Application Note

使用 MSPM0 MCU 在 CAN 和 SPI 之间进行桥接设计

Yuhao Zhao

摘要

本应用手册展示了一个将 CAN 连接到 SPI 桥接器的示例。本文描述了 CAN 转 SPI 桥接器的结构和行为。文中还讨论了软件实现、硬件连接和应用使用。用户可以通过修改预定义来配置桥接器。相关代码也将提供给用户。

内容

插图清单

表格清单

商标

LaunchPad™ is a trademark of Texas Instruments.
所有商标均为其各自所有者的财产。

1 简介

器件之间有多种通信方法，具体取决于应用。当今的 MCU 通常支持多种通信方法。例如，MSPM0 可以在特定器件上支持 UART、SPI、CAN 等。当器件需要通过不同的通信接口传输数据时，可以构建一个桥接器。

对于 CAN 和 SPI，CAN-SPI 桥接器充当两个接口之间的转换器。CAN-SPI 桥接器使器件能够在一个接口上发送或接收信息，并在另一个接口上接收或发送信息。

本应用手册介绍了创建和使用 CAN-SPI 桥接器时使用的软件和硬件设计。MSPM0G3507 微控制器 (MCU) 可通过提供 CAN 和 SPI 通信接口来使用。随附的演示使用具有 2Mbps CANFD 和 500k 比特率 SPI 的 MSPM0G3507 来演示在通道之间收发数据。

1.1 在 CAN 与 SPI 之间建立桥接

CAN-SPI 桥接器可连接 CAN 和 SPI 接口。该桥接器支持 SPI 在从机模式或主机模式下工作。本文档中的示例使用 CAN 分析仪来观察 CAN 数据。用户还可以通过 CAN-SPI 桥接器，从 CAN 分析仪向 SPI 侧发送消息。对于 SPI 数据，用户可以使用逻辑分析仪或使用两个 LaunchPAD 形成一个循环来进行观察，例如 图 4-1 中随附的演示。

本文中的示例支持透明传输和协议传输。图 1-1 展示了逻辑分析仪对透明传输的观察结果。图 1-2 展示了逻辑分析仪对协议传输的观察结果。

对于协议传输，此示例指定了 SPI 消息格式。用户可以根据应用需求来修改格式。从 SPI 接收消息时，消息格式为 < 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2 ...> 用户可以使用相同的格式通过 SPI 发送数据。55 AA 是标头。ID 区域为 4 字节。长度区域为 1 字节，表示数据长度。

对于透明传输，SPI 从机使用可配置的超时来检测一条消息。来自 SPI 的数据被填充到 CAN 的数据区域（反之亦然）。CAN ID 是默认值。

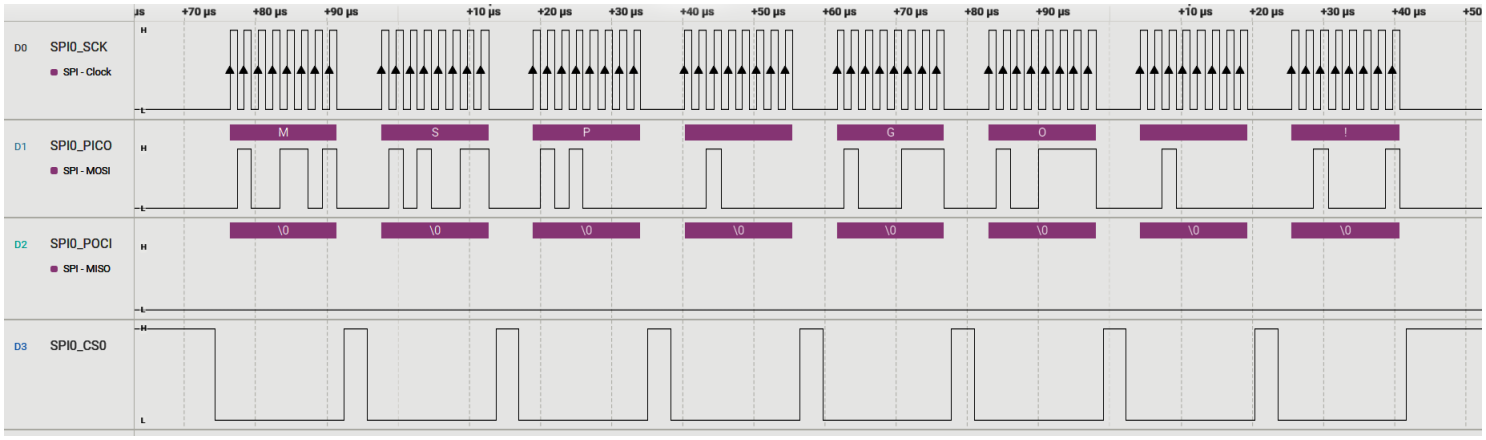

图 1-1. 用于 SPI 透明传输的逻辑分析仪

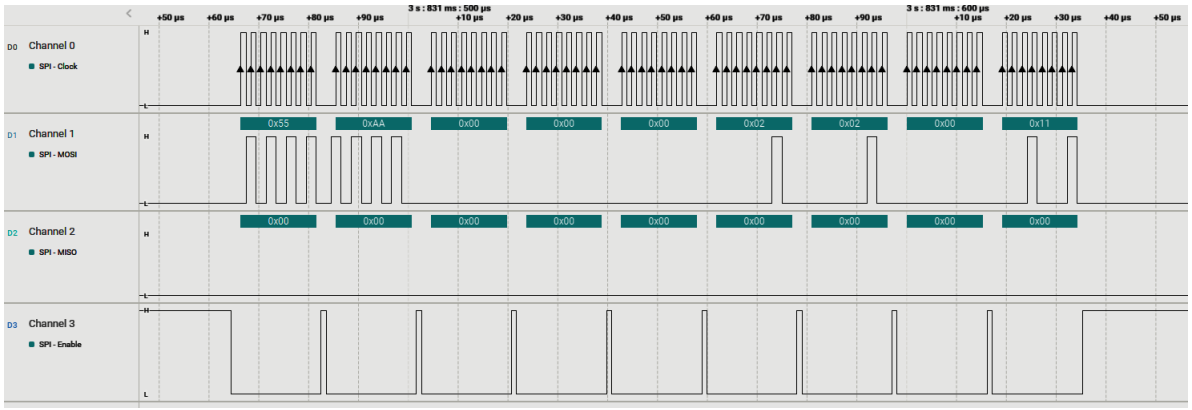

图 1-2. 用于 SPI 协议传输的逻辑分析仪

2 实施

2.1 原理

在本文档中，CAN-SPI 桥接器使用 CAN 接收和传输功能及 SPI 接收和传输功能。所以，必须配置 CAN 模块和 SPI 模块。由于不同通信的消息格式是不同的，CAN-SPI 桥接器还必须转换消息格式。

对于 CAN，CAN 模块支持传统 CAN 和 CAN FD（具有灵活数据速率的 CAN）协议。CAN 模块符合 ISO 11898-1:2015 标准。如需更多信息，请参阅相关文档。对于 SPI，该接口可用于在 MSPM0 器件和另一个采用串行异步通信协议的器件之间传送数据。如需更多信息，请参阅相关文档。由于 SPI 从机的接收和传输由 SPI 主机控制，因此 SPI 从机无法发起到 SPI 主机的传输。为了实现从机到主机的通信，该设计中增加了一条线路。从机的 IO 下拉会通知主机必须发送信息。

图 2-1 所示为 CAN-SPI 桥接器的基本原理。通常，CAN 的通信速率与 SPI 的通信速率不同。对于 CAN FD，波特率最高可达 5Mbps，而 SPI 以 500k 比特率运行，如示例代码所示。因此，一个接口接收到的数据可能不会及时由另一个接口发送。为了匹配速率，该方案使用缓冲器在 CAN 和 SPI 之间传输数据。此缓冲区不仅实现数据缓存，还实现数据格式转换。这相当于在两个通信接口之间添加了屏障。用户可以针对过载情况，添加过载控制操作。

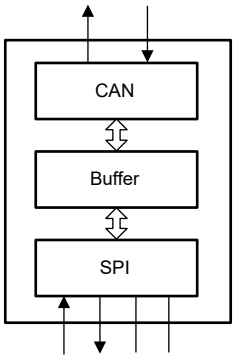

图 2-1. CAN-SPI 桥接器的基本原理

2.2 结构

在 图 2-3 和 图 2-4 中，可以看到具有协议传输功能的 CAN-SPI 桥接器的结构。图 2-3 用于 SPI 主机，图 2-4 用于 SPI 从机。CAN-SPI 桥接器可以分为四个独立的任务：从 SPI 接收、从 CAN 接收、通过 CAN 传输、通过 SPI 传输。两个 FIFO 实现双向消息传输和消息缓存。

具有透明传输功能的 CAN- SPI 桥接器的结构如 图 2-3 和 图 2-5 所示。图 2-3 用于 SPI 主机，图 2-5 用于 SPI 从机。将计时器中断添加到 CAN-SPI（SPI 目标）桥接器，以检测一个数据包结束时的超时情况。

SPI 和 CAN 接收均设置为中断触发，以便及时接收报文。进入中断时，首先通过 *getXXXRxMsg()* 接收消息。

对于 CAN，CAN 帧是固定格式。MSPM0 支持传统 CAN 或 CANFD。图 2-2 显示了 CANFD 帧。本文中的示例可以在协议传输的数据区域中定义 0/1/4 字节的附加 ID，如 表 2-1 所示。

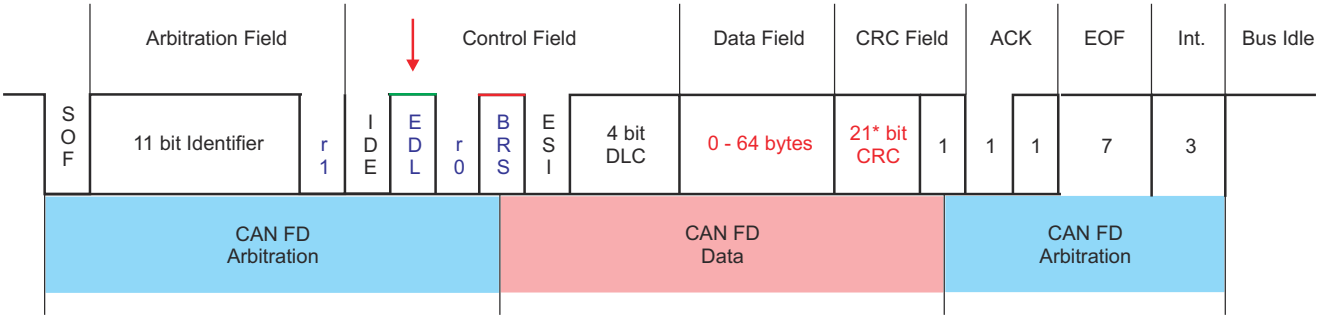

图 2-2. CAN FD 帧

表 2-1. CAN 数据包形式

	ID 区域	数据
协议传输	4/1/0 字节	（数据长度）字节

对于 SPI 协议传输，基于串行帧信息来识别消息。表 2-2 列出了 SPI 消息格式。

表 2-2. SPI 数据包形式

	接头	ID 区域	数据长度	数据
协议传输	0x55 0xAA	4/1/0 字节	1 字节	（数据长度）字节
透明传输	—	—	—	主机到从机 -（数据长度）字节 从机到主机 -（SPI_TRANSPARENT_LENGTH）字节

标头是一个固定的十六进制数，与 *0x55 0xAA* 组合在一起，表示组的开头。ID 区域默认占用 4 个字节，以匹配 CAN ID，可配置为 1 个字节，也可以不存在。数据长度区域占用 1 个字节。在数据长度区域之后，会跟随一定长度的数据。此格式作为示例提供。用户可以根据应用需求来修改格式。

请注意，SPI 是一种通信方法，SPI 主机负责控制发送和接收。通常，SPI 从机无法发起通信。对于 SPI 从机到主机通信，SPI 从机可以在必须发送消息时下拉 IO，如 图 2-4 中所示。当检测到 IO 为低电平时，SPI 从机可以在 IO 中断中启动 SPI 读取命令，如 图 2-3 所示。

对于 SPI 透明传输，超时发生时（在 SPI 从机上），会识别消息，如 图 2-4 所示。所有字节均视为纯数据。加载数据包信息的默认值（例如 ID）。

收到消息后，*processXXXRxMsg()* 会转换消息的格式，并将消息作为新元素存储在 FIFO 中。FIFO 元素的格式可以在 图 2-6 中看到。在 FIFO 元素的格式中，有 *origin_id*、*destination_id*、*data length* 和 *data* 函数。用户还

可以根据应用程序要求来更改消息项目。此外，该方案还会检查 FIFO 是否已满，以进行过载控制。用户可以在需求变化时添加过载控制操作。

CAN 和 SPI 传输均在主函数中执行。当 FIFO 不为空时，将提取 FIFO 元素。消息将被格式化并发送。对于 CAN，CAN 帧为固定格式，如 表 2-1 所示。对于 SPI，消息以 SPI 数据包表单中列出的格式发送。

请注意，SPI 是一种发送和接收同步的通信方法。当主机启动发送一个字节时，主机期望接收一个字节。在本文的设计中，SPI RX 中断不仅用于 SPI 接收，还用于将 TX 数据填充到 SPI TX FIFO 中。如果 SPI 在主机模式下运行，则在 SPI TX FIFO 存入数据后，SPI 通信会立即开始。如果 SPI 在从机模式下运行，则在存储数据后，SPI 使用 IO 来触发主机以发起通信。在本演示中，用户可以选择 SPI 模式。

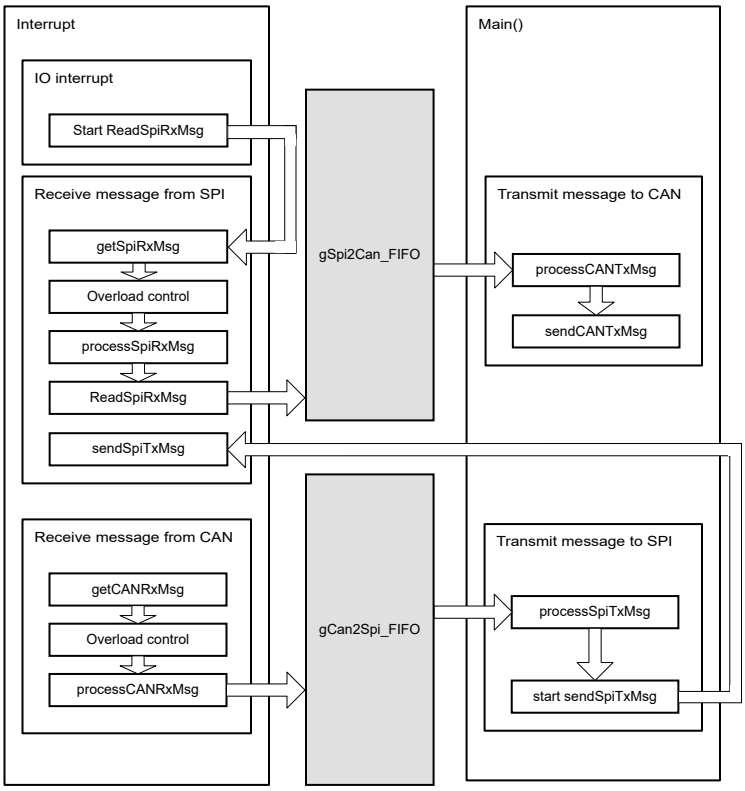

图 2-3. CAN-SPI（SPI 控制器）桥接器的结构：协议和透明

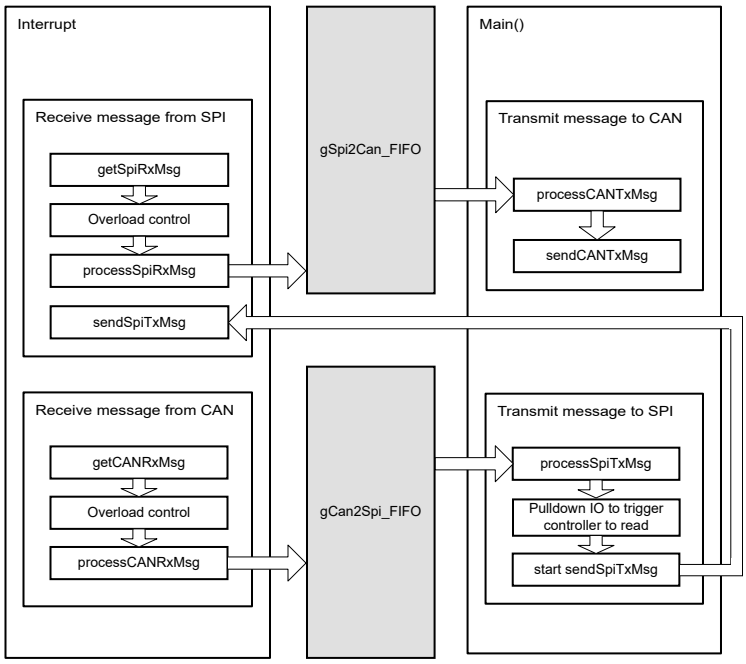

图 2-4. CAN-SPI（SPI 目标）桥接器的结构：协议

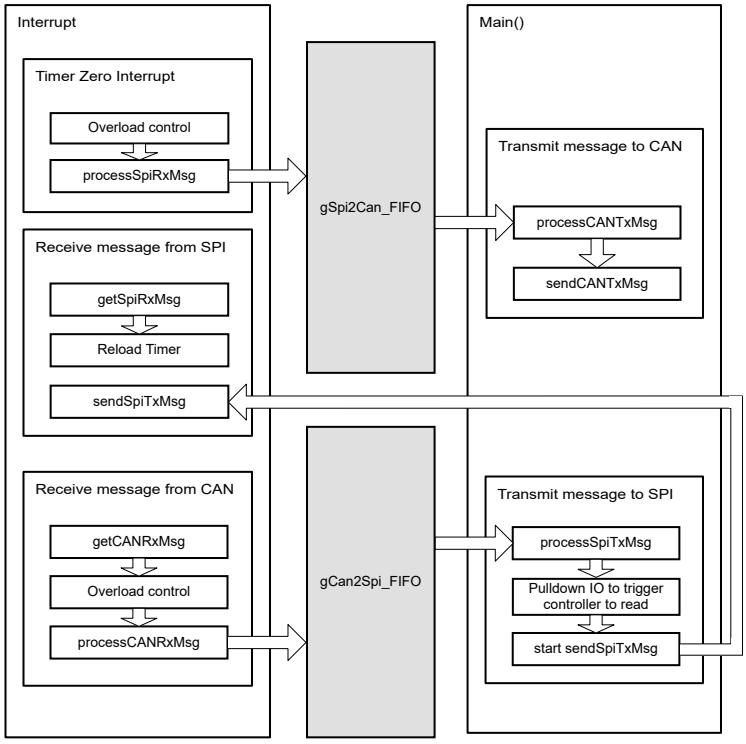

图 2-5. CAN-SPI（SPI 目标）桥接器的结构：透明

图 2-6 展示了 FIFO 结构。每个 FIFO 使用三个全局变量来指示 FIFO 状态。对于 *gSpi2Can_FIFO*，*gSpi2Can_FIFO.fifo_in* 表示写入位置，*gSpi2Can_FIFO.fifo_out* 表示读取位置，*gSpi2Can_FIFO.fifo_Count* 表示 *gSpi2Can_FIFO* 中的元素数量。

如果 *gSpi2Can_FIFO* 为空，则 *gSpi2Can_FIFO.fifo_in* 等于 *gSpi2Can_FIFO.fifo_out*，*gSpi2Can_FIFO.fifo_count* 为零。

执行 *processSpiRxMsg()* 时，来自 SPI 的新消息将存储到 *gSpi2Can_FIFO* 中。因此，*gSpi2Can_FIFO.fifo* 会移动到下一个位置，并且 *gSpi2Can_FIFO.fifo_count* 会递增 1。

从 *gSpi2Can_FIFO* 向 CAN 传输消息时，*gSpi2Can_FIFO.fifo_out* 移动到下一个位置，而 *gSpi2Can_FIFO.fifo_count* 减 1。*gCan2Spi_FIFO* 与 *gSpi2Can_FIFO* 类似。

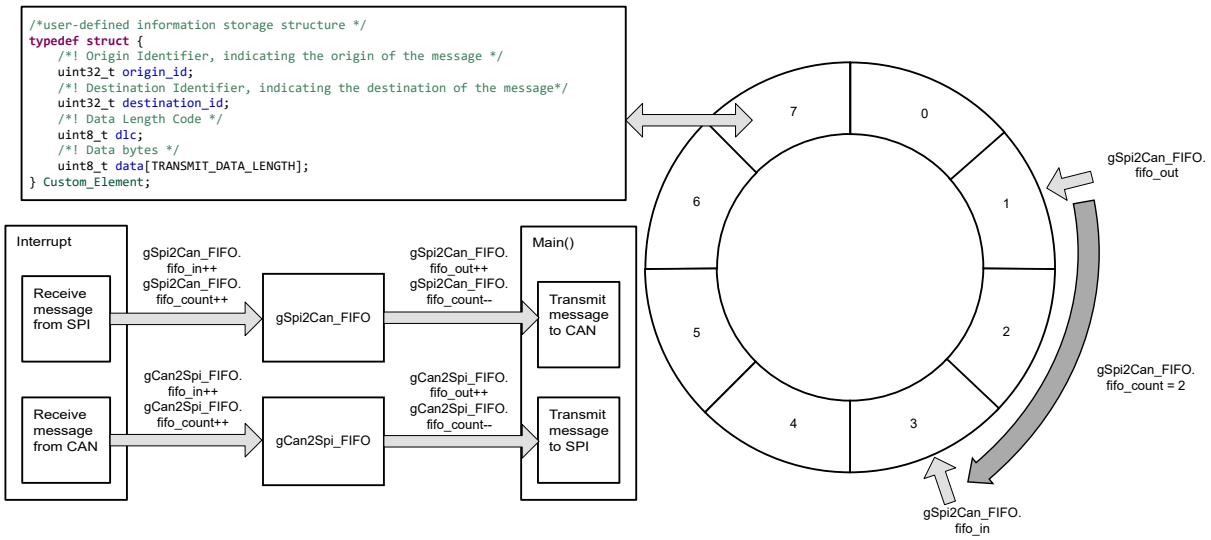

图 2-6. FIFO 的结构

3 软件说明

3.1 软件功能

这些函数是根据 图 2-3 设计的。表 3-1 中列出了相关函数。

表 3-1. 函数和说明

任务	函数	说明	位置
SPI 接收	readSpiRxMsg()	发送字节以接收消息（仅限 SPI 主机）	bridge_spi.c bridge_spi.h
	getSpiRxMsg()	获取接收到的 SPI 消息（协议）	
	getSpiRxMsg_transparent()	观察接收到的 SPI 消息（透明）	
	processSpiRxMsg()	转换接收到的 SPI 消息格式（协议），并将消息存储到 gSPI_RX_Element 中	
	processSpiRxMsg_transparent()	转换接收到的 SPI 消息格式（透明），并存储到 gSPI_RX_Element 中	
SPI 发送	processSpiTxMsg()	转换要通过 SPI 发送的 gSPI_TX_Element 格式（协议）	
	processSpiTxMsg()	转换要通过 SPI 发送的 gSPI_TX_Element 格式（透明）	
	sendSpiTxMsg()	通过 SPI 发送消息	
CAN 接收	getCANRxMsg()	获取接收到的 CAN 消息	bridge_can.c bridge_can.h
	processCANRxMsg()	转换接收到的 CAN 消息格式，并将消息存储到 gCAN_RX_Element 中	
CAN 发送	processCANTxMsg()	转换要通过 CAN 发送的 gCAN_TX_Element 格式	
	sendCANTxMsg()	通过 CAN 发送消息	

3.2 可配置参数

所有可配置参数都在 *user_define.h* 中定义，如 表 3-3 中所列。

对于 SPI，此示例同时支持透明传输和协议传输，即通过定义 SPI_TRANSPARENT 或 SPI_PROTOCOL 进行切换。

在透明传输中，用户可以配置 SPI 从机的超时，以检测一条 SPI 消息接收完成。用户还可以为从 SPI 从机到 SPI 主机的消息配置默认数据长度 (SPI_TRANSPARENT_LENGTH)。表 3-2 列出了在不同模式下接收或发送的字节数。

在协议传输中，用户可以为不同格式配置 ID 长度。请注意有一个固定的 2 字节标头 (0x55 0xAA) 和一个字节的数据长度。若要更详细地修改格式，用户可以直接修改代码。

```
#define SPI_TRANSPARENT
#ifdef SPI_TRANSPARENT
/* The format of SPI:
     * Transparent transmission - Data1 Data2 ...*/
/* data length for SPI master receiving or SPI slave transmitting*/
#define SPI_TRANSPARENT_LENGTH  (8)     //need be <= TRANSMIT_DATA_LENGTH
#define SPI_TIMEOUT     (0x4000)      //timeout 250ms
#else
/* The format of SPI:
     * if SPI_ID_LENGTH = 4, format is 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2 ...
     * if SPI_ID_LENGTH = 1, format is 55 AA ID Length Data1 Data2 ...
     * if SPI_ID_LENGTH = 0, format is 55 AA Length Data1 Data2 ...*/
//#define SPI_ID_LENGTH  (0)
//#define SPI_ID_LENGTH  (1)
#define SPI_ID_LENGTH  (4)
#endif
```

表 3-2. 用于在不同 SPI 模式下进行接收或发送的字节数

参数	SPI 接口：主机		SPI 接口：从器件	
	接收到多少字节？	发送了多少字节？	接收到多少字节？	发送了多少字节？
协议传输	(2+SPI_ID_LENGT+1+Length) 字节	(2+SPI_ID_LENGT+1+Length) 字节	(2+SPI_ID_LENGT+1+Length) 字节	(2+SPI_ID_LENGT+1+Length) 字节
透明传输	(SPI_TRANSPARENT_LENGTH) 字节	(Length) 字节	超时用于标识消息结束	(SPI_TRANSPARENT_LENGTH) 字节

对于 CAN，CAN 帧中包含 ID 或数据长度。用户可以通过更改 CAN_ID_LENGTH (默认值为 0) 来在数据区域中添加另一个 ID。

```
/* The format of CAN:
     * if CAN_ID_LENGTH = 4, format is ID1 ID2 ID3 ID4 Data1 Data2 ...
     * if CAN_ID_LENGTH = 1, format is ID Data1 Data2 ...
     * if CAN_ID_LENGTH = 0, format is Data1 Data2 ...*/
#define CAN_ID_LENGTH  (0)
//#define CAN_ID_LENGTH  (1)
//#define CAN_ID_LENGTH  (4)
```

表 3-3. 可配置参数

参数	可选值	说明
#define SPI_TRANSPARENT	定义/未定义	启用 SPI 透明传输。
#define SPI_PROTOCOL	定义/未定义	启用 SPI 协议传输。
#defineSPI_TRANSPARENT_LENGTH (8)		SPI 从机到 SPI 主机的消息的默认数据长度。仅在定义了 SPI_TRANSPARENT 时可用。在本例中，默认值为八个字节。
#define SPI_TIMEOUT (0x4000)	超时 = SPI_TIMEOUT/ 32768 s	超时用于指示一条 SPI 消息完成接收。仅在定义了 SPI_TRANSPARENT 时可用。在本例中，默认值为 250ms。
#define SPI_ID_LENGTH (4)	0/1/4	可选 SPI ID 长度，与协议中的 ID 区域相关。仅在定义了 SPI_PROTOCOL 时可用。在本例中，默认值为四个字节。
#define CAN_ID_LENGTH (0)	0/1/4	可选 CAN ID 长度，与协议中的 ID 区域相关。在本例中，默认值为 0 字节
#define TRANSMIT_DATA_LENGTH (12)	<=64	数据区域的大小。如果接收到的消息包含的数据多于此值，会导致数据丢失
#define C2S_FIFO_SIZE (8)		CAN 到 SPI FIFO 的大小。请注意 SRAM 的使用量。
#define S2C_FIFO_SIZE (8)		SPI 到 CAN FIFO 的大小。请注意 SRAM 的使用量。
#define DEFAULT_SPI_ORIGIN_ID (0x00)		SPI 原始 ID 的默认值
#define DEFAULT_SPI_DESTINATION_ID (0x00)		SPI 目标 ID 的默认值
#define DEFAULT_CAN_ORIGIN_ID (0x00)		CAN 原始 ID 的默认值
#define DEFAULT_CAN_DESTINATION_ID (0x00)		CAN 目标 ID 的默认值

3.3 自定义元件的结构

Custom_Element 是在 user_define.h 中定义的结构。*Custom_Element* 也显示在 图 2-6 中。

*来源标识符*用于指示消息的来源。以下是*来源标识符*示例（CAN_ID_LENGTH =0，SPI_ID_LENGTH =4）。

- 示例 1：CAN 接口接收和传输
  1. 当 CAN-SPI 桥接器接收到 CAN 消息时，来自 CAN 帧的 ID 是*来源标识符*，指示消息来自何处。
  2. 当 CAN-SPI 桥接器传输 CAN 消息时，*来源标识符*被忽略（CAN_ID_LENGTH 默认设置为 0）。
- 示例 2：SPI 接口接收和传输（SPI 协议传输）
  1. 当 CAN-SPI 桥接器接收到 SPI 消息（SPI 协议传输）时，DEFAULT_SPI_ORIGIN_ID 是*来源标识符*，因为 SPI 没有 ID。
  2. 当 CAN-SPI 桥接器传输 SPI 消息（SPI 协议传输）时，*来源标识符*是 SPI 数据中的 4 字节 ID（SPI_ID_LENGTH 默认设置为 4），指示消息来自何处。
- 示例 3 - SPI 接口接收和传输（SPI 透明传输）
  1. 当 CAN-SPI 桥接器接收到 SPI 消息（SPI 透明传输）时，DEFAULT_SPI_ORIGIN_ID 是*来源标识符*，因为 SPI 没有 ID。
  2. 当 CAN-SPI 桥接器传输 SPI 消息（SPI 透明传输）时，*来源标识符*会被忽略。（透明传输没有 ID 区域）。

*目标标识符*用于指示消息的目标。

- 示例 1 - CAN 接口接收和传输
  1. 当 CAN-SPI 桥接器接收 CAN 消息时，DEFAULT_CAN_DESTINATION_ID 是*目标标识符*，因为 CAN_ID_LENGTH 默认设置为 0。SPI 传输不需要 ID。
  2. 当 CAN-SPI 桥接器传输 CAN 消息时，*目标标识符*是 CAN 帧中的 CAN ID。在此示例中，11 位或 29 位均得到支持。
- 示例 2 - SPI 接口接收和传输（SPI 协议传输）
  1. 当 CAN-SPI 桥接器接收到 SPI 消息（SPI 协议传输）时，来自 SPI 数据的 4 字节 ID 是目标标识符（SPI_ID_LENGTH 默认设置为 4）。CAN 传输需要 ID 信息。
  2. 当 CAN-SPI 桥接器传输 SPI 消息（SPI 协议传输）时，*目标标识符*将被忽略，因为 SPI 传输不需要 ID。
- 示例 3 - SPI 接口接收和传输（SPI 透明传输）
  1. 当 CAN-SPI 桥接器接收到 SPI 消息（SPI 透明传输）时，DEFAULT_SPI_DESTINATION_ID 是*目标标识符*（透明传输没有 ID 区域）。CAN 传输需要 ID 信息。
  2. 当 CAN-SPI 桥接器传输 SPI 消息（SPI 透明传输）时，*目标标识符*将被忽略，因为 SPI 传输不需要 ID。

```
/*user-defined information storage structure */
typedef struct {
    /*! Origin Identifier, indicating the origin of the message */
    uint32_t origin_id;
    /*! Destination Identifier, indicating the destination of the message */
    uint32_t destination_id;
    /*! Data Length Code */
    uint8_t dlc;
    /*! Data bytes */
    uint8_t data[TRANSMIT_DATA_LENGTH];
} Custom_Element;
```

3.4 FIFO 的结构

Custom_FIFO 结构在 *user_define.h* 中定义。图 2-6 中也展示了定义。

```
typedef struct {
    uint16_t fifo_in;
    uint16_t fifo_out;
    uint16_t fifo_count;
    Custom_Element *fifo_pointer;
} Custom_FIFO;
```

gCan2Spi_FIFO 和 *gSpi2Can_FIFO* 在 *main.c* 中定义。请注意 SRAM 的用法，它与 *C2S_FIFO_SIZE*、*S2C_FIFO_SIZE* 以及 *Custom_Element* 的大小有关。

```
/* Variables for C2S_FIFO
 * C2S_FIFO is used to temporarily store message from CAN to SPI */
Custom_Element gC2S_FIFO[C2S_FIFO_SIZE];
Custom_FIFO gCan2Spi_FIFO = {0, 0, 0, gC2S_FIFO};

/* Variables for S2C_FIFO
 * S2C_FIFO is used to temporarily store message from SPI to CAN */
Custom_Element gS2C_FIFO[S2C_FIFO_SIZE];
Custom_FIFO gSpi2Can_FIFO = {0, 0, 0, gS2C_FIFO};
```

3.5 SPI 接收和传输（透明传输）

通常，SPI 主机控制 SPI 通信，SPI 从机无法触发从机到主机的通信。在本设计中，使用的是另一个 IO。从机的 IO 下拉会通知主机存在需要发送的信息。用户可以根据需要，修改引脚或删除 IO 功能。

对于 SPI 接收，在 *bridge_spi.c* 中定义了三个全局变量。

```
uint8_t gSpiReceiveGroup[SPI_RX_SIZE];
Custom_Element gSPI_RX_Element;
uint16_t gGetSpiRxMsg_Count;
```

下面是 SPI 主机接收数据的过程。IO 中断用于检测 IO 下拉。

1. 在 IO 中断中，调用 *readSpiRxMsg()* 来发送 (SPI_TRANSPARENT_LENGTH) 字节，以接收来自 SPI 从机的消息（SPI 发送和接收一起进行）。
2. 调用 *getSpiRxMsg_transparent()* 将消息存储到 *gSpiReceiveGroup* 中。当接收到 (SPI_TRANSPARENT_LENGTH) 个字节时，消息接收完成。
3. 调用 *processSpiRxMsg_transparent()* 从 *gSpiReceiveGroup* 中提取数据并将数据存储在 *gSPI_RX_Element* 中。
4. 将 *gSPI_RX_Element* 放入 *gSpi2Can_FIFO*。

下面是 SPI 从机接收数据的过程

1. 调用 *getSpiRxMsg_transparent()* 将消息存储到 *gSpiReceiveGroup* 中。发生超时即表示消息接收完成。
2. 调用 *processSpiRxMsg_transparent()* 从 *gSpiReceiveGroup* 中提取数据并将数据存储在 *gSPI_RX_Element* 中。
3. 将 *gSPI_RX_Element* 放入 *gSpi2Can_FIFO*。

对于 SPI 传输，在 bridge_spi.c 中定义了两个全局变量。

```
uint8_t gSpiTransmitGroup[SPI_TX_SIZE];
Custom_Element gSPI_TX_Element;
```

下面是 SPI 主机和从机传输的过程。

1. 从 *gCan2Spi_FIFO* 获取 *gSPI_TX_Element*。
2. 调用 *processSpiTxMsg_transparent()* 从 *gSPI_TX_Element* 获取数据并将数据存储到 *gSpiTransmitGroup* 中。
3. 调用 *sendSpiTxMsg()* 以通过 SPI 传输 *gSpiTransmitGroup*。对于 SPI 从机，仅发送 *(SPI_TRANSPARENT_LENGTH)* 个字节。
4. （仅限 SPI 从机）使用 IO 触发主机，向从机读取数据。

3.6 SPI 接收和传输（协议传输）

通常，SPI 主机用于控制 SPI 通信，SPI 从机不能触发从机到主机的通信。在本设计中，使用的是另一个 IO。从机的 IO 下拉会通知主机存在需要发送的信息。用户可以根据需要，修改引脚或删除 IO 功能。

对于 SPI 接收，在 *bridge_spi.c* 中定义了两个全局变量。

```
uint8_t gSpiReceiveGroup[SPI_RX_SIZE];
Custom_Element gSPI_RX_Element;
```

下面是 SPI 主机接收的过程。IO 中断用于检测 IO 下拉。

1. 在 IO 中断中，调用 *readSpiRxMsg()* 来发送字节，以接收来自 SPI 从机的消息（SPI 发送和接收一起进行）。
2. 调用 *getSpiRxMsg()* 来检测标头并将完整的消息存储到 *gSpiReceiveGroup* 中。
3. 调用 *processSpiRxMsg()* 从 *gSpiReceiveGroup* 中提取信息并将数据存储在 *gSPI_RX_Element* 中。
4. 将 *gSPI_RX_Element* 放入 *gSpi2Can_FIFO* 中。

下面是 SPI 从机接收的过程。

1. 调用 *getSpiRxMsg()* 将消息存储到 *gSpiReceiveGroup* 中。发生超时即表示消息接收完成。
2. 调用 *processSpiRxMsg()* 从 *gSpiReceiveGroup* 中提取数据并将数据存储在 *gSPI_RX_Element* 中。
3. 将 *gSPI_RX_Element* 放入 *gSpi2Can_FIFO*。

对于 SPI 传输，在 *bridge_spi.c* 中定义了两个全局变量。

```
uint8_t gSpiTransmitGroup[SPI_TX_SIZE];
Custom_Element gSPI_TX_Element;
```

下面是 SPI 主机和从机传输的过程

1. 从 *gCan2Spi_FIFO* 获取 *gSPI_TX_Element*。
2. 调用 *processSpiTxMsg()* 从 *gSPI_TX_Element* 获取信息并将数据存储在 *gSpiTransmitGroup* 中。
3. 调用 sendSpiTxMsg() 以通过 SPI 传输 *gSpiTransmitGroup*。
4. （仅限 SPI 从机）使用 IO 触发主机，向从机读取数据。

3.7 CAN 接收和传输

对于 CAN 接收，在 *bridge_can.c* 中定义了两个全局变量。

```
DL_MCAN_RxBufElement rxMsg;
Custom_Element gCAN_RX_Element;
```

下面是 CAN 接收的过程。

1. 调用 *getCANRxMsg()* 以获取从 *CAN message RAM* 到 *rxMsg* 的完整消息。
2. 调用 *processCANRxMsg()* 从 *rxMsg* 中提取信息并将其存储到 *gCAN_RX_Element* 中。
3. 将 *gCAN_RX_Element* 放入 *gCan2Spi_FIFO* 中。

对于 CAN 传输，在 bridge_can.c 中定义了两个全局变量。

```
DL_MCAN_TxBufElement txMsg0;
Custom_Element gCAN_TX_Element;
```

下面是 CAN 传输的过程。

1. 从 *gSpi2Can_FIFO* 获取 *gCAN_TX_Element*。
2. 调用 *processCANTxMsg()* 从 *gCAN_TX_Element* 接收信息并将其存储到 *txMsg0* 中。
3. 调用 *sendCANTxMsg()* 通过 CAN 传输 *txMsg0*。

3.8 应用集成

表 3-1 中的函数被分类到不同的文件中。SPI 接收和传输函数包含在 *bridge_spi.c* 和 *bridge_spi.h* 中。CAN 接收和传输函数包含在 *bridge_can.c* 和 *bridge_can.h* 中。FIFO 元素结构在 *user_define.h* 中定义。

用户可以通过文件分离函数。例如，如果只需要 SPI 函数，用户可以保留 *bridge_spi.c* 和 *bridge_spi.h* 以调用相应函数。

为了进行外设的基本配置，该工程集成了一个 SysConfig 配置文件。用户可以使用 SysConfig 轻松修改外设的基本配置。

需要此功能的应用必须包含 CAN 模块和 API 以及 SPI 模块 API。所有 API 文件都包含在 SDK 下载中。

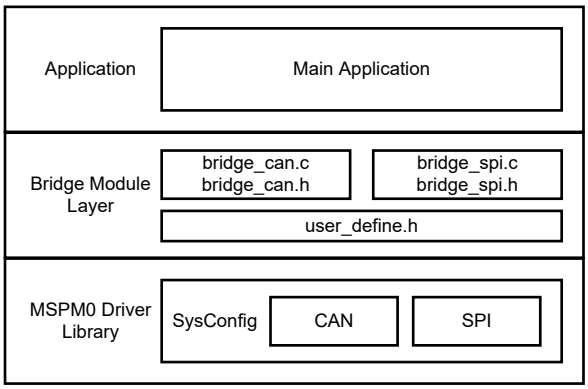

图 3-1. 软件所需的文件

表 3-4 列出了 CAN-SPI 桥接设计在闪存大小和 RAM 大小方面的占用空间。表 3-4 中的数据是使用 Code Composer Studio（版本：12.7.1.00001）且优化级别为 2 的条件下确定的。

用户可以调整 FIFO 的大小。FIFO 越大意味着缓存容量越大，但需要的 RAM 空间越大。有关详细信息，请参见应用程序方面 中的相关内容。用户可以根据实际数据长度，配置数据字段大小。如 表 3-4 中所示，使用字节数较少的数据字段可以显著减少 RAM 的使用。

表 3-4. CAN-SPI 桥接器的内存占用空间

所需的最小代码大小（字节）	闪存	SRAM
CAN-SPI 主桥接器 （协议传输 S2C_FIFO_SIZE=8 C2S_FIFO_SIZE=8 数据大小 = 12 字节）	6128	1466
CAN-SPI 从桥接器 （协议传输 S2C_FIFO_SIZE=8 C2S_FIFO_SIZE=8 数据大小 = 12 字节）	6344	1462
CAN-SPI 主桥接器 （协议传输 S2C_FIFO_SIZE=8 C2S_FIFO_SIZE=8 数据大小 = 64 字节）	6224	2610
CAN-SPI 从桥接器 （协议传输 S2C_FIFO_SIZE=8 C2S_FIFO_SIZE=8 数据大小 = 64 字节）	6440	2606
CAN-SPI 主桥接器 （协议传输 S2C_FIFO_SIZE=30 C2S_FIFO_SIZE=30 数据大小 = 12 字节）	6232	2522
CAN-SPI 从桥接器 （协议传输 S2C_FIFO_SIZE=30 C2S_FIFO_SIZE=30 数据大小 = 12 字节）	6448	2518

4 硬件

通过使用 CAN 分析仪，用户可以在 CAN 侧发送和接收消息。作为演示，可以将两个 LaunchPad™ 用作两个 CAN-SPI 桥接器（一个 SPI 主机和一个 SPI 从机）以形成一个环路。当 CAN 分析仪通过主机 LaunchPad 发送 CAN 消息时，它可以从从机 LaunchPad 接收 CAN 消息。图 4-1 展示了基本结构。请注意，构建 CAN 总线需要 CAN 收发器。图 4-2 显示了在演示中 CAN 分析仪发送和接收的消息。

随附的演示使用两个 LaunchPad、一个 TCAN1046EVM 和一个 CAN 分析仪。TCAN1046EVM 是一款高速双通道 CAN 收发器评估模块。图 4-3 显示了演示的连接。PA12 LaunchPad 用于 CAN 传输，PA13 LaunchPad 用于 CAN 接收。PA12 和 PA13 必须连接到 TCAN1046EVM 的 TX 引脚和 RX 引脚。PB9 用于 SPI SCLK（时钟）。PB8 用于 SPI MOSI（主机输出、从机输入）。PB7 用于 SPI MISO（主机输入、从机输出）。PB6 用于 SPI CS（芯片选择）。PB20 用于由 I2C 从机到主机的 IO 触发。

对于 TCAN1046EVM（因为 TCAN1046 支持电平转换），VCC 必须连接到 5V，VIO 必须连接到 3.3V。CAN 总线上的端接（CANH 和 CANL）必须使用 J2（或 J3）和 J6（或 J8）跳线进行配置。每个跳线都将 120Ω 终端添加到相应的总线。有关更多信息，请参阅相关文档。

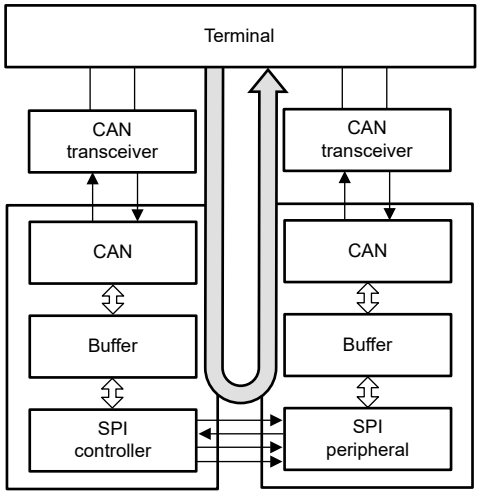

图 4-1. 随附演示的基本结构

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALL		
0	0.000000	Device0	0	0x1	StandardFrame	CANFD Accelerate	Tx	1	00
1	0.000300	Device0	1	0x1	StandardFrame	CANFD Accelerate	Rx	1	00
2	18.323700	Device0	0	0x2	StandardFrame	CANFD Accelerate	Tx	2	00 11
3	18.324100	Device0	1	0x2	StandardFrame	CANFD Accelerate	Rx	2	00 11
4	33.411500	Device0	0	0x3	StandardFrame	CANFD Accelerate	Tx	4	00 11 22 33
5	33.411900	Device0	1	0x3	StandardFrame	CANFD Accelerate	Rx	4	00 11 22 33
6	50.216400	Device0	0	0x4	StandardFrame	CANFD Accelerate	Tx	8	00 11 22 33 44 55 66 77
7	50.216900	Device0	1	0x4	StandardFrame	CANFD Accelerate	Rx	8	00 11 22 33 44 55 66 77
8	67.378700	Device0	0	0x5	StandardFrame	CANFD Accelerate	Tx	12	00 11 22 33 44 55 66 77 88 99 AA BB
9	67.379400	Device0	1	0x5	StandardFrame	CANFD Accelerate	Rx	12	00 11 22 33 44 55 66 77 88 99 AA BB
10	344.182200	Device0	0	0x6	StandardFrame	CANFD Accelerate	Tx	32	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF...
11	344.183400	Device0	1	0x6	StandardFrame	CANFD Accelerate	Rx	32	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF...

图 4-2. CAN 分析仪针对演示发送和接收的消息

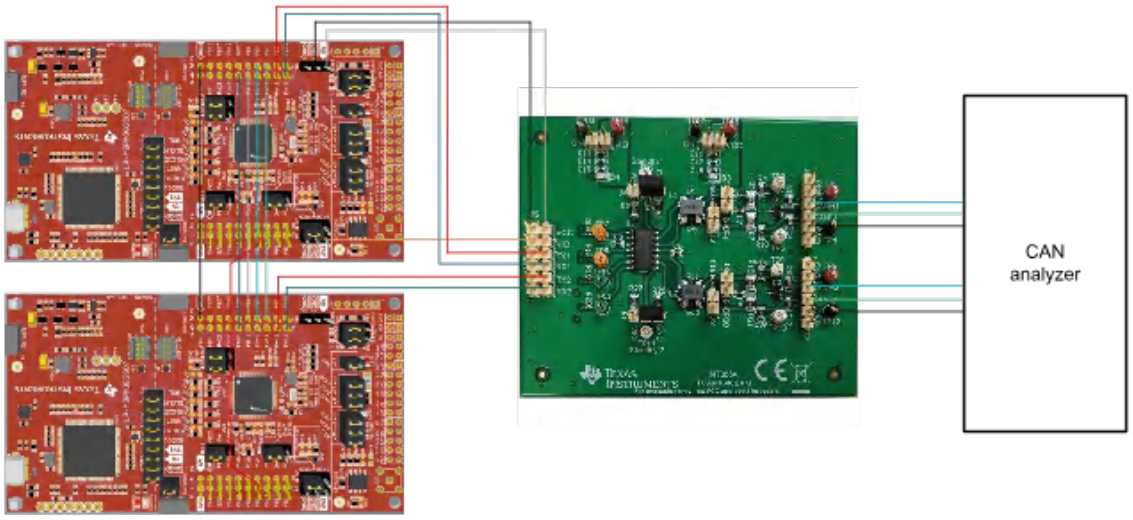

图 4-3. 演示的硬件连接

5 应用程序方面

本节介绍了 CAN-SPI 桥接设计在应用方面的特性，以及如何对其进行配置以满足应用需求。

5.1 结构灵活

表 3-3 中列出了各种可配置的参数。用户可以通过修改 *user_define.h* 中定义的所有参数来配置 CAN 或 SPI 数据包帧、FIFO 的大小或数据区域的最大规模。

用户还可以在 *user_define.h* 中修改 *Custom_Element* 的定义。可根据应用程序和存储要求，增加或减少条目。

```
/*user-defined information storage structure */
typedef struct {
    /*! Origin Identifier, indicating the origin of the message */
    uint32_t origin_id;
    /*! Destination Identifier, indicating the destination of the message */
    uint32_t destination_id;
    /*! Data Length Code */
    uint8_t dlc;
    /*! Data bytes */
    uint8_t data[TRANSMIT_DATA_LENGTH];
} Custom_Element;
```

两个通信接口的接收和发送是分开的。报文将通过 FIFO 传送。因此，用户可以对结构进行更改。例如，使消息遵循特定的格式，甚至遵循特定的通信协议。此外，可以根据 图 2-3，将结构拆分为单向传输。

5.2 可选的 SPI 配置

SPI 模块充当主机或从机接口，用于与外设和其他控制器进行同步串行通信。该设计提供了用于 CAN-SPI 桥接器（SPI 主机）的代码和用于 CAN-SPI 桥接器（SPI 从机）的代码。

用户可以配置 I2C 模块的各种功能。通过使用 SysConfig，用户可以更改 I2C 的基本配置。有关更多配置，请参阅相关文档。

5.3 可选的 CAN 配置

MSPM0 的 CAN 模块符合 CAN 协议 2.0 A、B 和 ISO 11898-1:2015 标准。用户可以配置 CAN 模块的各种功能。通过使用 SysConfig，用户可以更改 CAN 的基本配置。（例如，数据传输速率）。

该代码附带了 CAN ID 的可选配置。示例代码默认为 11 位 ID（标准 ID）。可以通过修改 *user_define.h* 来更改配置。

- 添加“#define CAN_ID_EXTEND”可启用 29 位 ID（扩展 ID）。

此外，该示例代码支持在单个帧中携带 64 字节的数据。用户可以根据应用要求来配置适当的数据大小，进一步减少 FIFO 占用的 RAM 空间。

5.4 CAN 总线多节点通信示例

CAN 通信是总线通信。用户可以使用此 CAN-SPI 桥接器设计来测试 CAN 总线的多节点通信。图 5-1 显示了基本结构。当用户通过任何 CAN-SPI 桥接器向 CAN 总线发送消息时，系统会立即从其他节点读回该消息。

必须使用至少三个 LaunchPad。LaunchPad 上的每个 CAN 通信都需要一个收发器。请参阅 图 4-3 以查看 LaunchPad 和收发器之间的连接。

MSPM0 的 CAN 模块支持硬件过滤，以选择具有特定 ID 的消息。请注意，在此示例代码中，默认情况下不执行硬件过滤。用户可以配置硬件过滤。有关具体配置，请参阅相关文档。

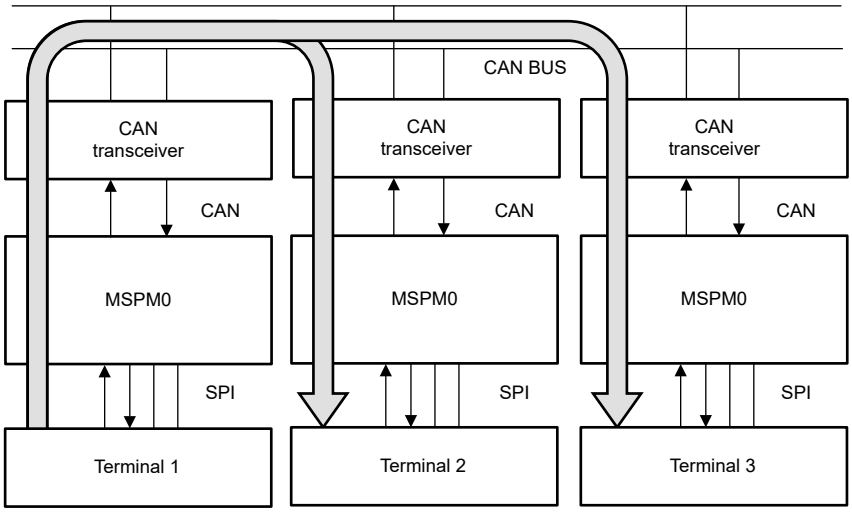

图 5-1. 多节点通信的基本结构

6 总结

本文介绍了 CAN 转 SPI 桥接器的实现方式，包括结构、函数定义、接口使用和应用方面。MSPM0 器件可用作 CAN 和 SPI 之间的转换器，允许桥接器在一个接口上发送和接收信息，并在另一个接口上接收和发送信息。

7 参考资料

- 德州仪器 (TI)，使用 MSPM0 MCU 在 CAN 和 UART 之间进行桥接设计，应用手册。
- 德州仪器 (TI)，使用 MSPM0 MCU 在 CAN 和 I2C 之间进行桥接设计，应用手册。
- 德州仪器 (TI)，CAN 转 UART 桥接器，子系统设计。
- 德州仪器 (TI)，CAN 转 SPI 桥接器，子系统设计。
- 德州仪器 (TI)，CAN 转 I2C 桥接器，子系统设计。
- 德州仪器 (TI)，MSPM0 SDK，工具。
- 德州仪器 (TI)，SysConfig 系统配置工具，网页。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265