SPI 转 CAN 的桥接传输

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码演示使用 M0A23 系列微控制器进行 SPI 与 CAN 之间桥接传输的操作。
BSP 版本	M0A23_Series_BSP_CMSIS_V3.03.000
开发平台	NuMaker-M0A23OC V1.1

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

本范例代码示范了如何建立一个 SPI 到 CAN 的网桥，让装置能通过 SPI 接口接收数据，并将这些数据通过 CAN 接口转送到另一个装置，用以實現裝置間相互傳送和接收信息。此网桥作为 SPI 与 CAN 接口之间的通信连接，实现两种通信协议之间的数据传输。

1.1 原理

当 SPI 接收到数据并确认封包内容无误后，会将该数据封装成 CAN 封包，并透过 CAN_Transmit() 函数进行传送。这个流程可确保只有有效且格式正确的数据会被转送到 CAN 总线上，从而维护跨通信接口间的数据完整性。透过将 SPI 接收的数据转换为 CAN 封包，系统能有效地在 SPI 装置与 CAN 之间建立通信桥接，使多协议的嵌入式系统实现互通。

传送的 CAN 数据将依照表 1-1 所示的格式进行。

Frame Type	ID Type	ID	DLC	Data
CAN_Data_FRAME	CAN_STD_ID	0x7FF	8	g_au32DestinationData

表 1-1 CAN 封包格式

相关流程与判断逻辑如图 1-1 所示。以下简述 CAN 主机端与从机端的程序流程:

CAN 主机端

步骤 1: 等待 CAN 从机端透过 USI_SPI 通信接口发送数据。

步骤 2: USI_SPI 数据接收完毕后则将其数据使用 CAN 通信接口传送给 CAN 从机端。

步骤 3: 等待 CAN 从机端发送回应封包，接收后则发送下一个 CAN 封包。

步骤 4: 直到数据全数传送完毕，等待 CAN 从机端发送 USI_SPI 数据并回到步骤 1。

CAN 从机端

步骤 1: 发送数据透过 USI_SPI 通信接口给 CAN 主机端。

步骤 2: USI_SPI 数据发送完毕后等待接收 CAN 主机端发送的封包。

步骤 3: 发送回应封包给 CAN 主机端，等待 CAN 主机端发送下一个封包并回到步骤 1。

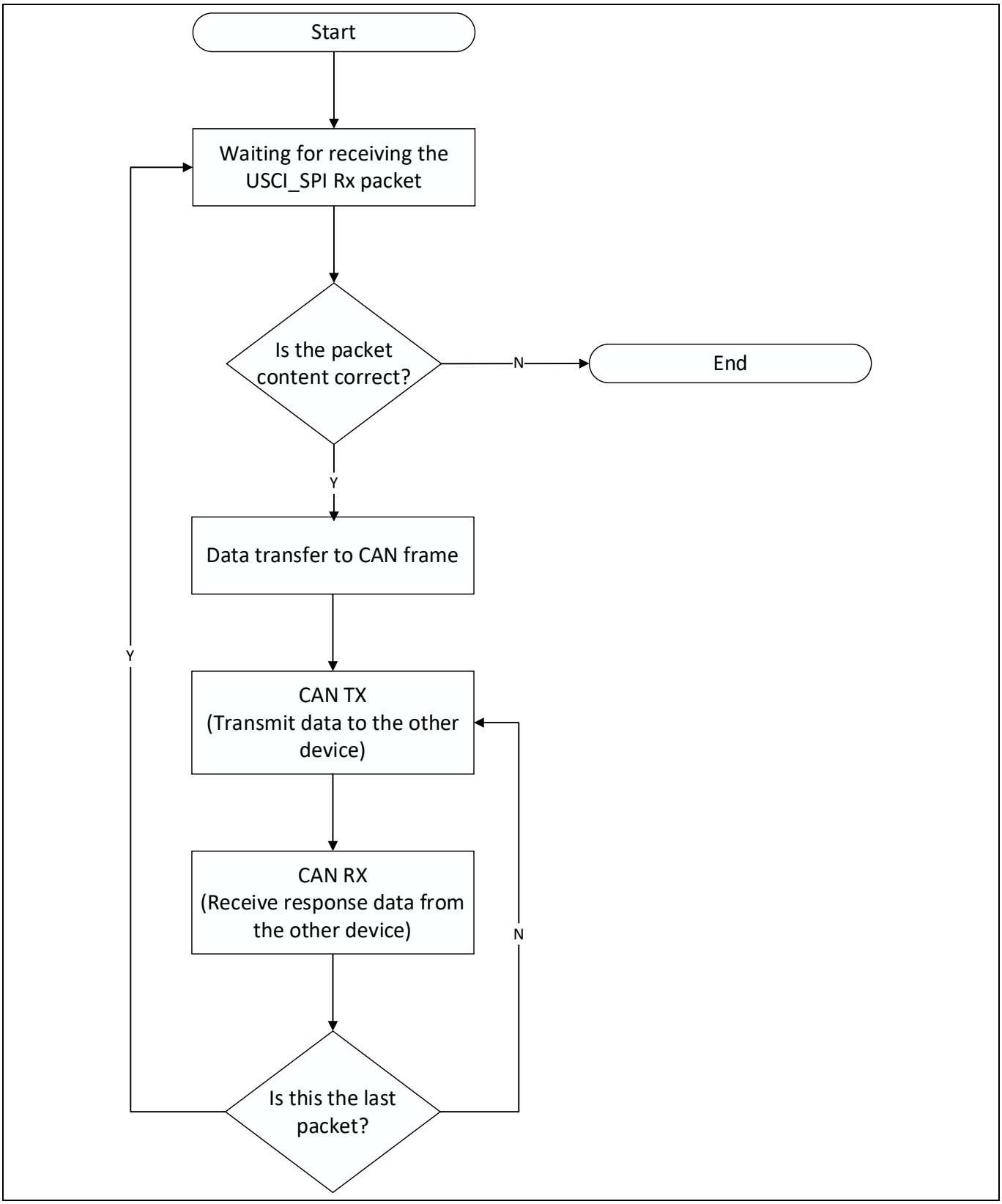


图 1-1 USCI_SPI 接收数据转换为 CAN 传输的过程

1.2 执行结果

开启计算机串口调试助手软件，设置 UART 传输速度为 115200 bps，无校验位，数据长度为 8 位，停止位长度为 1 位。CAN 主机端与从机端所接收到的封包内容如图 1-2 與图 1-3 所示。

```
CAN master is waiting for reciving USCI_SPI package...
Press any key to continue ...

USCI_SPI data compCAN master is waiting for reciving USCI_SPI package...
Press any key to continue ...
CAN master is waiting for reciving USCI_SPI package...
Press any key to continue ...

USCI_SPI data comparison successful.

Pkt=0 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=8 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=16 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=24 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=32 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=40 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=48 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Pkt=56 Read ID=     788, Type=STD, DLC=2,Data=4F,4B,

Transfor done

CAN master is waiting for reciving USCI_SPI package...
Press any key to continue ...
```

图 1-2 CAN 主机端串口调试助手屏幕截图

```
Press any key to continue ...

Send Msg:
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
Read ID=     7FF, Type=STD, DLC=8,Data=00,01,02,03,04,05,06,07,

Read ID=     7FF, Type=STD, DLC=8,Data=08,09,0A,0B,0C,0D,0E,0F,

Read ID=     7FF, Type=STD, DLC=8,Data=10,11,12,13,14,15,16,17,

Read ID=     7FF, Type=STD, DLC=8,Data=18,19,1A,1B,1C,1D,1E,1F,

Read ID=     7FF, Type=STD, DLC=8,Data=20,21,22,23,24,25,26,27,

Read ID=     7FF, Type=STD, DLC=8,Data=28,29,2A,2B,2C,2D,2E,2F,

Read ID=     7FF, Type=STD, DLC=8,Data=30,31,32,33,34,35,36,37,

Read ID=     7FF, Type=STD, DLC=8,Data=38,39,3A,3B,3C,3D,3E,3F,

Press any key to continue ...
```

图 1-3 CAN 从机端串口调试助手屏幕截图

2. 代码介绍

章节 2.1 与 2.2 将分别说明 USCI_SPI 与 CAN 的初始化设定、传送与接收功能。

2.1 USCI_SPI

初始化设定與接收功能如下，相关程序代码位于 *USCI_SPI_transfer.c* 中：

- 将 USCI_SPI0 设定为主机模式，频率为 2MHz

```
void USCI_SPI_Init(void)
{
    …

    /* Configure USCI_SPI0 as a master, clock idle low, 16-bit transaction,
         drive output on falling clock edge and latch input on rising edge. */
    /* Set USCI_SPI0 clock rate = 2MHz */
    USPI_Open(USPI0, USPI_SLAVE, USPI_MODE_0, 16, 2000000);

    …
}
```

USCI_SPI 从机端将接收到的数据与传送的数据进行比对，若封包内容正确比对无误，则会开始进行 CAN 封包的传送与接收。

```
uint8_t USCI_SPI_Rx(void)
{

    /* Access TX and RX Buffer */
    while (u32RxDataCount < TEST_COUNT)
    {
        /* Check RX EMPTY flag */
        if (USPI_GET_RX_EMPTY_FLAG(USPI0) == 0){
            /* Read RX Buffer */
            g_au32DestinationData[u32RxDataCount++] = USPI_READ_RX(USPI0);
         }
    }

    /* Check the USPI0 received data */
    for (u32RxDataCount = 0; u32RxDataCount < TEST_COUNT; u32RxDataCount++)
    {
        if (g_au32DestinationData[u32RxDataCount] != u32RxDataCount)
        {
            u32Err = 1;
            break;
        }
    }
    …
}
```

2.2 CAN

初始化设定與接收功能如下，相关程序代码位于 *USCI_SPI_transfer.c* 中：

- 将 CAN 设定为 500 kbps 的传输速率及 normal 模式，用于将 USCI_SPI 封包数据传输至另一装置。

```
void CAN_Init(CAN_T  *tCAN)
{
    /* Enable IP clock */
    CLK->APBCLK0 |= CLK_APBCLK0_CAN0CKEN_Msk;
}

void SelectCANSpeed(CAN_T  *tCAN)
{
    /* Set CAN speed = 500 Kbps */
    CAN_Open(tCAN,  500000, CAN_NORMAL_MODE);
}
```

CAN 相关的设定值，请参考表 1-1。

```
void CAN_Master_NormalMode_Tx(CAN_T *tCAN)
{
    STR_CANMSG_T tMsg;
    static uint32_t i = 0, j = 0;

    /* Send a 8 bytes message */
    tMsg.FrameType = CAN_DATA_FRAME;
    tMsg.IdType    = CAN_STD_ID;
    tMsg.Id        = 0x7FF;
    tMsg.DLC       = 8;

    if (u32TxLength < TEST_COUNT)
    {

        for (i = 0; i < 8; i++)
        {
            tMsg.Data[i]  = g_au32DestinationData[i + u32TxLength];
        }
         /* Configure Msg RAM and send the Msg in the RAM */
        if (CAN_Transmit(tCAN, MSG(0), &tMsg) == FALSE)
        {
            printf("Set Tx Msg Object failed\n");
            return;
        }

        u32TxLength += 8;

    }
}
```

CAN 封包传输完成后，会等待来自另一个 CAN 装置回传的信息封包。当确认收到响应后，才会开始下一个封包的传输。

```
void CAN_Master_NormalMode_Rx(CAN_T *tCAN)
{

    …
    /* Polling Mode */
    while (1)
    {
         /* Wait IDR is changed */
        while (tCAN->IIDR != 0x01);

        CAN_Receive(tCAN, tCAN->IIDR - 1, &rrMsg);

        if (rrMsg.Data[0] == 'O' && rrMsg.Data[1] == 'K')
        {
            CAN_ShowMsg(&rrMsg);
            u32RxPkgCnt++;
            break;
        }
    }
    …
     return 0;
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M0A21_Series_BSP_CMSIS_V3.03.000
- IDE 版本
  - Keil uVersion 5.28

3.2 硬件需求

- 电路组件
  - NuMaker-M0A23OC V1.1
- 线路示意图
  - USCI0_DAT0 (PC.5) pin 相当于 SPI_MOSI 功能脚位
  - USCI0_DAT1 (PC.4) pin 相当于 SPI_MISO 功能脚位
  - USCI0_CLK (PC.6) pin 相当于 SPI_CLK 功能脚位
  - USCI0_CTL0 (PC.7) pin 相当于 SPI_SS 功能脚位

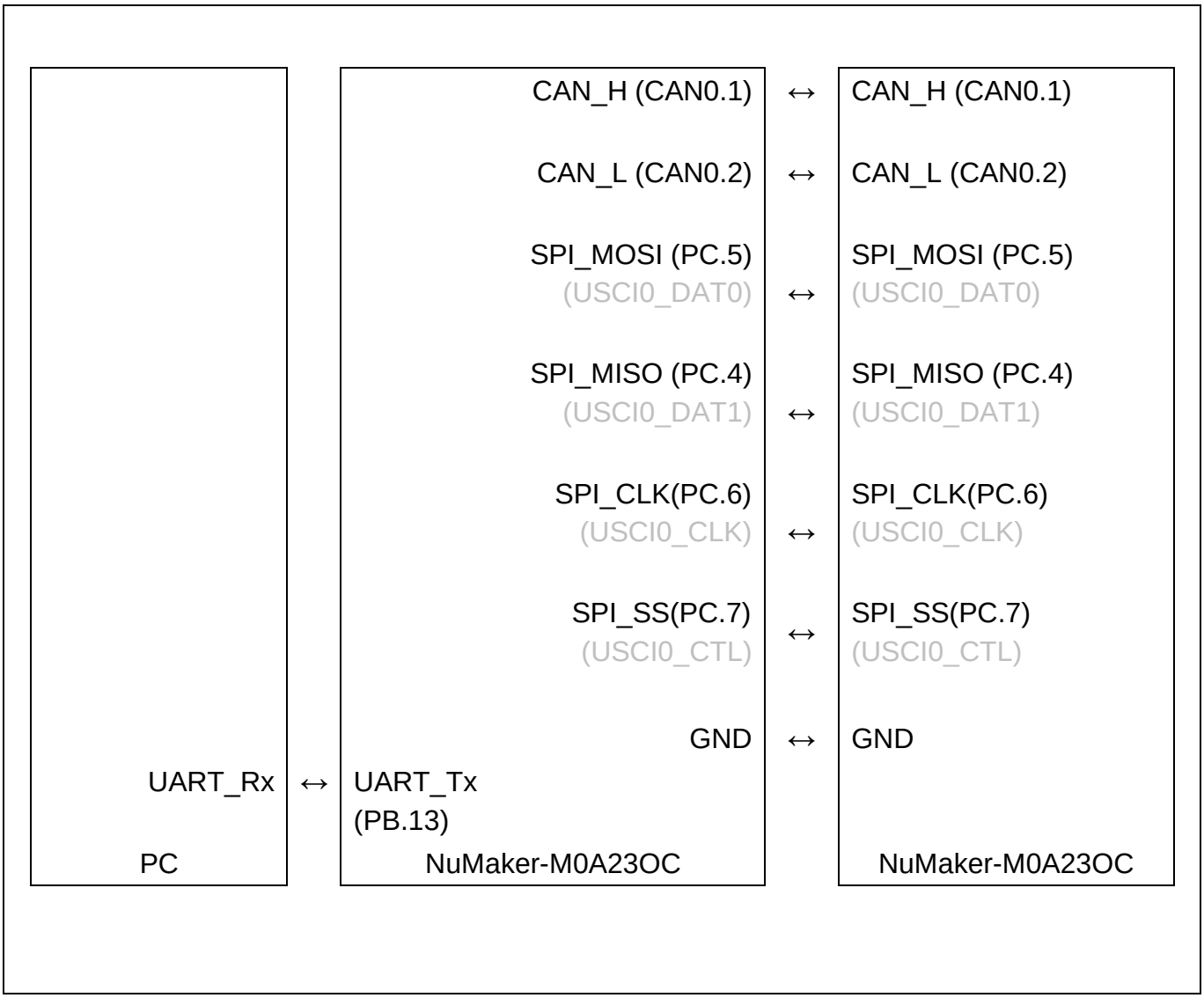

CAN_H (CAN0.1)
↔
CAN_H (CAN0.1)
CAN_L (CAN0.2)
↔
CAN_L (CAN0.2)
SPI_MOSI (PC.5)
(USCI0_DAT0)
↔
SPI_MOSI (PC.5)
(USCI0_DAT0)
SPI_MISO (PC.4)
(USCI0_DAT1)
↔
SPI_MISO (PC.4)
(USCI0_DAT1)
SPI_CLK(PC.6)
(USCI0_CLK)
↔
SPI_CLK(PC.6)
(USCI0_CLK)
SPI_SS(PC.7)
(USCI0_CTL)
↔
SPI_SS(PC.7)
(USCI0_CTL)
GND
↔
GND
UART_Rx
↔
UART_Tx
(PB.13)
PC
NuMaker-M0A23OC
NuMaker-M0A23OC

图 3-1 引脚连接图

4. 目录信息

目录结构如下图所示。

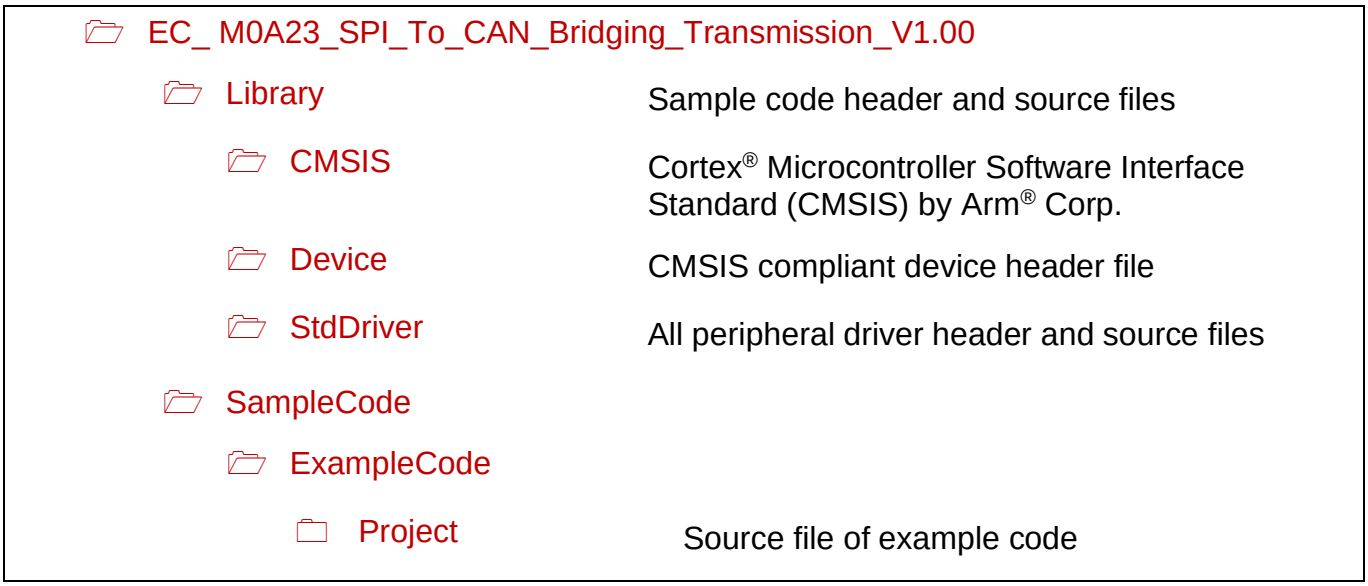


图 4-1 目录信息

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 *M0A23_SPI_To_CAN_Bridging_Transmission*.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
2025.06.16	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.