. The location of this sampling point is defined as the Secondary Sample Point (SSP).

Figure 1-3 demonstrates the calculation methods of automatic delay and SSP sampling points.

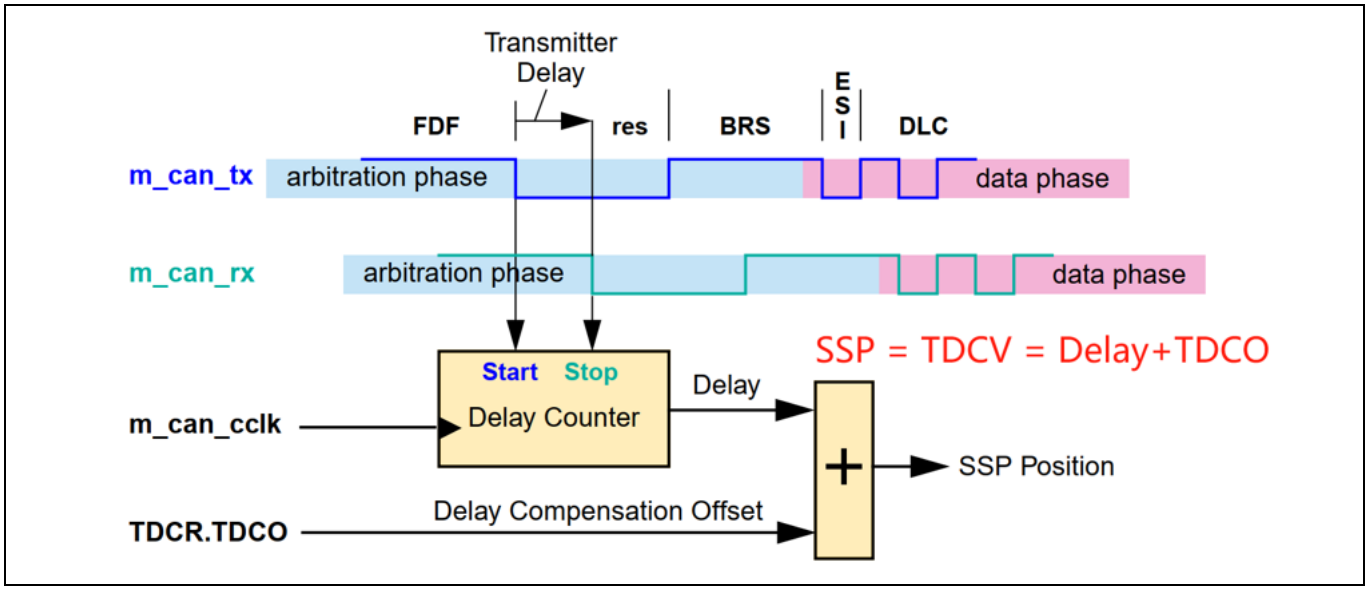


Figure 1-3 Schematic Diagram of the Automatic Calculation of the SSP

1.3 How to Set TDC?

TDC is mainly used to set the SSP. This function involves 3 registers and 4 parameter values. TDCF, TDCO and TDCV are all counted in the unit of minimum time tq, and the calculation method is the same as that of the baud rate register.

- TDC bit in the DBTP register:

  This bit indicates that the delay sampling function is enabled, and it is recommended to enable it when the communication speed is greater than 1 Mbps.

- TDCO and TDCF in TDCR registers: secondary sampling point SSP = Delay + TDCO.

  TDCO: The delay offset of the second sampling, which is generally set by referring to the DTSEG1 value.

  TDCF: Minimum secondary sampling point. At the same time, this value must be greater than TDCO.

  If you do not know the specific delay parameters, you can default TDCF = DTSEG1+1 to ensure that TDCF > TDCO.

  TDCF can make some adjustments according to the actual hardware test situation, and the physical delay will be determined after the hardware is confirmed.

- TDCV in PSR register: Automatically calculated delay sampling point TDCV = TDCO + read back physical delay (Delay).

  When TDCO = 0, what is actually measured is the physical delay of read back.

  When TDCF ≥ TDCV, the secondary sampling point SSP = TDCF.

  When TDCF < TDCV, the secondary sampling point SSP = TDCV.

1.4 Case Analysis

1.4.1 Waveform Analysis

Experimental Environment:

- 10 Mbps isolation chip ADUM1201BRZ (latency 20 ~ 60 ns).
- 1 Mbps isolation chip ADUM1201ARZ (latency 50 ~ 150 ns).
- CAN transceiver: TJA1462A
- MCU control: M467HJHAN

The latency data for the isolator can be found in Figure 1-2.

Delay data TJA1462A the CAN transceiver in Figure 1-4:

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$t_{d(TXDL-RXDL)}$	delay time from TXD LOW to RXD LOW	Normal mode	-	-	190	ns
		Normal mode; V_{CC} = 4.5 V to 5.5 V	-	-	255	ns
$t_{d(TXDH-RXDH)}$	delay time from TXD HIGH to RXD HIGH	Normal mode	-	-	190	ns
		Normal mode; V_{CC} = 4.5 V to 5.5 V	-	-	255	ns

Figure 1-4 CAN Transceiver Delay Data

Experimental Methods:

The M467 evaluation board's CAN sends a frame of data, uses the oscilloscope to connect the M467CAN RXD and TXD with two channels respectively, measures the TXD and RXD signals at the same time when sending the data, and records the delay difference between the 1 Mbps isolation and the 10 Mbps isolation.

Figure 1-5 is one frame of data sent and received for the 10 Mbps isolation chip CAN FD.

Standard frame, address 0x000, data length 12 bytes, variable baud rate.

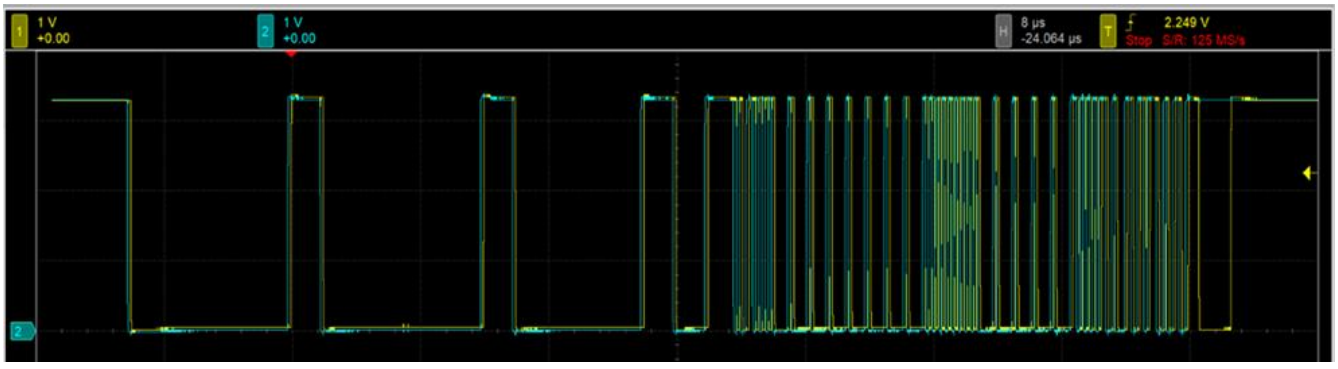


Figure 1-5 10 Mbps Isolated CAN FD Transmit and Receive Data Waveform in One Frame

Measuring the TXD and RXD pins on the MCU side, it is obvious that the TXD (blue line) and RXD (yellow line) signals are superimposed.

Amplify the waveform of one bit of the data segment as Figure 1-6 :

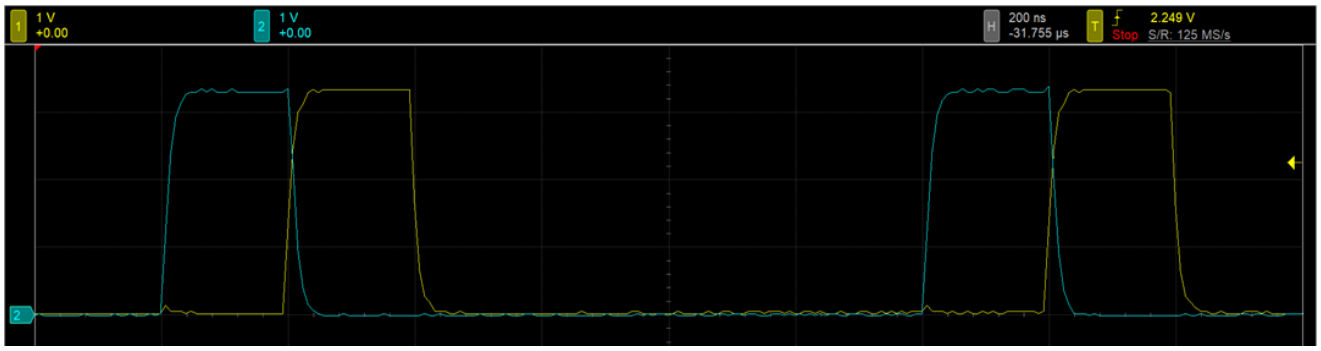


Figure 1-6 10 Mbps Isolated Data Segment One-bit Transceiver Waveform

From Figure 1-6_it can be clearly seen that the RXD(yellow line)is delayed by about 200 ns from the TXD(blue line) to the MCU side, and if the sampling point of the transmission timing is still used at this time, the signal will not be sampled correctly. At this time, the TDC function is very important.

When using a 1 Mbps isolation chip, the delay test of one bit waveform in the data segment is as Figure 1-7:

ADUM1201ARZ (delay 50 ~ 150 ns); the CAN transceiver is also TJA1462A.

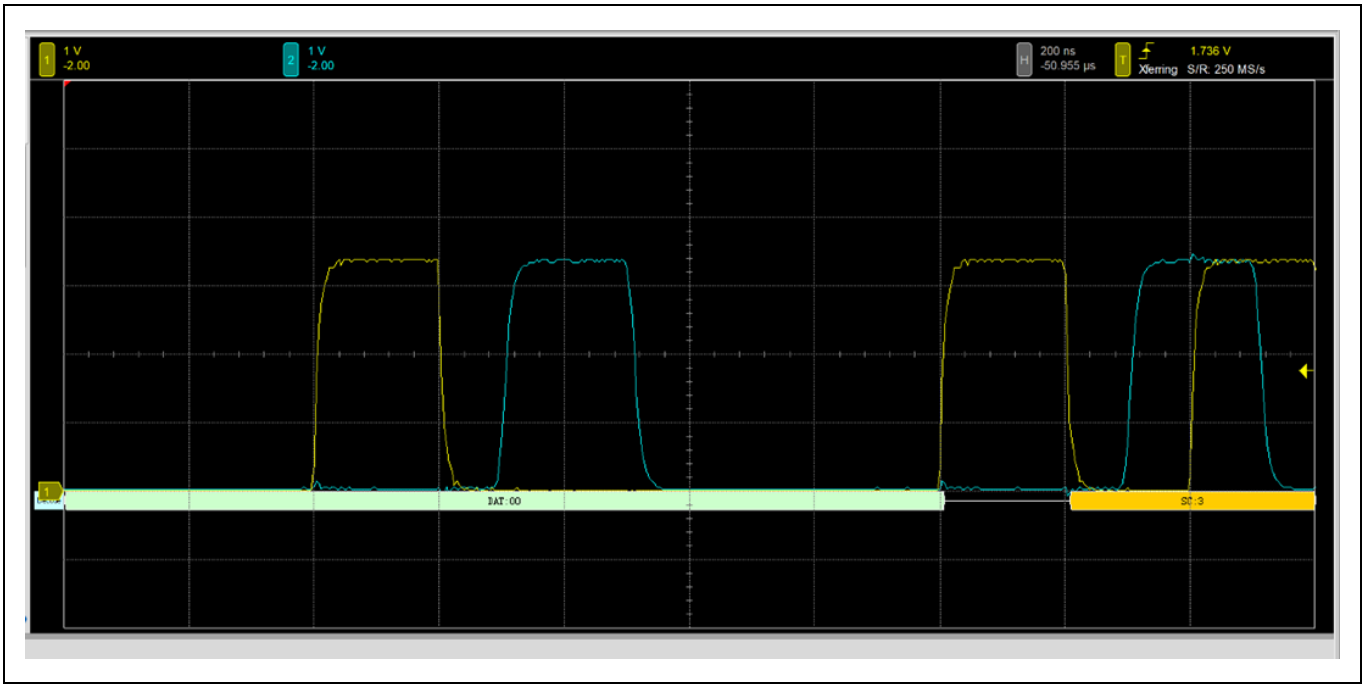


Figure 1-7 1 Mbps Isolate the Transmitting and Receiving Waveform of One-bit of the Data Segment

From Figure 1-7 1 Mbps Isolate the Transmitting and Receiving Waveform of One-bit of the Data Segment can be seen that the 1 Mbps isolated RXD (blue line) is delayed by nearly 300ns compared to the TXD (yellow line), and the delay performance of different isolators varies greatly.

When the hardware design is completed, the physical delay time is basically determined, and the accurate delay time can be confirmed with the oscilloscope, that is, the value of TDCF can be confirmed.

1.4.2 Example of TDC Setting

The instance observes the settings of the actual TDC register in Keil through the Nulink screenshot and verifies whether the calculation process is correct.

The baud rate of the arbitration segment is 500 kbps, the data segment is 5 Mbps, and the baud rate register of the data segment:

Total Tq = (DTSEG1+1) + (DTSEG2+1) + DSJW = (5 + 1) + (2 + 1) + 1 = 10tq

The time of one bit is a total Tq, and the total Tq is split into 10 elements, and these 10 elements are used to set the sampling points.

The actual register settings are as Figure 1-8

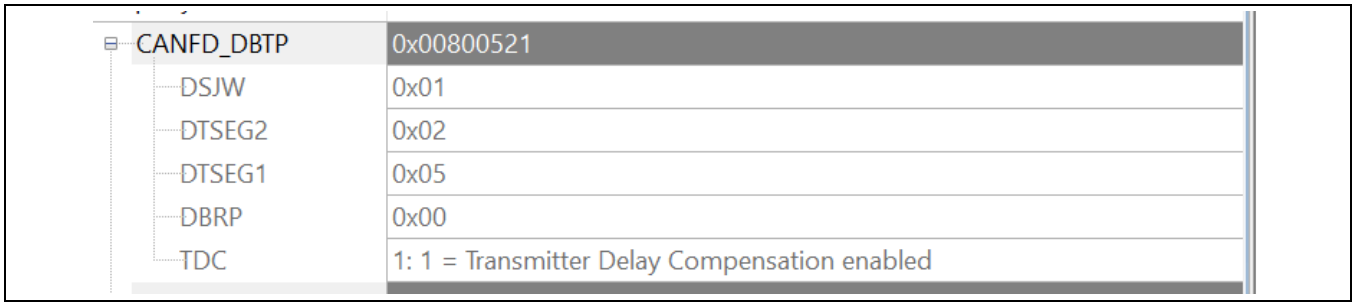
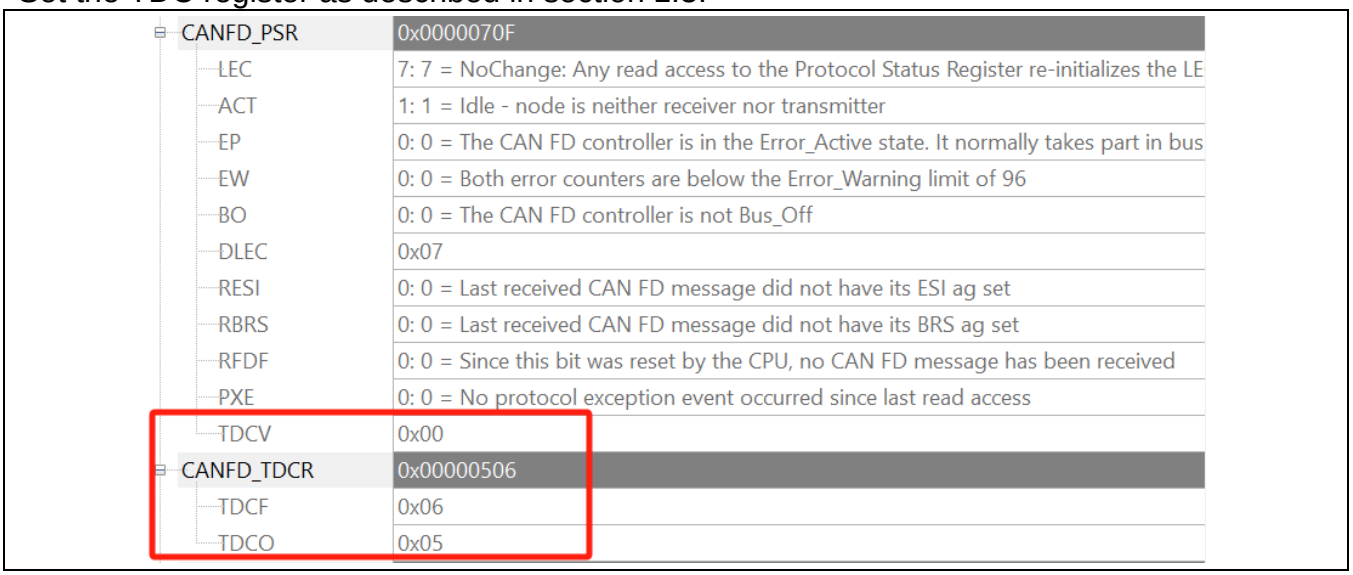


Figure 1-8 CAN FD Data Segment Baud rate Setting Register

Set the TDC register as described in section 1.3:



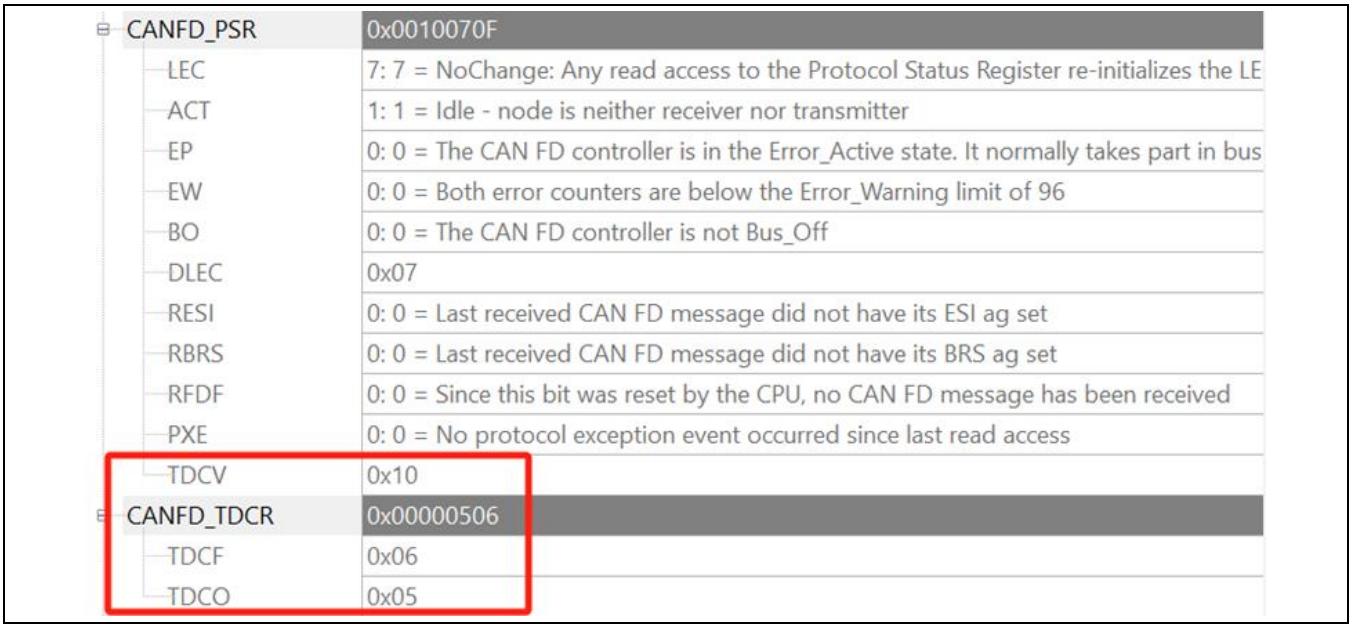
CANFD_PSR	0x0000070F
LEC	7: 7 = NoChange: Any read access to the Protocol Status Register re-initializes the LE
ACT	1: 1 = Idle - node is neither receiver nor transmitter
EP	0: 0 = The CAN FD controller is in the Error_Active state. It normally takes part in bus
EW	0: 0 = Both error counters are below the Error_Warning limit of 96
BO	0: 0 = The CAN FD controller is not Bus_Off
DLEC	0x07
RESI	0: 0 = Last received CAN FD message did not have its ESI ag set
RBRS	0: 0 = Last received CAN FD message did not have its BRS ag set
RFDF	0: 0 = Since this bit was reset by the CPU, no CAN FD message has been received
PXE	0: 0 = No protocol exception event occurred since last read access
TDCV	0x00
CANFD_TDCR	0x00000506
TDCF	0x06
TDCO	0x05

Figure 1-9 TDC-related Register Status after CAN FD Initialization

CAN initialization has not sent data at this time:

TDCV = 0; TDCO = DTSEG1; TDCF = DTSEG1 + 1.

After the CAN sends the data once, the bus automatically measures the delay and stores the measured value in the TDCV register.



CANFD_PSR	0x0010070F
LEC	7: 7 = NoChange: Any read access to the Protocol Status Register re-initializes the LE
ACT	1: 1 = Idle - node is neither receiver nor transmitter
EP	0: 0 = The CAN FD controller is in the Error_Active state. It normally takes part in bus
EW	0: 0 = Both error counters are below the Error_Warning limit of 96
BO	0: 0 = The CAN FD controller is not Bus_Off
DLEC	0x07
RESI	0: 0 = Last received CAN FD message did not have its ESI ag set
RBRS	0: 0 = Last received CAN FD message did not have its BRS ag set
RFDF	0: 0 = Since this bit was reset by the CPU, no CAN FD message has been received
PXE	0: 0 = No protocol exception event occurred since last read access
TDCV	0x10
CANFD_TDCR	0x00000506
TDCF	0x06
TDCO	0x05

Figure 1-10 TDC-related Register Status after Sending Data

After sending data: TDCV = 16; TDCO = DTSEG1; TDCF = DTSEG1 + 1.

Data segment 5 Mbps calculated based on waveform and parameters. (Verify the calculation method described in section 1.3):

Total Tq = 10tq = 200ns (5 Mbps) for the data segment and 1 tq = 20ns for a single element .

TDCO= DTSEG1 = 5tq.

TDCF = DTSEG1 + 1 = 6tq.

When 10 Mbps isolation is used, the location of the secondary sampling point is (at 320 ns):

TDCV = 16tq

= 11tq（delay） + 5tq（TDCO）

= 11*20ns + 5*20ns

= 320ns

In this case, the TDCV(16tq) > TDCF(6tq) secondary sampling points are calculated according to the TDCV, as shown in Figure 1-11:

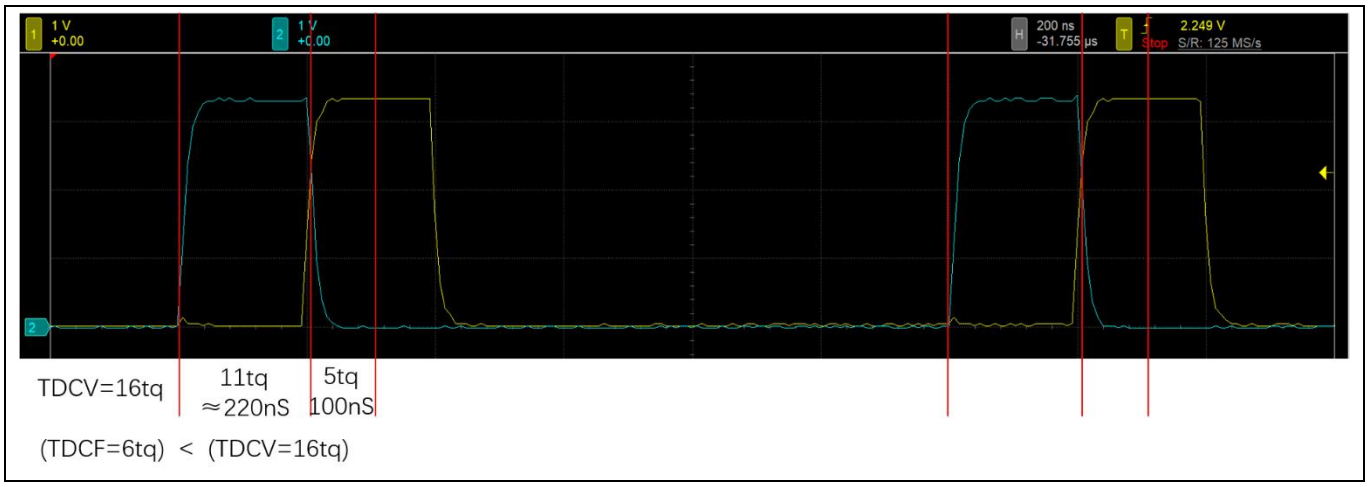


Figure 1-11 Secondary Sampling Point of CAN FD

In Figure 1-11 if the secondary sampling point falls on the red line at the end of the yellow line, it can be seen that TDCV = 16tq can correctly sample the bit level of the backhaul. After the hardware is finalized, a suitable value can be selected to set the TDCF through the measurements in Figure 1-11.

Note:

- There is a slight error between the TDCV measurements and the oscilloscope's observations.
- Oscilloscope 1 is 200 ns, and the actual sampling point can be correctly sampled from 280 ns to 360 ns (14≤TDCF≤18).
- The blue line is TXD and the yellow line is RXD.

2 Code Description

First, check whether the baud rate of the routine is set to 5 Mbps, and change the baud rate in *the CANFD_Init*() function in the *main.c* file.

```
/*---------------------------------------------------------------------------*/
/*                          Init CAN FD0                                     */
/*---------------------------------------------------------------------------*/
void CANFD_Init(void)
{
    printf("+-----------------------------------------------------------------+\n");
    printf("|     Pin Configure                                               |\n");
    printf("+-----------------------------------------------------------------+\n");
    printf("|    CAN_TXD                                 CAN_TXD(Any board)   |\n");
    printf("|    CAN_RXD                                 CAN_RXD(Any board)   |\n");
    printf("|          |-----------| CANBUS  |-----------|                     |\n");
    printf("|  ------> |           |<------->|           |<------              |\n");
    printf("|    CAN_TX|   CANFD   | CANFD_H |    CANFD  |CAN_TX               |\n");
    printf("|          |Transceiver|         |Transceiver|                     |\n");
    printf("|  <------ |           |<------->|           |------>              |\n");
    printf("|    CAN_RX|           | CANFD_L |           |CAN_RX               |\n");
    printf("|          |-----------|         |-----------|                     |\n");
    printf("|                                                                 |\n");
    printf("+-----------------------------------------------------------------+\n\n");

    /*Get the CAN FD configuration value*/
    CANFD_GetDefaultConfig(&sCANFD_Config, CANFD_OP_CAN_FD_MODE);
    sCANFD_Config.sBtConfig.sNormBitRate.u32BitRate =  500000;// NormBitRate = 500Kbps
    sCANFD_Config.sBtConfig.sDataBitRate.u32BitRate = 5000000;// DataBitRate = 5Mbps

    /*Open the CAN FD feature*/
    CANFD_Open(g_pCanfd, &sCANFD_Config);
......
```

The TDO function of CAN FD, the code is located in *canfd.c*, and the TDC function is automatically enabled when the DataBitRate is greater than 1 Mbps.

```
/* @brief       Open TDC Function
 *
 * @param[in]   psCanfd     The pointer to CAN FD module base address.
 * @param[in]   psCanfdStr  message ram setting and bit-time setting
 *
 * @return      None.
 *
 * @details     Open Transceiver Delay Compensation
 *               Secondary sampling point  = SSP = TDCV = Delay + TDCO;
 */
void CANFD_Open_TDC(CANFD_T *psCanfd, CANFD_FD_T *psCanfdStr)
{
    uint8_t tdco;//TDCO = DTSEG1 + 1

   if(psCanfdStr->sBtConfig.sDataBitRate.u32BitRate> 1000000)//bitrate > 1Mbps
     {
```

```
    //Get minimum tdco
    tdco = (((psCanfd->DBTP) & CANFD_NBTP_NTSEG1_Msk)>>CANFD_NBTP_NTSEG1_Pos);

    //TDCO need to measure reference DBTP:NTSEG1
    psCanfd->TDCR  = (psCanfd->TDCR & (~CANFD_TDCR_TDCO_Msk)) |
                            (tdco << CANFD_TDCR_TDCO_Pos);

    //TDCF mast greater than TDCO,so set TDCF = TDCO + 1
    psCanfd->TDCR  = (psCanfd->TDCR & (~CANFD_TDCR_TDCF_Msk)) |
                         ((tdco + 1) << CANFD_TDCR_TDCF_Pos);

    //Open compensation function
    psCanfd->DBTP  |= CANFD_DBTP_TDC_Msk;

     }
}
```

This function needs to be placed in the *CANFD_Open (CANFD_T *psCanfd, CANFD_FD_T *psCanfdStr)* function to determine the baud rate after initializing the CAN FD clock (CANFD_CalculateTimingValues).

```
/**
 * @brief       Config message ram and Set bit-time.
 *
 * @param[in]   psCanfd     The pointer to CAN FD module base address.
 * @param[in]   psCanfdStr  message ram setting and bit-time setting
 *
 * @return      None.
 *
 * @details     Converts a Data Length Code into a number of message bytes.
 */
void CANFD_Open(CANFD_T *psCanfd, CANFD_FD_T *psCanfdStr)
{
    uint32_t u32RegLockLevel = SYS_IsRegLocked();

    if (u32RegLockLevel)
        SYS_UnlockReg();

    if (psCanfd == (CANFD_T *)CANFD0)
    {
        CLK_EnableModuleClock(CANFD0_MODULE);
        SYS_ResetModule(CANFD0_RST);
    }
    else if (psCanfd == (CANFD_T *)CANFD1)
    {
        CLK_EnableModuleClock(CANFD1_MODULE);
        SYS_ResetModule(CANFD1_RST);
    }
    else if (psCanfd == (CANFD_T *)CANFD2)
    {
        CLK_EnableModuleClock(CANFD2_MODULE);
        SYS_ResetModule(CANFD2_RST);
    }
    else if (psCanfd == (CANFD_T *)CANFD3)
    {
        CLK_EnableModuleClock(CANFD3_MODULE);
```

```
        SYS_ResetModule(CANFD3_RST);
    }
    else
    {
        if (u32RegLockLevel)
            SYS_LockReg();

        return;
    }

    /* configuration change enable */
    psCanfd->CCCR  = CANFD_CCCR_INIT_Msk;
    psCanfd->CCCR |= CANFD_CCCR_CCE_Msk;

    if(psCanfdStr->sBtConfig.bBitRateSwitch)
    {
        /* enable FD and baud-rate switching */
        psCanfd->CCCR |= CANFD_CCCR_BRSE_Msk;
    }

    if(psCanfdStr->sBtConfig.bFDEn)
    {
        /*FD Operation enabled*/
        psCanfd->CCCR |= CANFD_CCCR_FDOE_Msk;
    }

    /*Clear the Rx Fifo0 element setting */
    psCanfd->RXF0C = 0;
    /*Clear the Rx Fifo1 element setting */
    psCanfd->RXF1C = 0;

    /* calculate and apply timing */
    if(CANFD_CalculateTimingValues(psCanfd,
         psCanfdStr->sBtConfig.sNormBitRate.u32BitRate,
         psCanfdStr->sBtConfig.sDataBitRate.u32BitRate,
         SystemCoreClock, &psCanfdStr->sBtConfig.sConfigBitTing))
    {
        CANFD_SetTimingConfig(psCanfd, &psCanfdStr->sBtConfig.sConfigBitTing);
    }

     /* open TDC function */
     CANFD_Open_TDC(psCanfd,psCanfdStr);
    /* 7=>Default TX maximum data length(64Byte) */
     psCanfd->TXESC = (7 << CANFD_TXESC_TBDS_Pos);
    /* 7=>Default RX maximum data length(64Byte) */
    psCanfd->RXESC = (psCanfd->RXESC & (~CANFD_RXESC_F1DS_Msk)) |
                                    (7 << CANFD_RXESC_F1DS_Pos);


    if (u32RegLockLevel)
        SYS_LockReg();
......
```

Send CAN data through the prompt of the serial port, press the number 8 to select the frame to be sent, and you can see that the data can be sent out normally:

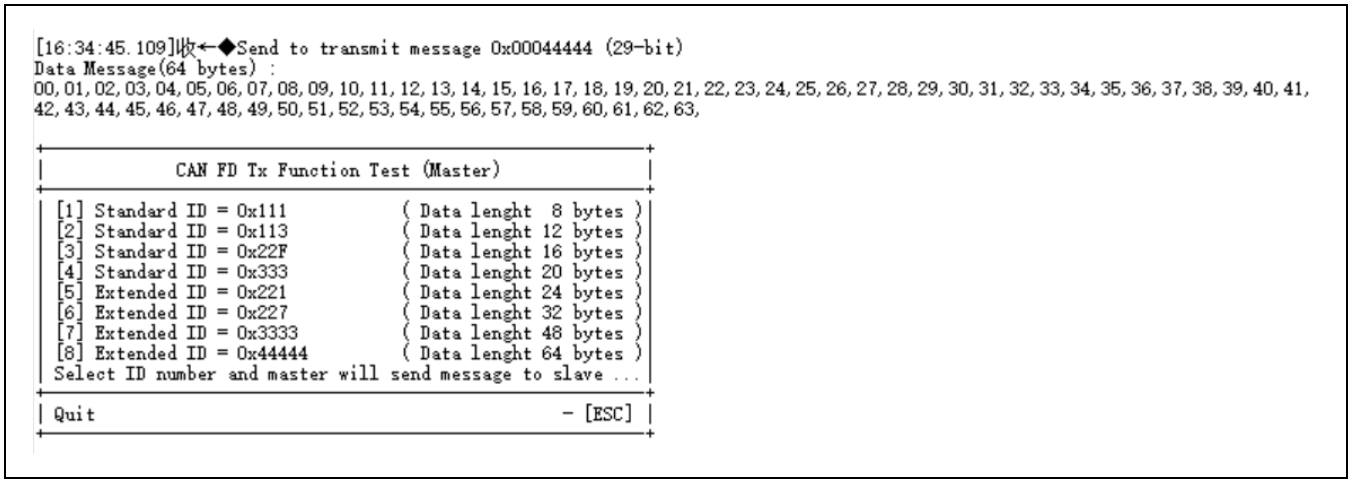


Figure 2-1 Routine CAN Sends One Frame of Data

In Figure 2-1, the data frame of CANFD sequence number 8 is:

Extended ID=0x44444 variable speed frame (Data length 64 bytes).

The 64-byte data increments from 0 to 63.CAN data received by the CAN tool:

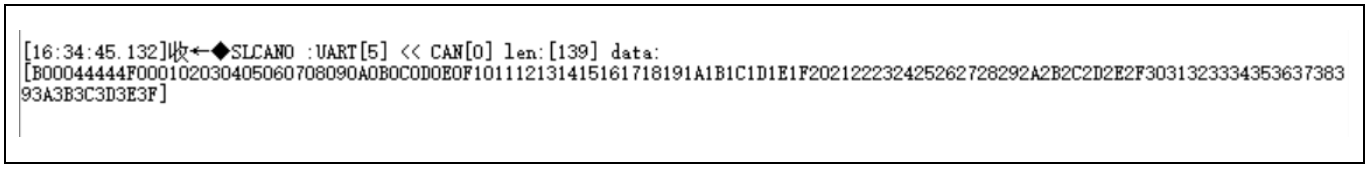


Figure 2-1 Data Received by the Receiver End of the CAN Tool

In Figure 2-1, the data received is in square brackets [xxx], and the data is divided into 4 parts:

- Frame type: B -> Variable speed extension frame
- Frame ID: 00044444 -> 0x44444
- length: F -> 64bytes
- data:

  00 · 01 · 02 · 03 · 04 · 05 · 06 · 07 · 08 · 09 · 0A · 0F · 0C · 01 · 0E · 0F ·

  10 · 11 · 12 · 13 · 14 · 15 · 16 · 17 · 16 · 19 · 1A · 1B · 1C · 1D · 1E · 1F ·

  20 · 21 · 22 · 23 · 24 · 25 · 26 · 27 · 26 · 29 · 2A · 2B · 2C · 2D · 2E · 2F ·

  30 · 31 · 32 · 33 · 34 · 35 · 36 · 37 · 38 · 39 · 3A · 3B · 3C · 3D · 3E · 3F

- 5 Mbps Variable Baud Rate Transmitter Foot Data Signal:

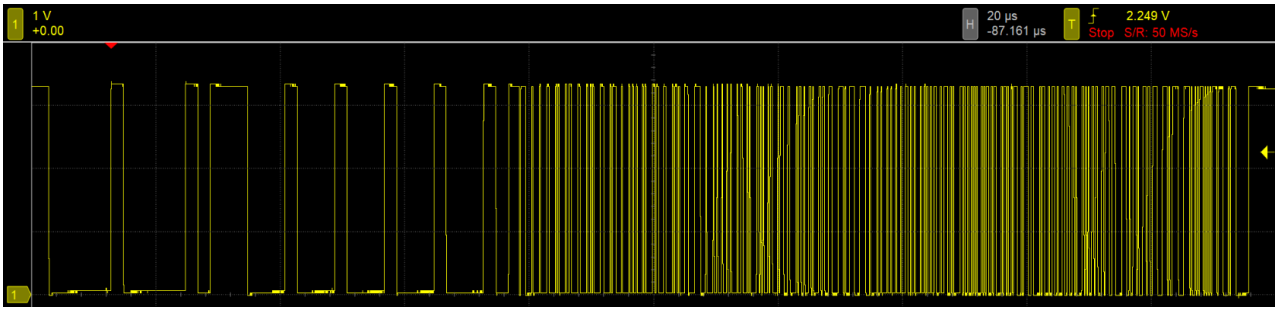


Figure 2–2 Data Waveform of the Routine Transmit Root Data

Note: If the TDC function is not enabled, the data of the 500kbps arbitrated segment will be sent, but the data in the 5Mbps data segment failed to be read back, then the data segment will stop being sent and an error will be reported (the part with a higher signal frequency in the figure above). Figure 2-3 shows the complete transmission of a frame of data.

3 Software and Hardware Requirements

1.5 Software Requirements

- BSP version
  - M460_Series_BSP_CMSIS_V3.00.002
- IDE version
  - Keil uVersion 5.30

1.6 Hardware Requirements

- Circuit components
  - NuMaker-M476HJ_V1.0CANFD analyzer (the analyzer shown in the diagram can be replaced by another brand).
- Schematic diagram of the route
  - Connect UART0 of the M476HJ evaluation board to the PC UART to display the execution result of the sample code.
  - Dock the CAN_Tool with the CANH/CANL of the M476 to ensure normal communication.

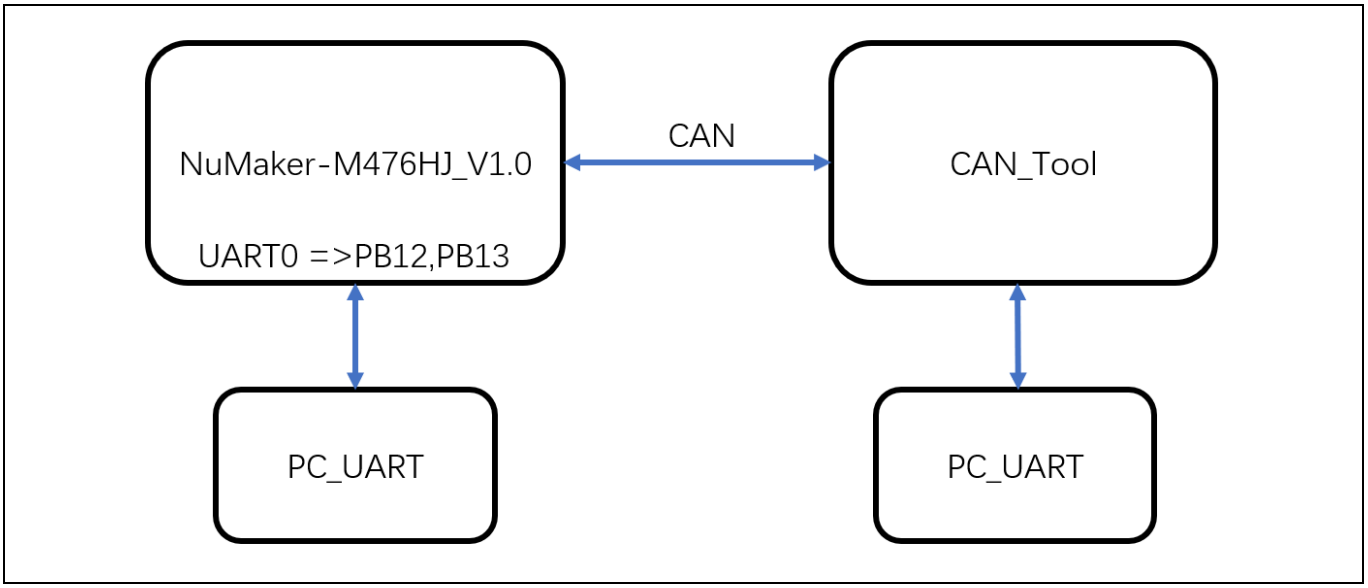


Figure 3-1 Pin Connect

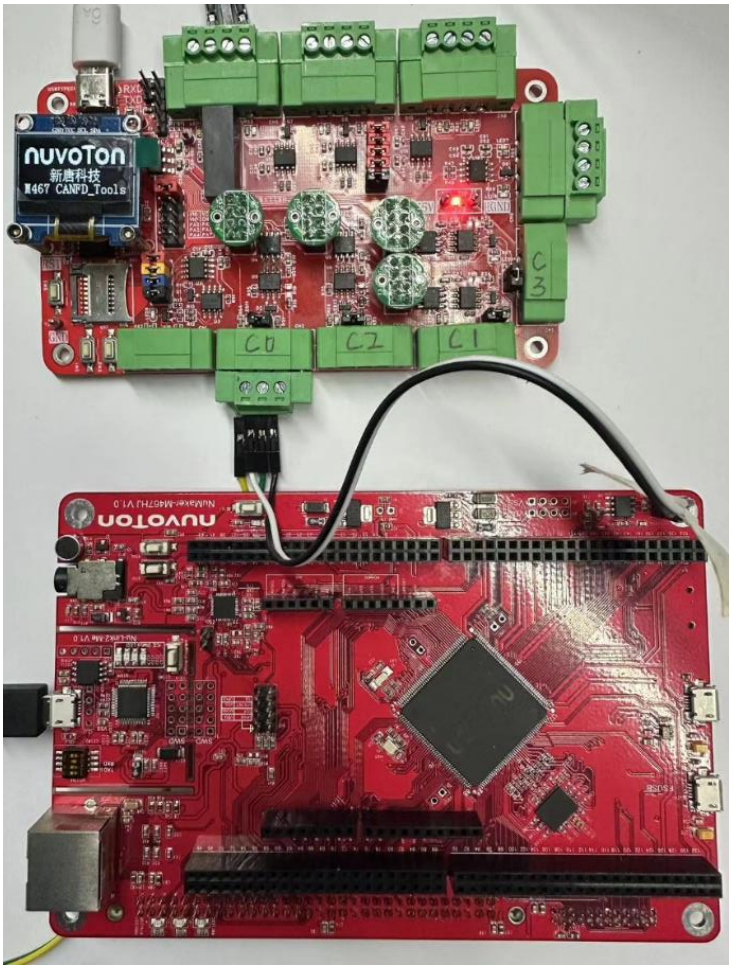


Figure 3-2 Physical Wiring

4 Directory Structure

The directory structure is shown below.

EC_M460_CANFD_TDC_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	
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 *CANFD_TxRxINT_TDC.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2024.03.28	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.