Using M2A23 ACMP as CAN FD Transceiver

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M2A23 ACMP as a CAN FD transceiver.
BSP Version	M2A23_Series_BSP_CMSIS_V3.00.001
Hardware	NuMaker-M2A23SG5AC_V1.0 CAN_ ACMP_TRANSCEIVER_EVB_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 document describes an alternative to traditional CAN FD transceivers using the chip's built-in comparator in conjunction with external BJTs. By separating the CAN TX path and utilizing the comparator as the CAN RX input mechanism, it meets the requirements of lower-speed applications while reducing the cost of building a communication platform.

1.1 Principle

Traditional CAN FD systems commonly use high-cost transceiver components for signal transmission and reception. This solution adopts the chip's internal comparator combined with external BJT circuitry to handle the CAN TX and CAN RX paths. The key design concepts are as follows:

- The CAN TX signal is split into TXH and TXL signals to control the external BJTs for high and low voltage output respectively.
- Depending on pin configuration, the CAN RX pin is switched to comparator output mode, using CAN_H and CAN_L as inputs to interpret the signal.

1.2 Signal Path Design

Figure 1-1 illustrates the wiring overview of this solution, showing the connections between TXH/TXL and external BJTs, and the signal paths between the internal comparator and CAN_H/CAN_L.

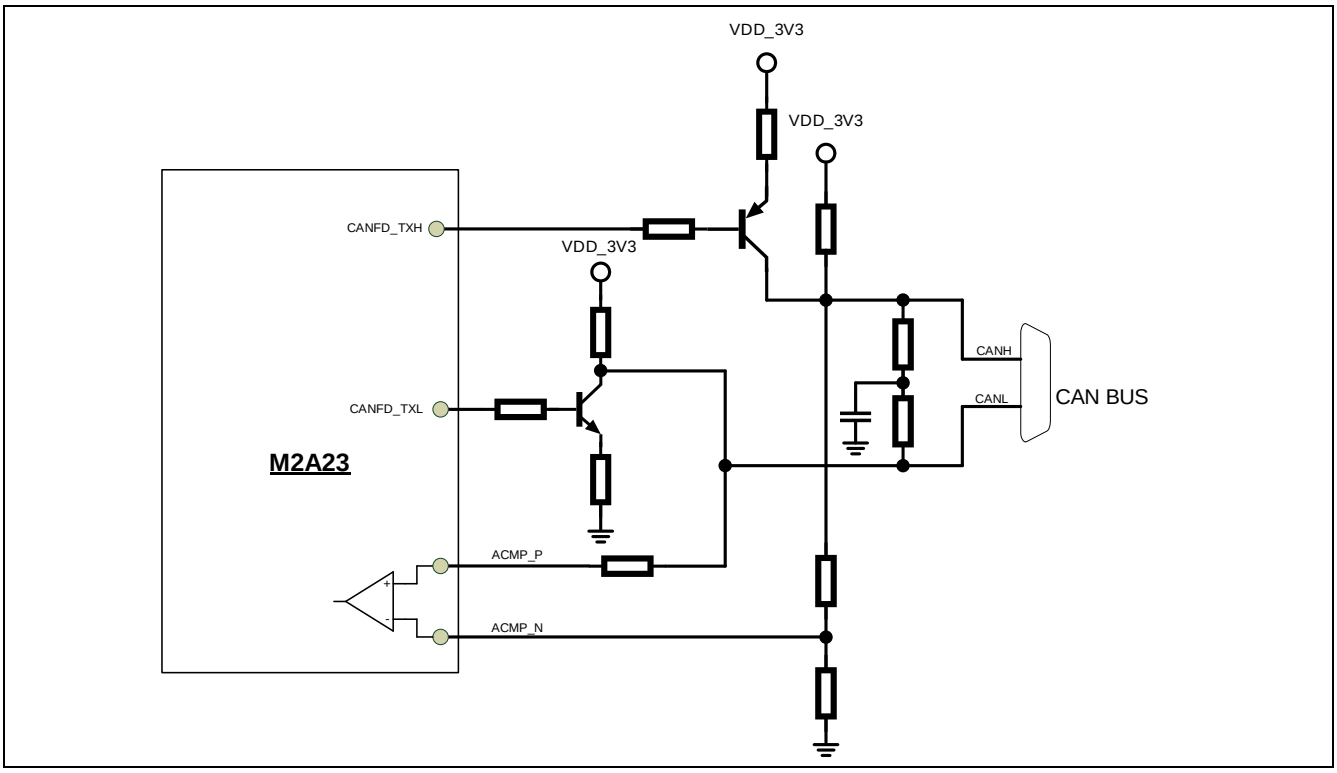


Figure 1-1 Circuit and Wiring Overview

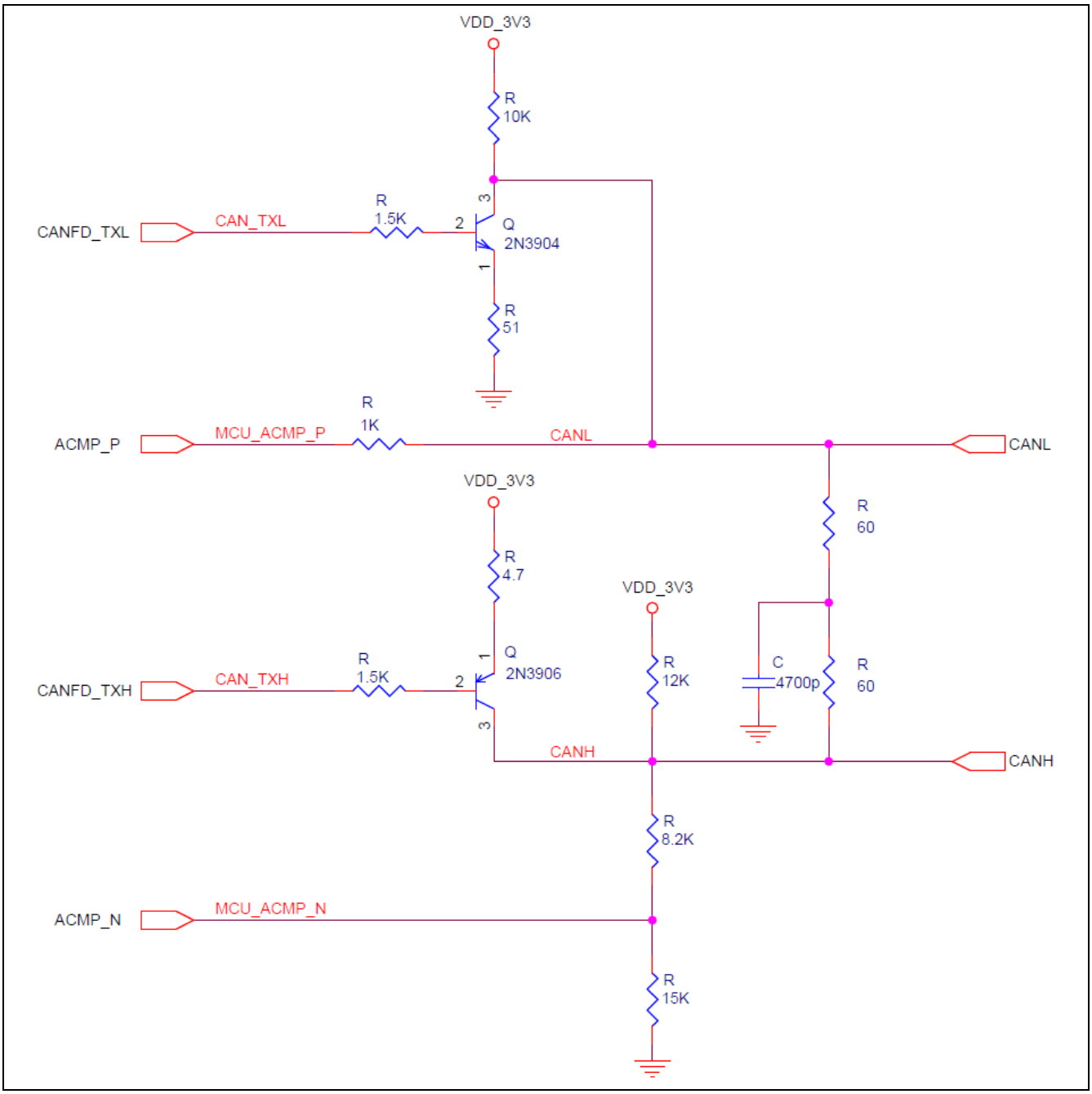


Figure 1-2 Suggested Circuit

1.2.1 CAN TX Path

Signal Separation:

- The CAN TX signal is split into TXH and TXL to control the conduction state of the BJTs, achieving high and low voltage output.

Signal Control:

- The use of external BJTs ensures correct signal transmission.

1.2.2 CAN RX Path

Comparator Output Configuration:

- The CAN RX pin can be configured to operate in comparator output mode as needed, enabling real-time voltage level detection of input signals.

Signal Comparison:

- The comparator uses CAN_H and CAN_L as input signals and generates a corresponding logic output based on their voltage difference for processing by the control unit.

1.3 Circuit Test Results

This section describes the maximum achievable data bit rates under various wire lengths when both the master and slave devices use this solution. It also presents oscilloscope waveforms of CAN_H and CAN_L at a specified bit rate.

1.3.1 Wire Length and Maximum Data Rate

Table 1-1 summarizes the highest achievable data bit rates for jumper wires, 1m, 5m and 10m wiring lengths:

Core clock (MHz)	Data bit rate (Mbps)	Wire length (m)
72	4.5	Jump wire
72	4.5	1
72	4.5	5
72	4	10

Table 1-1 Test Results of Wire Length vs. Maximum Data Rate

1.3.2 Oscilloscope Waveform Observation

At a data bit rate of 500 kHz, the differential signals CAN_H and CAN_L can be observed switching on an oscilloscope, as shown in Figure 1-3. In these waveforms, Channel 2 represents CAN_H and Channel 3 represents CAN_L.

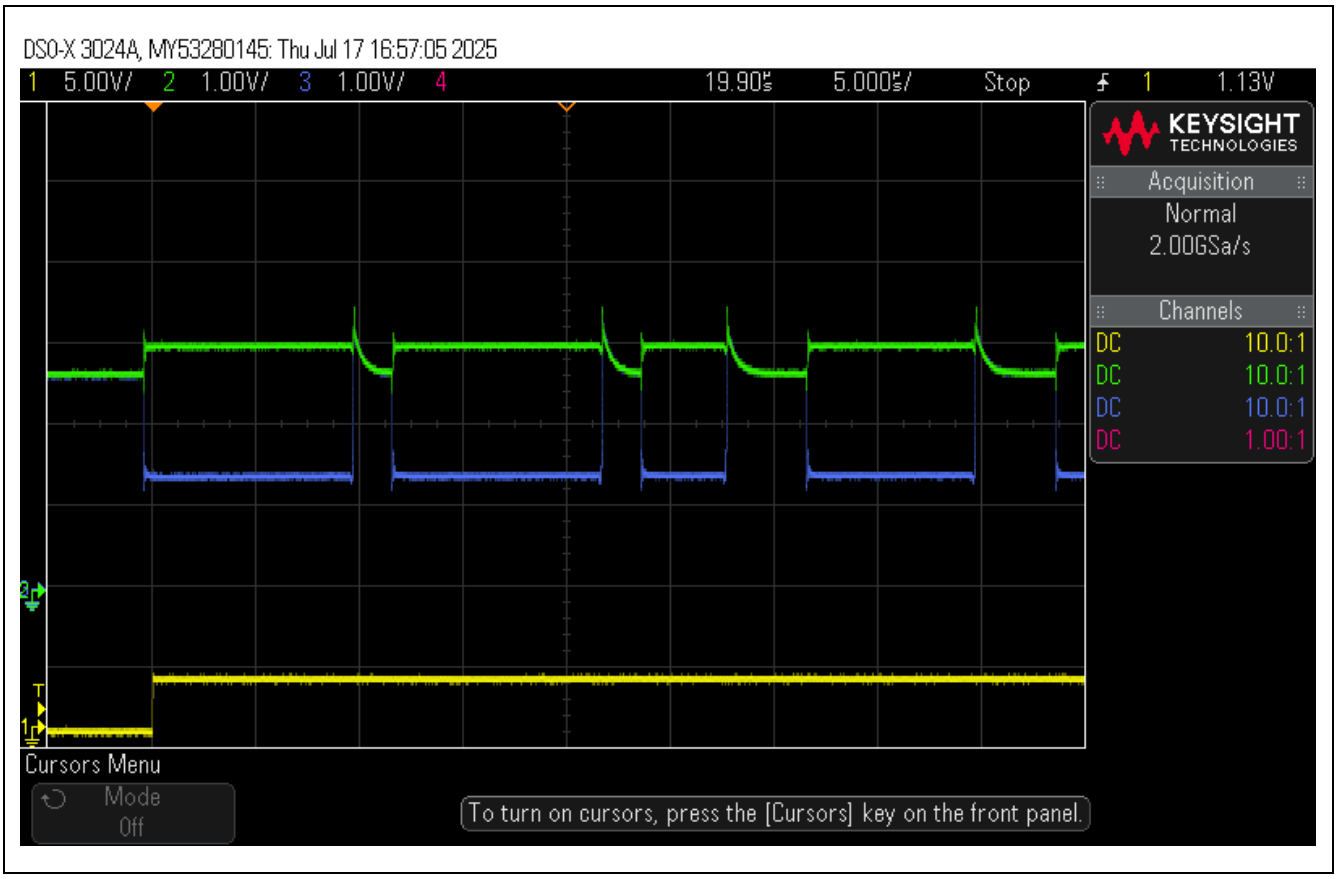


Figure 1-3 Oscilloscope Waveform

2. Code Description

The functions for configuring multi-function pins, CAN FD TX/RX signal format (CANFD_STSEL), and ACMP initialization are located in *main.c*.

In the *SYS_Init()* function, the multi-function pins for CAN FD TXH, TXL, and ACMP are configured, and the ACMP01 peripheral clock is enabled.

```
void SYS_Init(void)
{
    /*---------------------------------------------------------------------------------------------------------*/
    /* Init System Clock                                                                                       */
    /*---------------------------------------------------------------------------------------------------------*/
    /* Enable HIRC clock */
    CLK_EnableXtalRC(CLK_PWRCTL_HIRCEN_Msk);

    /* Wait for HIRC clock ready */
    CLK_WaitClockReady(CLK_STATUS_HIRCSTB_Msk);

    /* Set core clock to 72MHz */
    CLK_SetCoreClock(72000000);

    /* Enable all GPIO clock */
    CLK->AHBCLK |= CLK_AHBCLK_GPIOACKEN_Msk | CLK_AHBCLK_GPIOBCKEN_Msk | CLK_AHBCLK_GPIOCCKEN_Msk |
                   CLK_AHBCLK_GPIODCKEN_Msk | CLK_AHBCLK_GPIOFCKEN_Msk;

    SYS->GPF_MFPL = (SYS->GPF_MFPL & (~SYS_GPF_MFPL_PF2MFP_Msk)) | SYS_GPF_MFPL_PF2MFP_XT1_OUT;
    SYS->GPF_MFPL = (SYS->GPF_MFPL & (~SYS_GPF_MFPL_PF3MFP_Msk)) | SYS_GPF_MFPL_PF3MFP_XT1_IN;

    /* Enable UART0 module clock */
    CLK_EnableModuleClock(UART0_MODULE);

    /* Select UART0 module clock source as HIRC and UART0 module clock divider as 1 */
    CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL2_UART0SEL_HIRC, CLK_CLKDIV0_UART0(1));

    /* Enable ACMP01 peripheral clock */
    CLK_EnableModuleClock(ACMP01_MODULE);

    /* Select CAN FD0 clock source is HCLK */
    CLK_SetModuleClock(CANFD0_MODULE, CLK_CLKSEL0_CANFD0SEL_HCLK, CLK_CLKDIV1_CANFD0(1));

    /* Enable CAN FD0 peripheral clock */
    CLK_EnableModuleClock(CANFD0_MODULE);

    /*---------------------------------------------------------------------------------------------------------*/
    /* Init I/O Multi-function                                                                                 */
    /*---------------------------------------------------------------------------------------------------------*/

    /* Set PA multi-function pins for UART0 RXD and TXD */
    SET_UART0_RXD_PA6();
    SET_UART0_TXD_PA7();

    /* Set PB multi-function pins for CAN FD0 TXH and TXL */
    SET_CANFD0_TXH_PB13();
    SET_CANFD0_TXL_PB14();
```

```
    /* Set PB2 multi-function pin for ACMP0 positive input pin and PB2 multi-function pin
for ACMP0 negative input pin */
    SET_ACMP0_P1_PB2();
    SET_ACMP0_N_PB3();

    /* Disable digital input path of analog pin ACMP to prevent leakage */
    GPIO_DISABLE_DIGITAL_PATH(PB, BIT2);
    /* Disable pull-up resistor */

    PB->PUSEL = 0;
}
```

After system initialization, *ACMP_Init()* is called to enable ACMP0 and configure ACMP_CTL_POSSEL to use ACMP0_P1.

```
/*---------------------------------------------------------------------------------------
-----------------*/
/* Init ACMP
*/
/*---------------------------------------------------------------------------------------
-----------------*/
void ACMP_Init(void)
{
    /* Configure ACMP0. Enable ACMP0 and select ACMP0_N as the source of ACMP negative
input. */
    ACMP_Open(ACMP01, 0, ACMP_CTL_NEGSEL_PIN, ACMP_CTL_HYSTERESIS_DISABLE);
    /* Select P1 as ACMP positive input channel */
    ACMP_SELECT_P(ACMP01, 0, ACMP_CTL_POSSEL_P1);
}
```

In *CANFD_Init()*, aside from setting the baud rate and ID, the CANFD_STSEL register also needs to be configured.

The RXSS is set to select the ACMP0 output. The TXLS is configured so that logic 0 outputs high voltage, and logic 1 outputs low voltage. The TXHS is configured so that logic 0 outputs low voltage, and logic 1 outputs high voltage.

```
/*---------------------------------------------------------------------------------------
-----------------*/
/* Init CAN FD
*/
/*---------------------------------------------------------------------------------------
-----------------*/
void CANFD_Init(void)
{
    CANFD_FD_T sCANFD_Config;

    printf("+-------------------------------------------------------------+\n");
    printf("|                       Pin Configure                         |\n");
    printf("+-------------------------------------------------------------+\n");
    printf("|         CAN_TXD                             CAN_TXD         |\n");
    printf("|         CAN_RXD                             CAN_RXD         |\n");
    printf("|            +-----------+  CANBUS  +-----------+            |\n");
    printf("|     ------>|           |          |           |<------     |\n");
    printf("|     CAN_TXL|           |          |           |CAN_TXL     |\n");
    printf("|     ------>|           |<-------->|           |<------     |\n");
    printf("|     CAN_TXH|   CANFD   |  CAN_H   |   CANFD   |CAN_TXH     |\n");
    printf("|            | ACMP+BJTs |          | ACMP+BJTs |            |\n");
    printf("|     <------|           |<-------->|           |------>     |\n");
    printf("|      ACMP_P|           |  CAN_L   |           |ACMP_P      |\n");
    printf("|     <------|           |          |           |------>     |\n");
    printf("|      ACMP_N|           |          |           |ACMP_N      |\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;
    sCANFD_Config.sBtConfig.sDataBitRate.u32BitRate = 500000;

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

    NVIC_EnableIRQ(CANFD00_IRQn);

    /* Receive 0x111 (11-bit id) in CAN FD rx message buffer 0 by setting mask 0 */
    CANFD_SetSIDFltr(g_pCanfd, 0, CANFD_RX_BUFFER_STD(0x111, 0));
    /* Receive 0x22F (11-bit id) in CAN FD rx message buffer 0 by setting mask 1 */
    CANFD_SetSIDFltr(g_pCanfd, 1, CANFD_RX_BUFFER_STD(0x22F, 0));
    /* Receive 0x333 (11-bit id) in CAN FD rx message buffer 0 by setting mask 2 */
    CANFD_SetSIDFltr(g_pCanfd, 2, CANFD_RX_BUFFER_STD(0x333, 0));

    /* Receive 0x222 (29-bit id) in CAN FD rx message buffer 1 by setting mask 3 */
    CANFD_SetXIDFltr(g_pCanfd, 0, CANFD_RX_BUFFER_EXT_LOW(0x222, 1),
CANFD_RX_BUFFER_EXT_HIGH(0x222, 1));
    /* Receive 0x3333 (29-bit id) in CAN FD rx message buffer 1 by setting mask 3 */
    CANFD_SetXIDFltr(g_pCanfd, 1, CANFD_RX_BUFFER_EXT_LOW(0x3333, 1),
CANFD_RX_BUFFER_EXT_HIGH(0x3333, 1));
    /* Receive 0x44444 (29-bit id) in CAN FD rx message buffer 1 by setting mask 3 */
    CANFD_SetXIDFltr(g_pCanfd, 2, CANFD_RX_BUFFER_EXT_LOW(0x44444, 1),
CANFD_RX_BUFFER_EXT_HIGH(0x44444, 1));
    /* CAN FD Run to Normal mode */
    CANFD_RunToNormal(g_pCanfd, TRUE);

    /* Set RXSS from ACMP0 output,
       TXLS as output high while logical 0 and output low while logical 1,
       TXHS as output low while logical 0 and output high while logical 1 */
    g_pCanfd->STSEL =
        ((CANFD_RXSS_ACMP0 << CANFD_STSEL_RXSS_Pos) |
         (CANFD_TX_L0_HIGH_L1_LOW << CANFD_STSEL_TXLS_Pos) |
         (CANFD_TX_L0_LOW_L1_HIGH << CANFD_STSEL_TXHS_Pos));
}
```

After executing the program and connecting to UART, the following selection menu will be displayed in the terminal:

```
+------------------------------------------------------------------------+
|                          CAN FD Function Test                          |
+------------------------------------------------------------------------+
|  Description :                                                         |
|    The sample code needs two boards. One is master(CAN FD transmitter) |
|    and the other is slave(CAN FD receiver). Master will send 6 messages |
|    with different sizes of data and ID to the slave. Slave will check if |
|    received data is correct after getting 6 messages data.             |
|  Please select Master or Slave test                                    |
|  [0] Master(CAN FD transmitter)    [1] Slave(CAN FD receiver)          |
+------------------------------------------------------------------------+
```

Set the slave device to option 1, and the master device to option 0. The master will then send six different messages, and the slave terminal will display the received messages and their comparison results.

Message transmission on the master device:

```
Start to CAN FD Bus Transmitter :
Send to transmit message 0x00000111 (11-bit)
Data Message : 00,01,02,03,04,05,06,07,08,09,10,11,12,13,14,15,

Start to CAN FD Bus Transmitter :
Send to transmit message 0x0000022f (11-bit)
Data Message :
01,02,03,04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
31,32,

Start to CAN FD Bus Transmitter :
Send to transmit message 0x00000333 (11-bit)
Data Message :
02,03,04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,
62,63,64,65,

Start to CAN FD Bus Transmitter :
Send to transmit message 0x00000222 (29-bit)
Data Message : 03,04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,

Start to CAN FD Bus Transmitter :
Send to transmit message 0x00003333 (29-bit)
Data Message :
04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,

Start to CAN FD Bus Transmitter :
Send to transmit message 0x00044444 (29-bit)
Data Message :
05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,
35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,
65,66,67,68,


 Transmit Done
```

Message reception on the slave device:

```
Start to CAN FD Bus Receiver :
Rx buf 0: Received message 0x00000111 (11-bit)
Message Data : 00 ,01 ,02 ,03 ,04 ,05 ,06 ,07 ,08 ,09 ,10 ,11 ,12 ,13 ,14 ,15 ,

Rx buf 0: Received message 0x0000022F (11-bit)
Message Data :
01 ,02 ,03 ,04 ,05 ,06 ,07 ,08 ,09 ,10 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18 ,19 ,20 ,21 ,22 ,23
,24 ,25 ,26 ,27 ,28 ,29 ,30 ,31 ,32 ,

Rx buf 0: Received message 0x00000333 (11-bit)
Message Data :
02 ,03 ,04 ,05 ,06 ,07 ,08 ,09 ,10 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18 ,19 ,20 ,21 ,22 ,23 ,24
,25 ,26 ,27 ,28 ,29 ,30 ,31 ,32 ,33 ,34 ,35 ,36 ,37 ,38 ,39 ,40 ,41 ,42 ,43 ,44 ,45 ,46 ,4
7 ,48 ,49 ,50 ,51 ,52 ,53 ,54 ,55 ,56 ,57 ,58 ,59 ,60 ,61 ,62 ,63 ,64 ,65 ,

Rx buf 1: Received message 0x00000222 (29-bit)
Message Data : 03 ,04 ,05 ,06 ,07 ,08 ,09 ,10 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18 ,
```

```
Rx buf 1: Received message 0x00003333 (29-bit)
Message Data :
04 ,05 ,06 ,07 ,08 ,09 ,10 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18 ,19 ,20 ,21 ,22 ,23 ,24 ,25 ,26
,27 ,28 ,29 ,30 ,31 ,32 ,33 ,34 ,35 ,

Rx buf 1: Received message 0x00044444 (29-bit)
Message Data :
05 ,06 ,07 ,08 ,09 ,10 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18 ,19 ,20 ,21 ,22 ,23 ,24 ,25 ,26 ,27
,28 ,29 ,30 ,31 ,32 ,33 ,34 ,35 ,36 ,37 ,38 ,39 ,40 ,41 ,42 ,43 ,44 ,45 ,46 ,47 ,48 ,49 ,5
0 ,51 ,52 ,53 ,54 ,55 ,56 ,57 ,58 ,59 ,60 ,61 ,62 ,63 ,64 ,65 ,66 ,67 ,68 ,


 Receive OK & Check OK
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M2A23_Series_BSP_CMSIS_V3.00.001
- IDE version
  - Keil uVision 5.41

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M2A23SG5AC_V1.0
  - CAN_ ACMP_TRANSCEIVER_EVB_V1.0
- Pin Connect
  - Connect the M2A23SG5AC CAN FD TXH, TXL, ACMP_N, and ACMP_P pins to the CAN_ACMP_TRANSCEIVER evaluation board.
  - Connect the CAN_H and CAN_L lines of the master and slave CAN_ACMP_TRANSCEIVER evaluation boards to ensure proper communication.

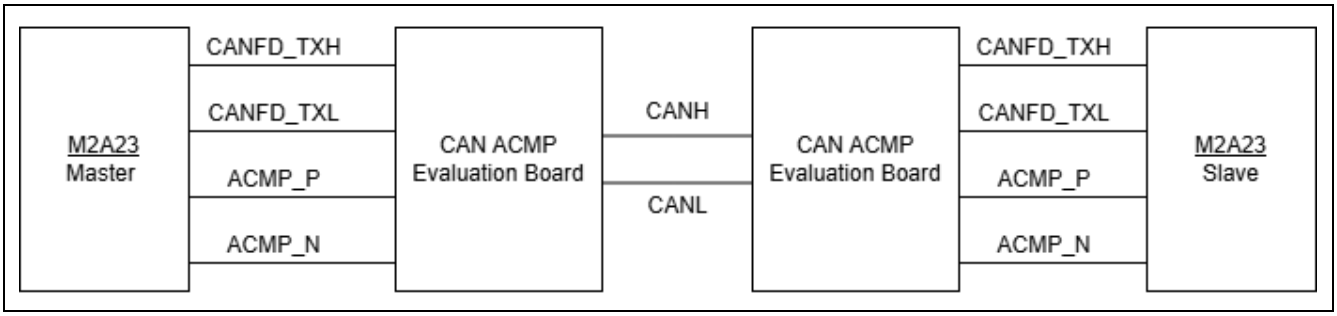


Figure 3-1 Pin Connect

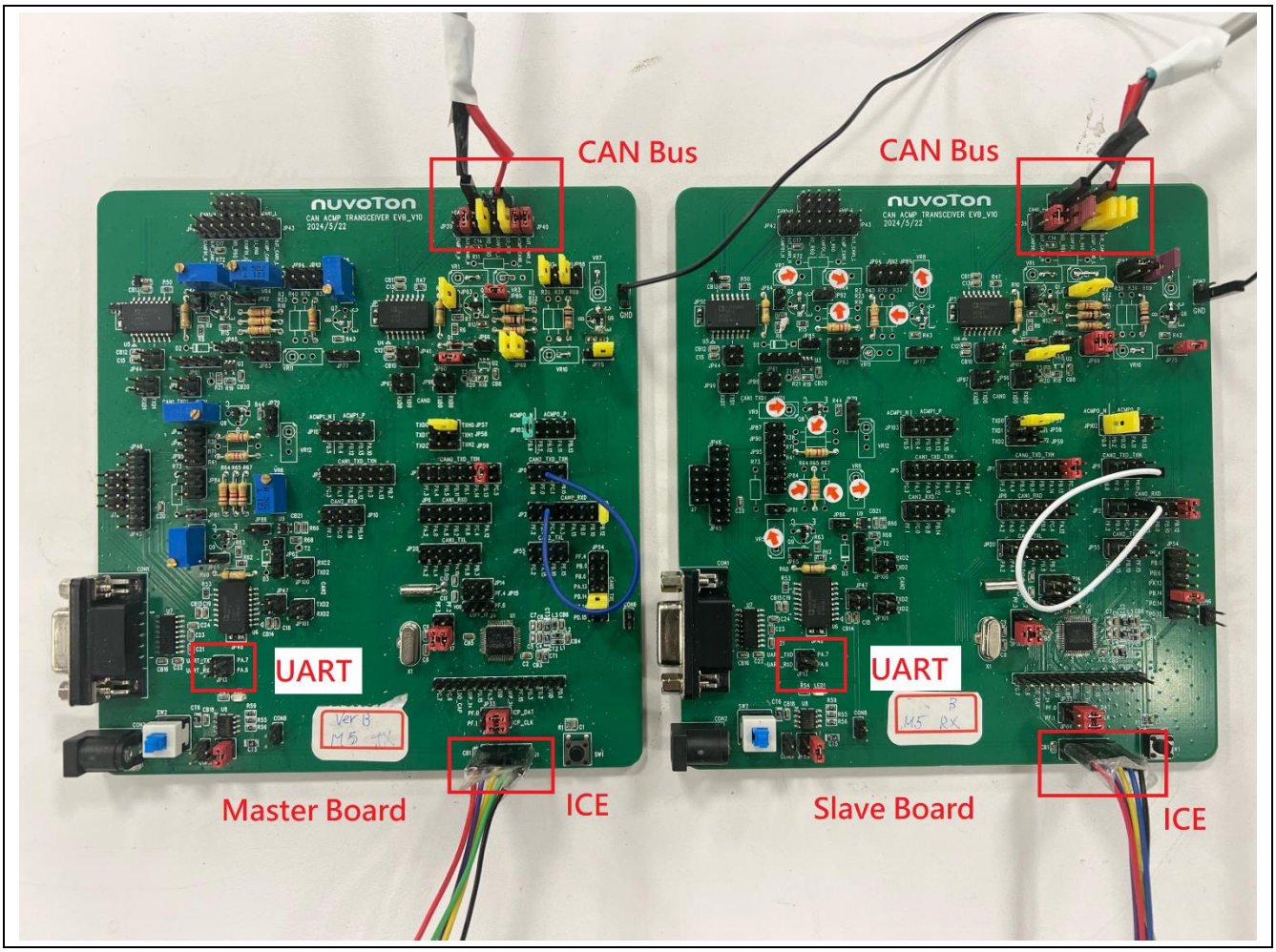


Figure 3-2 Physical Wiring

4. Directory Information

The directory structure is shown below.

EC_M2A23_CANFD_Using_ACMP_as_Transceiver _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	
M2A23_CANFD_ACMP_TxRx	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 *M2A23_CANFD_ACMP_TxRx.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.04.25	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.