SPI 轉 CAN 的橋接傳輸

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	本範例代碼演示使用 M0A23 系列微控制器進行 SPI 與 CAN 之間橋接傳輸的操作
BSP 版本	M0A23_Series_BSP_CMSIS_V3.03.000
開發平台	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

本範例代碼示範了如何建立一個 SPI 到 CAN 的橋接器，讓裝置能透過 SPI 介面接收資料，並將這些資料透過 CAN 介面轉送到另一個裝置，用以實現裝置間相互傳送和接收訊息。此橋接器作為 SPI 與 CAN 介面之間的通訊連接，實現兩種通訊協定之間的資料傳輸。

1.1 原理

當 SPI 接收到資料並確認封包內容無誤後，會將該資料封裝成 CAN 封包，並透過 CAN_Transmit() 函式進行傳送。這個流程可確保只有有效且格式正確的資料會被轉送到 CAN 匯流排上，從而維護跨通訊介面間的資料完整性。透過將 SPI 接收的資料轉換為 CAN 封包，系統能有效地在 SPI 裝置與 CAN 之間建立通訊橋接，使多協定的嵌入式系統實現互通。

傳送的 CAN 資料將依照表 1-1 所示的格式進行。

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

程式流程與判斷邏輯如圖 1-1 所示。以下簡述 CAN 主機端與從機端的程式流程:

CAN 主機端

步驟 1: 等待 CAN 從機端透過 USI_SPI 通訊介面發送資料。

步驟 2: USI_SPI 資料接收完畢後則將其資料使用 CAN 通訊介面傳送給 CAN 從機端。

步驟 3: 等待 CAN 從機端發送回應封包，接收後則發送下一個 CAN 封包。

步驟 4: 直到資料全數傳送完畢，等待 CAN 從機端發送 USI_SPI 資料並回到步驟 1。

CAN 從機端

步驟 1: 發送資料透過 USI_SPI 通訊介面給 CAN 主機端。

步驟 2: USI_SPI 資料發送完畢後等待接收 CAN 主機端發送的封包。

步驟 3: 發送回應封包給 CAN 主機端，等待 CAN 主機端發送下一個封包並回到步驟 1。

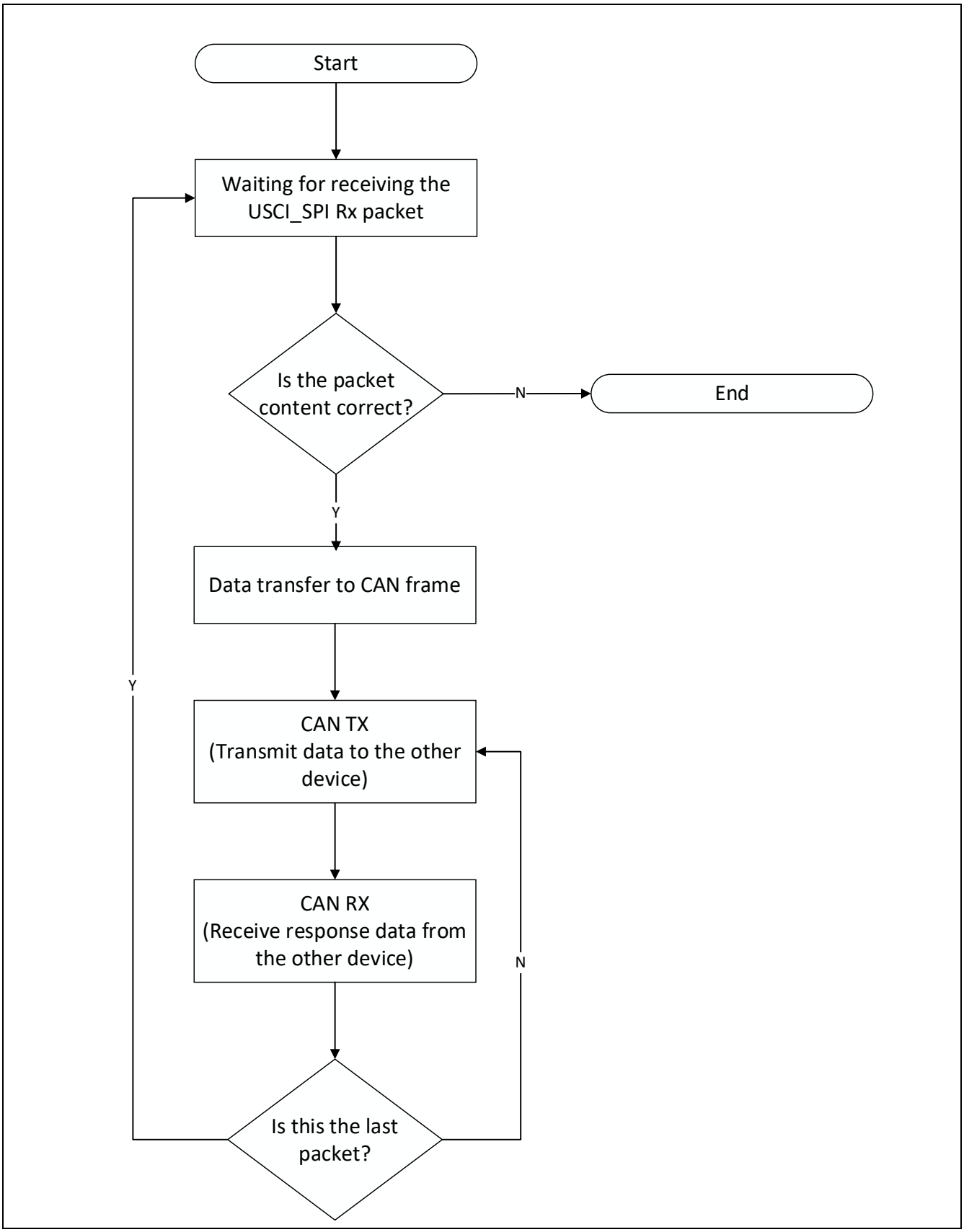


圖 1-1 USCI_SPI 接收資料轉換為 CAN 傳輸的過程

1.2 執行結果

開啟電腦終端機軟體，設置 UART 傳輸速度為 115200 bps，無校驗位元，數據長度為 8 位元，停止位元長度為 1 位元。CAN 主機端與從機端所接收到的封包內容分別如圖 1-2 與圖 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 ...
```

圖 1-2 CAN 主機端終端機螢幕截圖

```
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 ...
```

圖 1-3 CAN 從機端終端機螢幕截圖

2. 程式碼介紹

章節 2.1 與 2.2 將分別說明 USCI_SPI 與 CAN 的初始化設定、傳送與接收功能。

2.1 USCI_SPI

初始化設定與接收功能如下，相關程式碼位於 *USCI_SPI_transfer.c* 中：

- 將 USCI_SPI0 設定為主機模式，時脈為 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 從機端將接收到的資料與傳送的資料進行比對。若封包內容正確比對無誤，則會開始進行 CAN 封包的傳送與接收。

```
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

初始化設定，傳輸與接收功能如下，相關程式碼位於 *CAN_trainfer.c* 中。

- 將 CAN 設定為 500 kbps 的傳輸速率及 normal 模式，用於將 USCI_SPI 封包資料轉送至另一裝置。

```
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 相關的設定值，請參考表 1-1。

```
void CAN_Master_NormalMode_Tx(CAN_T *tCAN)
{
    STR_CANMSG_T tMsg;
    static uint32_t i = 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 封包傳輸完成後，會等待來自另一個 CAN 裝置回傳的訊息封包。當確認收到回應後，才會開始下一個封包的傳輸。

```
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. 軟體與硬體需求

3.1 軟體需求

- BSP 版本

  - M0A21_Series_BSP_CMSIS_V3.03.000

- IDE 版本

  - Keil uVision 5.28

3.2 硬體需求

- 電路元件

  - NuMaker-M0A23OC V1.1

- 線路示意圖

  - USCI0_DAT0 (PC.5) pin 相當於 SPI_MOSI 功能腳位
  - USCI0_DAT1 (PC.4) pin 相當於 SPI_MISO 功能腳位
  - USCI0_CLK (PC.6) pin 相當於 SPI_CLK 功能腳位
  - USCI0_CTL0 (PC.7) pin 相當於 SPI_SS 功能腳位

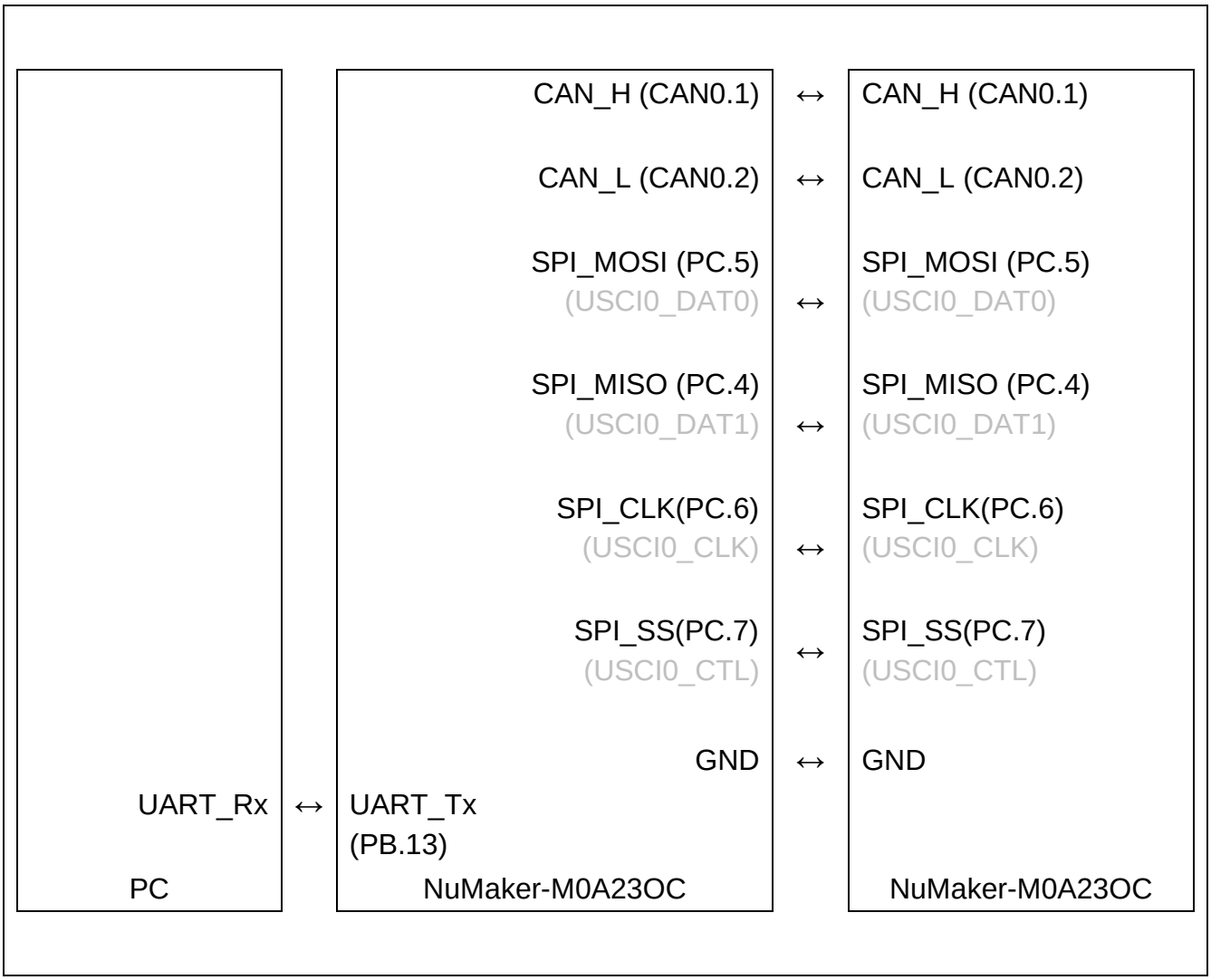

CAN_H (CAN0.1)
↔
CAN_H (CAN0.1)
CAN_L (CAN0.2)
↔
CAN_L (CAN0.2)
SPI_MOSI (PC.5)
(USCI0_DAT0)
↔
SPI_MOSI (PC.5)
(USCI0_DAT0)
SPI_MISO (PC.4)
(USCI0_DAT1)
↔
SPI_MISO (PC.4)
(USCI0_DAT1)
SPI_CLK(PC.6)
(USCI0_CLK)
↔
SPI_CLK(PC.6)
(USCI0_CLK)
SPI_SS(PC.7)
(USCI0_CTL)
↔
SPI_SS(PC.7)
(USCI0_CTL)
GND
↔
GND
UART_Rx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

圖 3-1 線路示意圖

4. 目錄資訊

目錄結構如下圖所示。

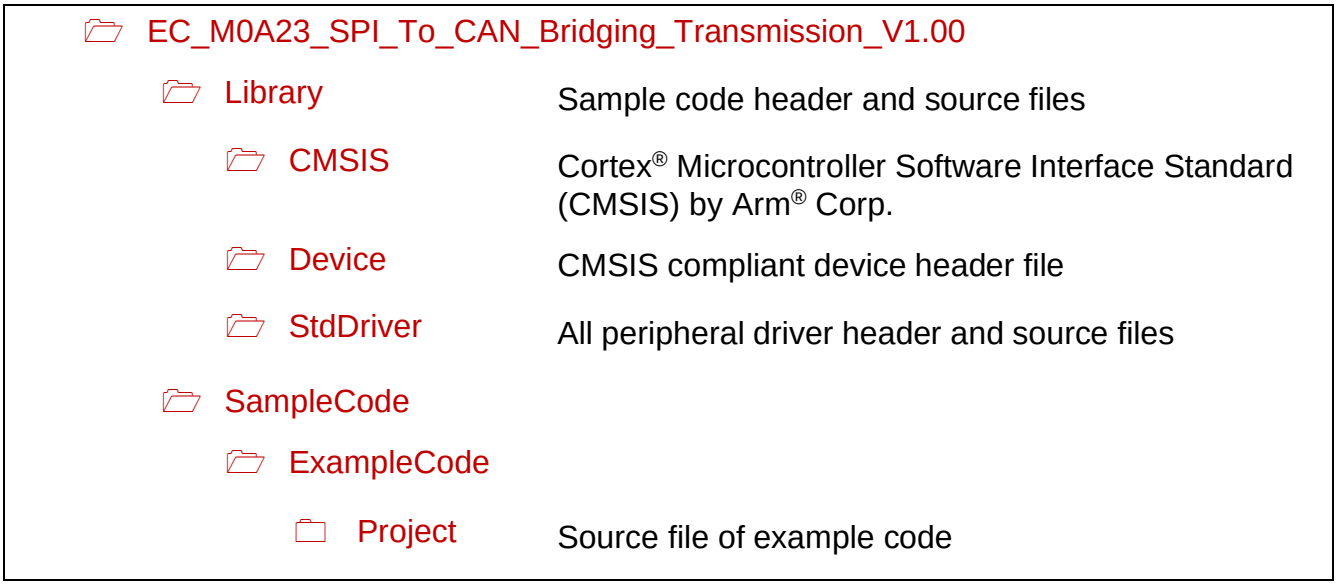


圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *M0A23_SPI_To_CAN_Bridging_Transmission.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載程式碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行程式碼

6. 修訂紀錄

Date	Revision	Description
2025.06.16	1.00	初始發布。

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.