M460 CAN FD 收发器延迟补偿功能

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码介绍如何使用 M460 CAN FD 收发器延迟补偿功能
BSP 版本	M460_Series_BSP_CMSIS_V3.00.002
开发平台	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. 概述

本范例代码介绍了 CAN FD 的收发器延时补偿功能。以确保 CAN FD 在高速通讯发消息的时候，避免发送器进入 BUSOFF。

Transceiver Delay Compensation 收发器延迟补偿(简称 TDC)，主要解决收发器环路物理延迟对信号回采可靠性校验的问题。

为了方便理解 TDC 的原理本范例先从以下几点来介绍 TDC 功能，然后再介绍代码配置方式：

- 什么是收发延迟
- 为什么要用发送器延迟补偿
- TDC 的设置方法
- 实例分析

本范例代码需要配一个 CAN FD 的调试工具，用于观测收发的数据。本例中已经有通讯信号的捕捉图示，如果需要进行验证，建议配个示波器进行观测。

1.1 什么是收发延迟

当我们进行数字通讯的时候信号通过 MCU 的 TTL 电平进行传输，而单一的数据信号不适合进行长距离，高可靠性的传输，这时我们需要对信号做一些变换，让变换后的信号适合长距离的可靠通讯。

本文中的使用 CAN 差分信号进行对称传输，图 1–1 是文中 CAN 转换芯片的内部结构和传输延迟示意：

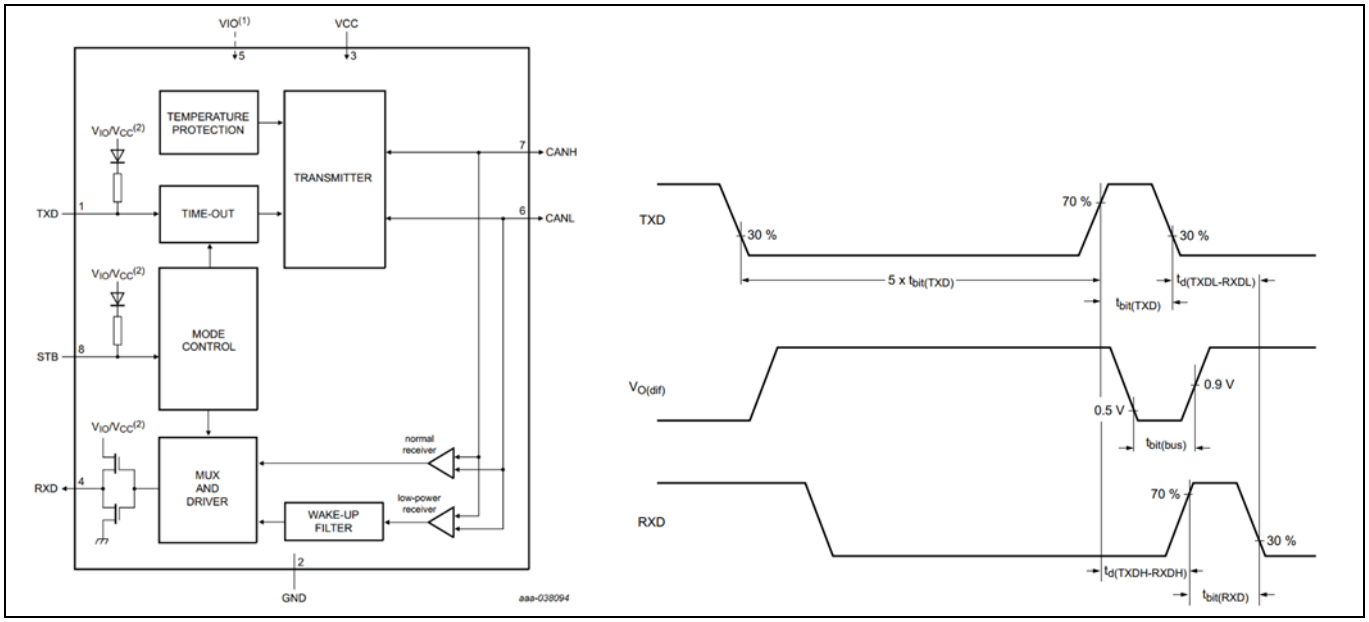


图 1–1 内部结构和传输延迟

TXD 为控制器信号发送，Vo(dif)为外部总线的物理信号，RXD 为控制器信号接收。

图 1–1 仅仅是转换器的延迟，同样在环境恶劣的通讯环境中，我们为了保护控制器还会加入信号隔离器，以免外界噪声干扰控制器的运行，但是同样也会带来更长时间的信号延迟。

图 1-2 是本例中使用的隔离器的传输延迟时间：

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

图 1-2 隔离器的传输延迟

从图 1-2 我们可以看到 10Mbps（20~60ns）的比 1Mbps（50~150ns）的隔离器延时长了近一倍。

综上在实际的应用中一共存在3种延迟：

- 收发器环路延迟（收发器接收发送数据后再回传给控制器）
- 发送延迟（发送隔离）
- 接收延迟（接收隔离）

1.2 为什么要用收发器延迟补偿

CAN 总线的设计之初就对其通讯提出了高性能和可靠性的要求，所以在 CAN 协议中规定：发送方发送数据时，需检测回传的数据与发送是否一致，若不一致则产生错误帧，就是用回采的方式判断是否信号发送成功。

CAN 检测到发送信息回采不正确并达到一定数量后，控制器会认为总线异常进入 BUSOFF 状态，并将自己挂起不影响总线其他控制器的通讯。

在经典 CAN ISO11898-5 的白皮书中规定了总环路延时为 255ns。这个规定在低速模式下能够确保环路延时不会对数据回采造成影响。但是当 CANFD 模式在 5Mbps 数据段通讯，实际位发送时间仅有 200ns，如果不考虑收发延迟，那么总线将完全不能正常收发数据了。另外在章节 1.1 的分析中几类延迟在实际应用时叠加起来就很有可能会超过 255ns。

针对上述的问题，BOSCH 设计了一个叫 Transceiver Delay Compensation 的功能（简称 TDC），这个功能可以自动去测量电路中的延时时间，并且通过延时时间重新设置数据段的信号采样点位置。这个采样点的位置定义为二次采样点 Second Sample Point (简称 SSP)。

图 1-3 展示了自动延时和 SSP 采样点的计算方法。

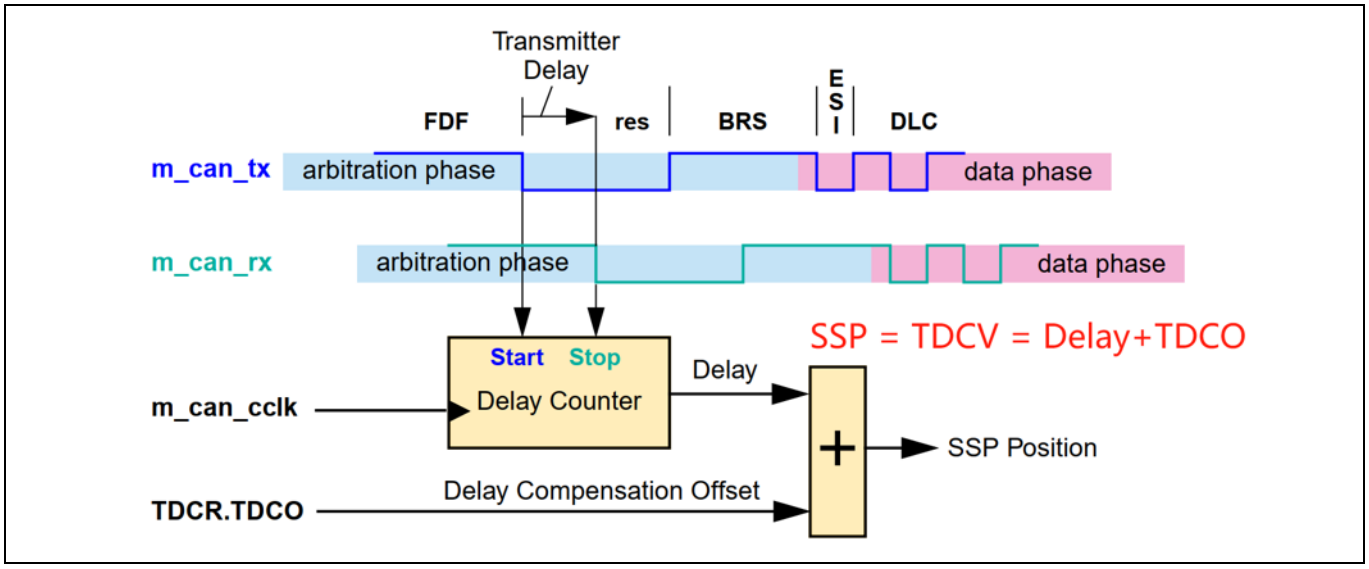


图 1-3 自动计算 SSP 的示意图

1.3 TDC 的设置方法

TDC 主要是用来设置二次采样点 SSP，此功能涉及 3 个寄存器，4 个参数值。

TDCF，TDCO 和 TDCV 均以最小时间 tq 为计数单位，计算方法同波特率寄存器。

- DBTP 寄存器中的 TDC 位：此位表示延时采样功能的使能，大于 1Mbps 通讯时建议打开。
- TDCR 寄存器中的 TDCO 和 TDCF：二次采样点 SSP = Delay+ TDCO 。

  TDCO ：二次采样延时偏移量，一般参考 DTSEG1 值进行设置。

  TDCF ：最小二次采样点。同时这个值是必须大于 TDCO 的 。

  在不知道具体延时参数的时候可以默认 TDCF =DTSEG1+1 ，确保 TDCF > TDCO。

  TDCF 可根据实际硬件的测试情况做些调整，硬件确认后物理延时也就定下来了。
- PSR 寄存器中的 TDCV：自动计算的延时采样点 TDCV = TDCO + 回采物理延时（Delay）。

  当 TDCO = 0 时实际测量到的就是回采的物理延时（Delay）。

  当 TDCF ≥ TDCV 时，二次采样点 SSP = TDCF 。

  当 TDCF < TDCV 时，二次采样点 SSP = TDCV 。

1.4 实例分析

1.4.1波形分析

实验环境：

- 10Mbps 的隔离芯片 ADUM1201BRZ（延迟 20~60ns）
- 1Mbps 的隔离芯片 ADUM1201ARZ（延迟 50~150ns）
- CAN 收发器：TJA1462A
- MCU 控制：M467HJHAN

隔离器的延时数据见图 1-2.

CAN 收发器 TJA1462A 的延时数据图 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

图 1-4 CAN 收发器延时数据

实验方法：

M467 开发板 CAN 发送一帧数据，用示波器 2 个通道分别接 M467CAN 的 RXD 和 TXD，发送数据的时候同时测量 TXD 和 RXD 的信号，记录测量 1Mbps 隔离和 1Mbps 隔离的延迟差异。

图 1-5 为 10Mbps 隔离芯片 CAN FD 收发的一帧数据：

标准帧，地址 0x000，数据长度 12byte，可变波特率。

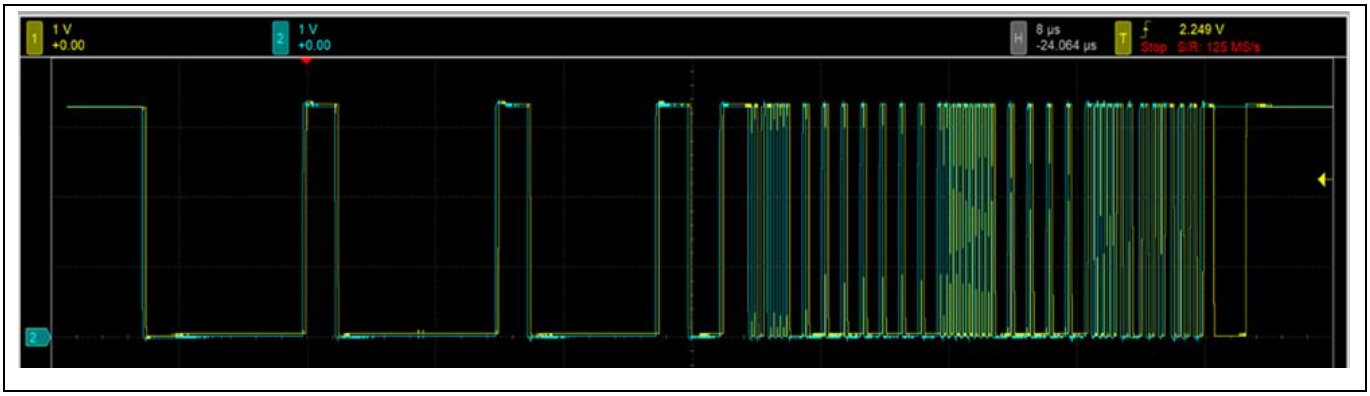


图 1-5 1Mbps 隔离的 CAN FD 一帧收发数据波形

测量 MCU 端 TXD 和 RXD 引脚，可以很明显的看到 TXD 蓝线和 RXD 黄线信号有叠影。

放大数据段 1 个位的波形图 1-6：

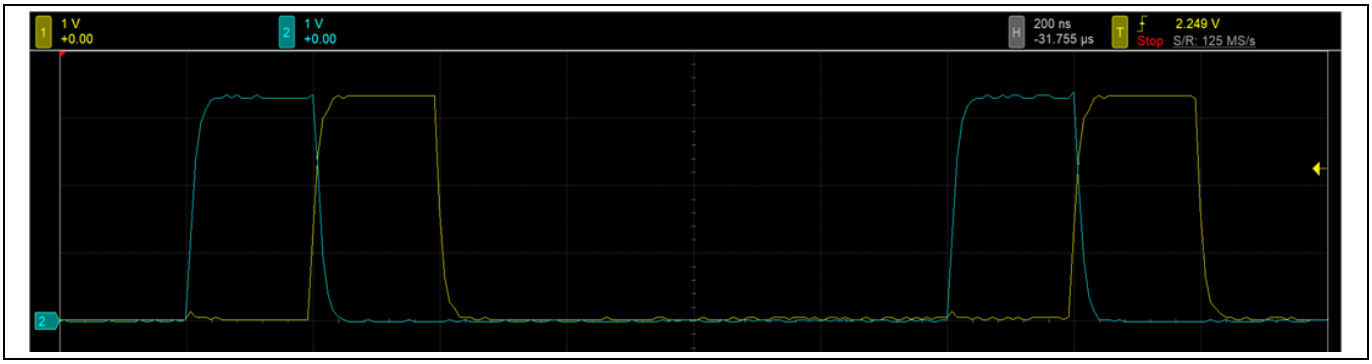


图 1-6 1Mbps 隔离数据段 1 个位的收发波形

从图 1-6 可以很明显的看到 RXD 黄线延迟于 TXD 蓝线约 200ns 才到 MCU 端，这个时候如果依然还用发射时序的采样点，那么会导致信号无法采样正确。这个时候 TDC 功能就非常重要了。

当使用 1Mbps 隔离芯片时数据段 1 个位波形的延时测试如图 1-7：

ADUM1201ARZ（延迟 50~150ns），CAN 收发器同样为 TJA1462A 。

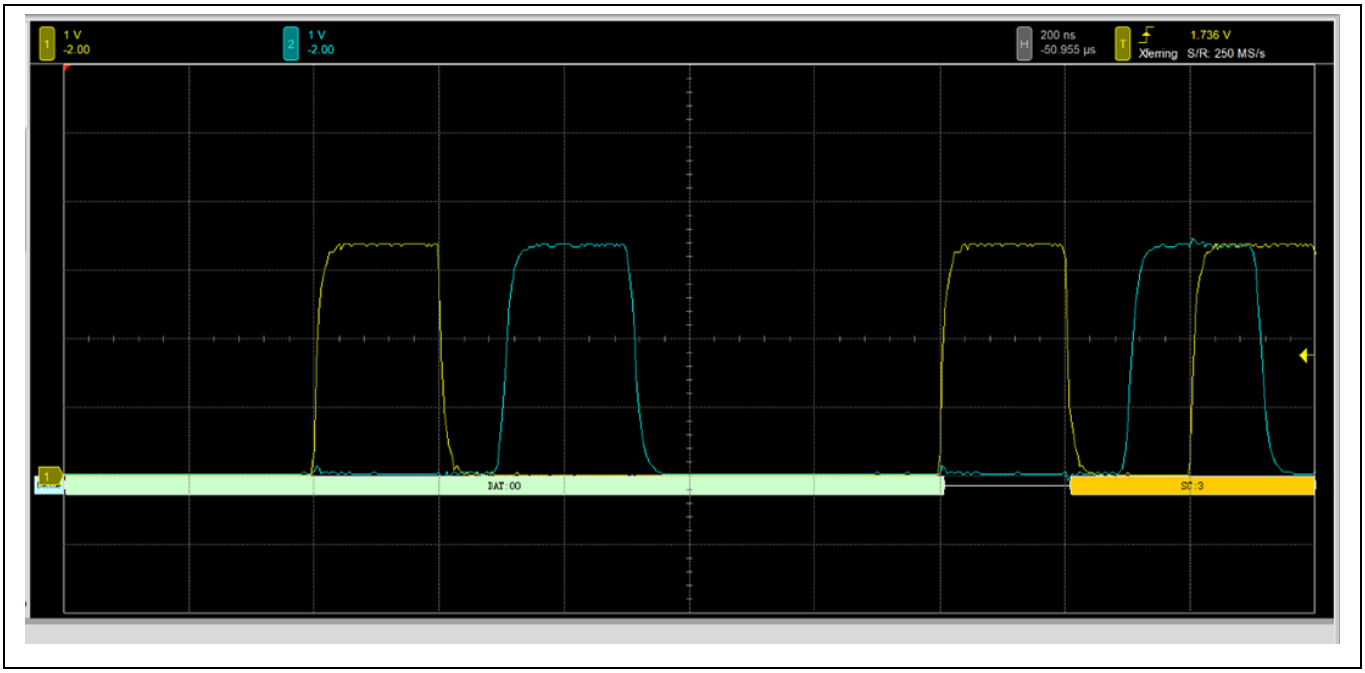


图 1-7 1Mbps 隔离数据段 1 个位的收发波形

从图 1-7 中可以看到 1Mbps 隔离 RXD 蓝线延迟于 TXD 黄线近 300ns，不同的隔离器延时性能差异较大。

当硬件设计完成之后，这个物理延时时间也就基本定下来了，可以同过示波器来确认准确的延时时间，即确认 TDCF 的值。

1.4.2 TDC 的设置实例

实例通过 Nu-Link 截图观测 Keil 内实际 TDC 寄存器的设置，并且校验演算过程是否正确。

仲裁段波特率为 500kbps，数据段为 5Mbps，数据段波特率寄存器：

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

一个位的时间为一个总 Tq，总 Tq 拆分成 10 个元素，用这个 10 个元素来进行采样点的设置。

实际的寄存器设置如图 1-8：

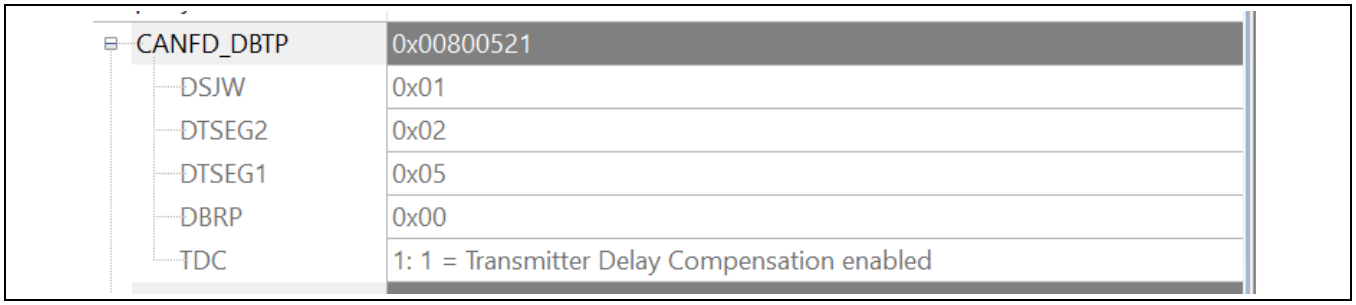


图 1-8 CAN FD 数据段波特率设置寄存器

按照上文 1.3 TDC 的设置方法进行寄存器设置：

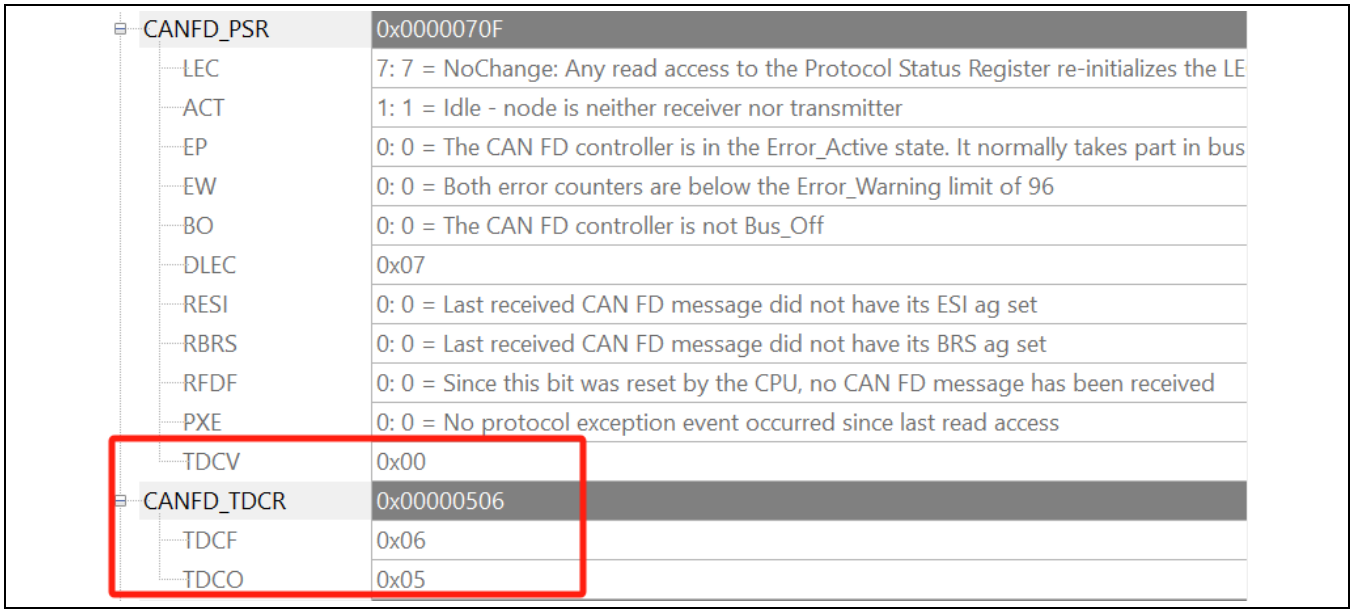
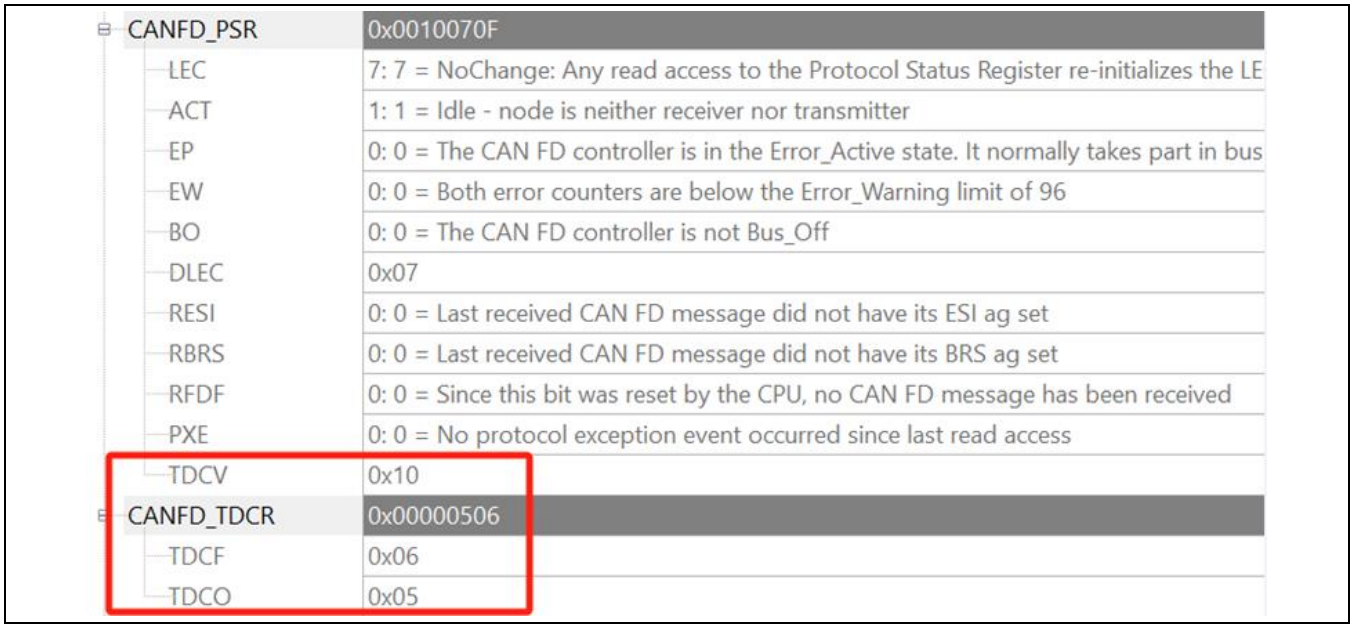


图 1-9 CANFD 初始化后 TDC 相关寄存器状态

CAN 初始化此时还未发送数据：

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

CAN 发送一次数据后，总线自动测量到延时，并将测量值存入 TDCV 寄存器：



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

图 1-10 CAN FD 发送数据后 TDC 相关寄存器状态

发送数据后：TDCV = 16； TDCO = DTSEG1 ；TDCF = DTSEG1 + 1。

数据段 5Mbps 根据波形和参数计算（验证 1.3 计算方法）：

数据段总 Tq = 10tq = 200ns（5Mbps）；单个元素 1tq = 20ns 。

TDCO= DTSEG1 = 5tq 。

TDCF = DTSEG1 + 1 = 6tq 。

采用 1Mbps 隔离时二次采样点的位置在（320ns 处）：

TDCV = 16tq

= 11tq（delay） + 5tq（TDCO）

= 11*20ns + 5*20ns

= 320ns

此时 TDCV(16tq) > TDCF(6tq) 二次采样点按照 TDCV 来计算如下图：

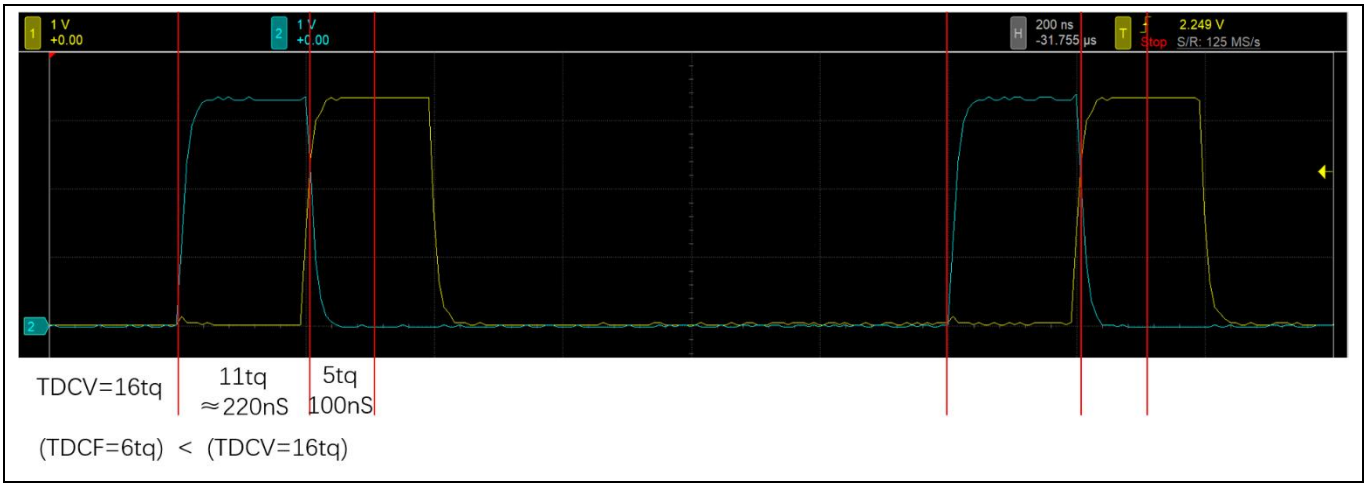


图 1-11 CAN FD 的二次采样点

由图 1-11 看二次采样点落在黄线端红线处，可见 TDCV= 16tq 能够正确采样到回传的位电平。硬件定型后通过上图的测量可以选择一个合适值设置 TDCF。

注：

- TDCV 的测量值和示波器的观测略有误差
- 示波器 1 格为 200ns，实际采样点在 280ns 到 360ns 都能正确采样（14≤TDCF≤18）
- 蓝线为 TXD，黄线为 RXD

2. 代码介绍

首先确认例程波特率是否设置成 5Mbps，修改波特率位于 *main.c* 文件中的 *CANFD_Init()*函数内。

```
/*---------------------------------------------------------------------------*/
/*                          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);
……
```

CAN FD 的 TDO 功能开启的函数，代码位于 *canfd.c*，当 DataBitRate 大于 1Mbps 时自动开启 TDC 功能。

```
/* @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;

      }
}
```

此函数需要放置于 *CANFD_Open(CANFD_T *psCanfd, CANFD_FD_T *psCanfdStr)*函数中，在初始化 CANFD 时钟(CANFD_CalculateTimingValues)之后进行波特率的判断。

```
/**
 * @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();
……
```

通过串口的提示进行 CAN 数据的发送，按数字 8 选择需要发送的帧，可以看到数据能够正常发送出去：

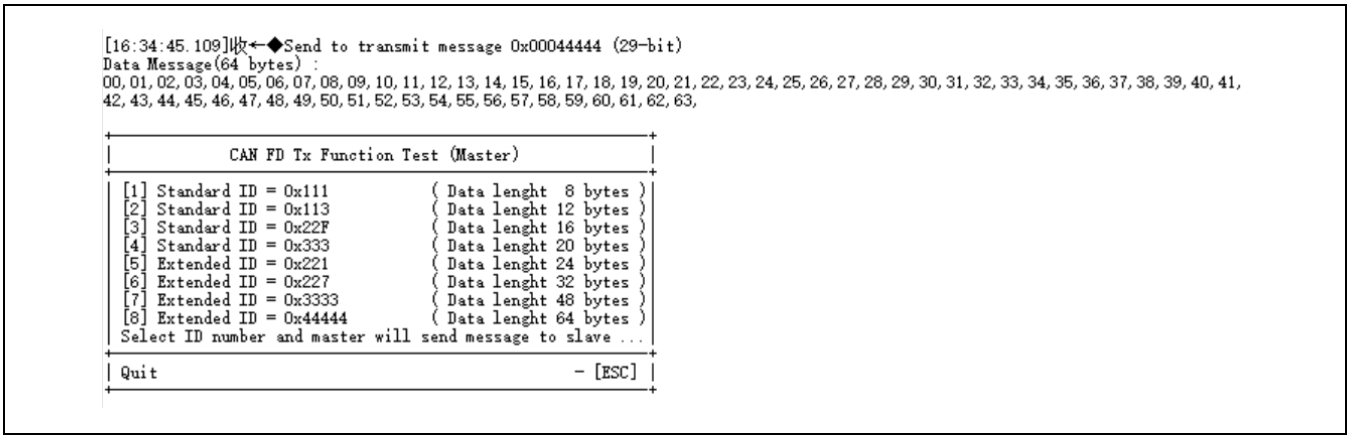


图 2-1 例程 CAN 发送一帧数据

图 2-1 CAN FD 序号 8 的数据帧为：

Extended ID=0x44444 变速帧（Data length 64 bytes）

64 个字节的数据从 0 开始递增到 63。

CAN 工具接收到的 CAN 数据：

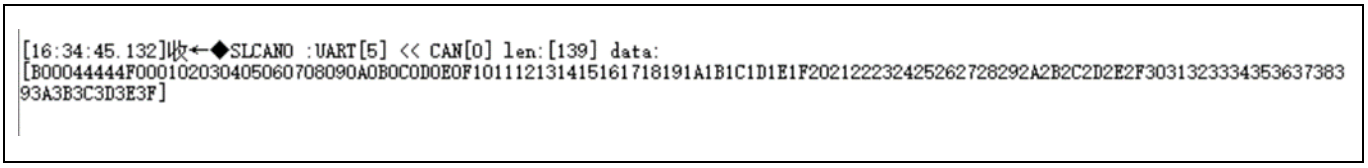


图 2-2 CAN 工具接收端收到的数据

图 2-2 中方括号[xxx]内为收到的数据，数据分为 4 个部分：

- 帧类型：B -> 变速扩展帧
- 帧 ID ：00044444 -> 0x44444
- 长度 ：F -> 64bytes
- 数据 ：

  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

- 5Mbps 可变波特率发送脚的数据信号：

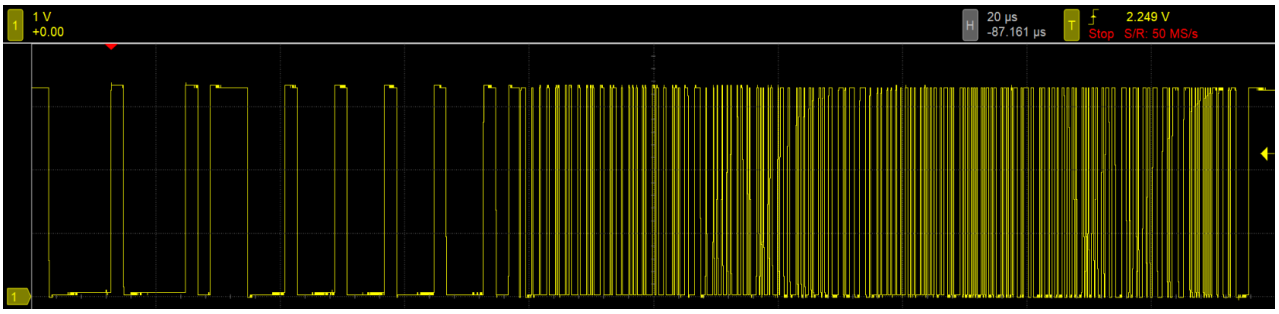


图 2-3 例程发射脚数据的数据波形

注：如果 TDC 功能没有打开，500kbps 仲裁段的数据会发送出去，但是由于 5Mbps 的数据段回采不正确，到数据段就会停止发送并报错（上图中信号频率较高的部分）。图 2-3 为完整的发送了一帧数据。

3. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-M476HJ_V1.0
  - CANFD 分析仪（图示中的分析仪可以用其他品牌代替）
- 线路示意图
  - 将 M476HJ 开发板 UART0 连接到 PC UART，以显示范例代码的执行结果。
  - 将 CAN_Tool 和 M476 的 CANH/CANL 对接以确保能够正常通讯

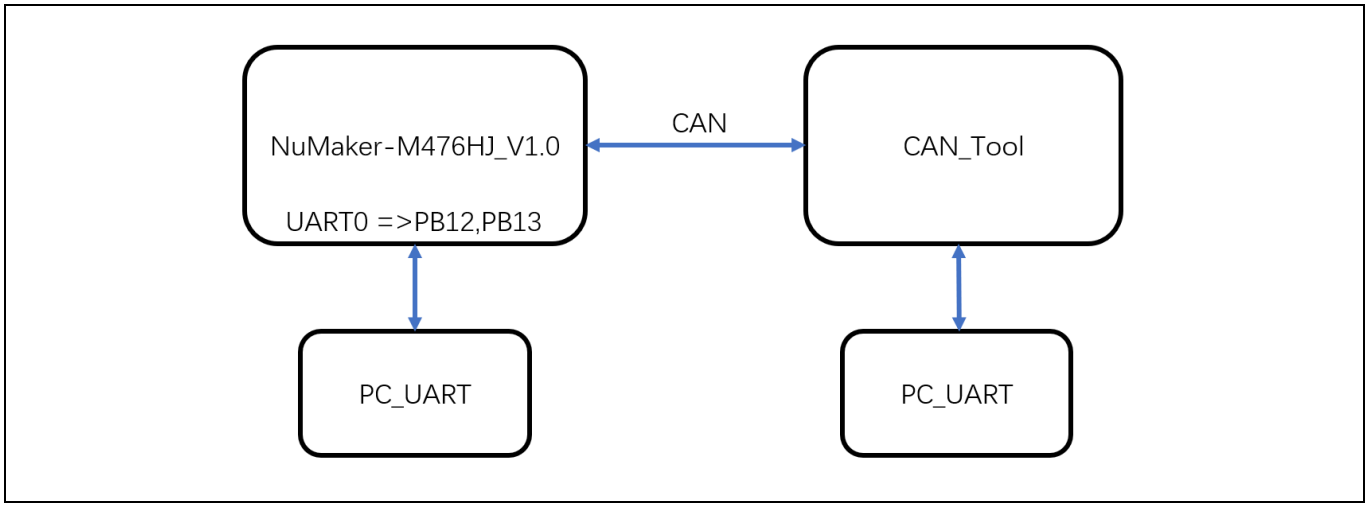


图 3-1 线路示意图

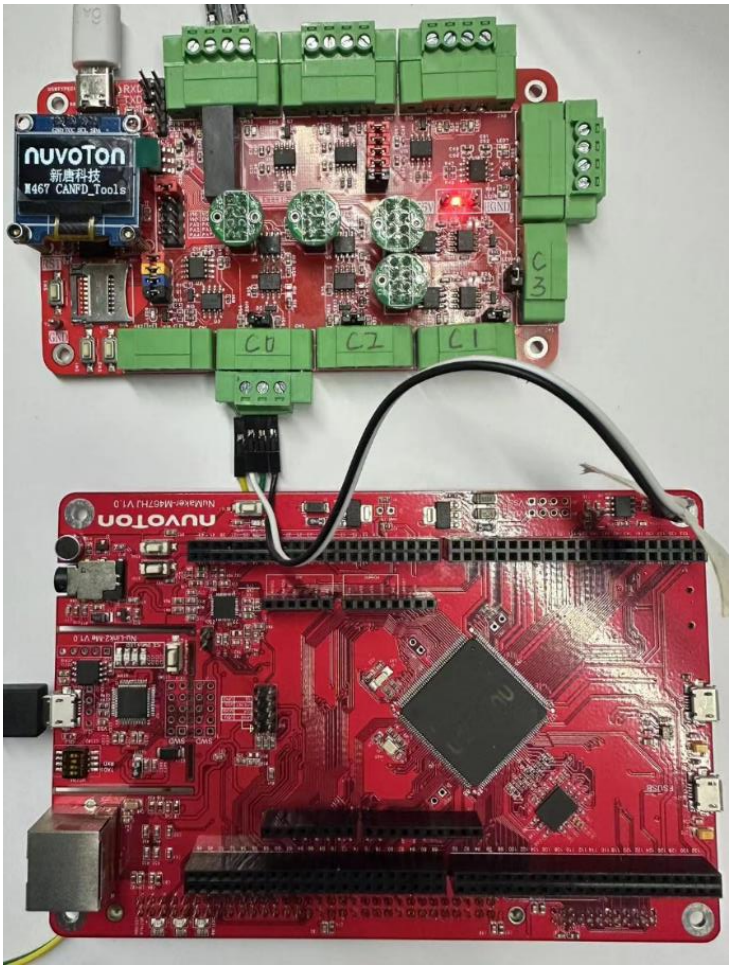


圖 3-1 实物接线

4. 目录信息

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

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *CANFD_TxRxINT_TDC.uvprojx*
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2024.03.28	1.00	初始发布。

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.