Application Note

使用 MSPM0 MCU 在 CAN 和 I2C 之间进行桥接设计

摘要

本应用手册介绍了 CAN 转 I2C 桥接器的示例。本文档说明 CAN 转 I2C 桥接器的结构和行为。文中还介绍了软件实现、硬件连接和应用程序用法。用户可以通过修改预定义来配置桥接器。此外，还向用户提供了相关代码。

内容

插图清单

表格清单

商标

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

1 引言

根据应用不同，现在器件之间的通信方法有很多。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、I2C、CAN 等。当器件需要通过不同的通信接口传输数据时，我们可以构建一个桥接器。

对于 CAN 和 I2C，CAN-I2C 桥接器充当两个接口之间的转换器。CAN-I2C 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-I2C 桥接器时采用的软件和硬件设计。通过提供 CAN 和 I2C 通信接口，我们可以将 MSPM0G3507 微控制器 (MCU) 作为一种设计。随附的演示使用具有 2Mbps CANFD 和 400kHz 总线速度 I2C 的 MSPM0G3507 来演示如何在通道之间进行数据收发。

1.1 连接 CAN 和 I2C 的桥接器

CAN-I2C 桥接器连接 CAN 和 I2C 接口。该桥接器支持 I2C 在从机模式或主机模式下运行。本文档中的示例使用 CAN 分析仪来观察 CAN 数据。用户也可以通过 CAN-I2C 桥接器从 CAN 分析仪向 I2C 侧发送消息。对于 I2C 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。

本文档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 I2C 消息格式。用户可以根据应用程序的要求来更改格式。接收到来自 I2C 的消息时，消息格式为 < 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2 ...>。用户可以采用相同的格式通过 I2C 发送数据。55 AA 是标头。ID 区域为四个字节。长度区域为一个字节，表示数据长度。

对于透明传输，通过 I2C 停止中断来检测一条消息。来自 I2C 的数据被填充到 CAN 的数据区域（反之亦然）。CAN ID 是默认值。

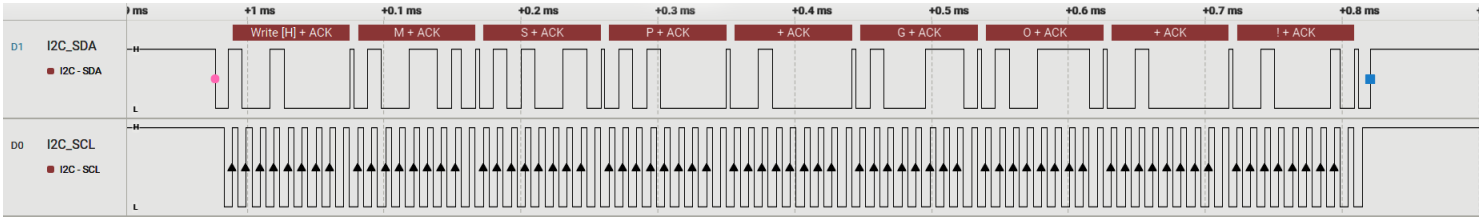

图 1-1. 用于 I2C 透明传输的逻辑分析仪

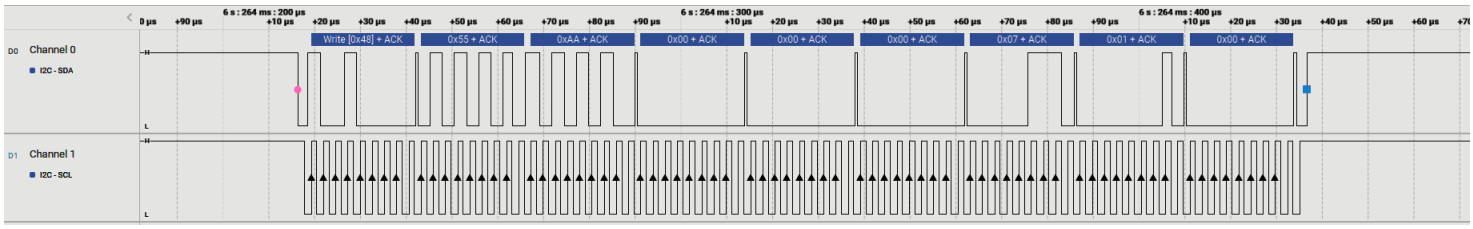

图 1-2. 用于 I2C 协议传输的逻辑分析仪

2 实施

2.1 原理

在本文档的设计中，CAN-I2C 桥接器同时使用 CAN 接收和传输功能以及 I2C 接收和传输功能。所以，必须配置 CAN 模块和 I2C 模块。由于不同通信的消息格式不同，CAN-I2C 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 I2C，可使用该接口作为从机或主机，在 MSPM0 器件和另一个 I2C 器件之间传输数据。如需更多信息，请参阅相关文档。由于 I2C 从机的接收和传输由 I2C 主机控制，因此 I2C 从机无法发起到 I2C 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 下拉会通知主机有需要发送的信息。

图 2-1 所示为 CAN-I2C 桥接器的基本原理。通常，CAN 的通信速率与 I2C 的通信速率不同。CAN FD 的波特率可以高达 5Mbps，而 I2C 以 400kHz 的总线速度运行，如示例代码所示。因此，一个接口接收到的数据可能未被另一个接口及时发送。为了匹配速率，该方案使用缓冲器在 CAN 和 I2C 之间传输数据。此缓冲器不仅实现数据缓存，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。

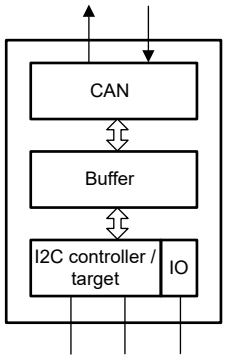

图 2-1. CAN-I2C 桥接器的基本原理

2.2 结构

具有协议传输和透明传输功能的 CAN-I2C 桥接器的结构如 图 2-3 和 图 2-4 所示。图 2-3 适用于 I2C 主机，图 2-4 适用于 I2C 从机。CAN-I2C 桥接器可以分为四个独立的任务：从 I2C 接收、从 CAN 接收、通过 CAN 传输、通过 I2C 传输。两个 FIFO 实现双向消息传输和消息缓存。

I2C 和 CAN 接收均设置为中断触发，以便及时接收消息。进入中断时，首先通过 *getXXXRxMsg()* 获取消息。

对于 CAN，CAN 帧是固定格式。MSPM0 支持传统 CAN 或 CANFD。CANFD 的帧如 图 2-2 所示。本文中的示例可以针对协议传输在数据区域中定义 0/1/4 个附加字节（I2C 地址的默认长度为一个字节），如 表 2-1 中所示。

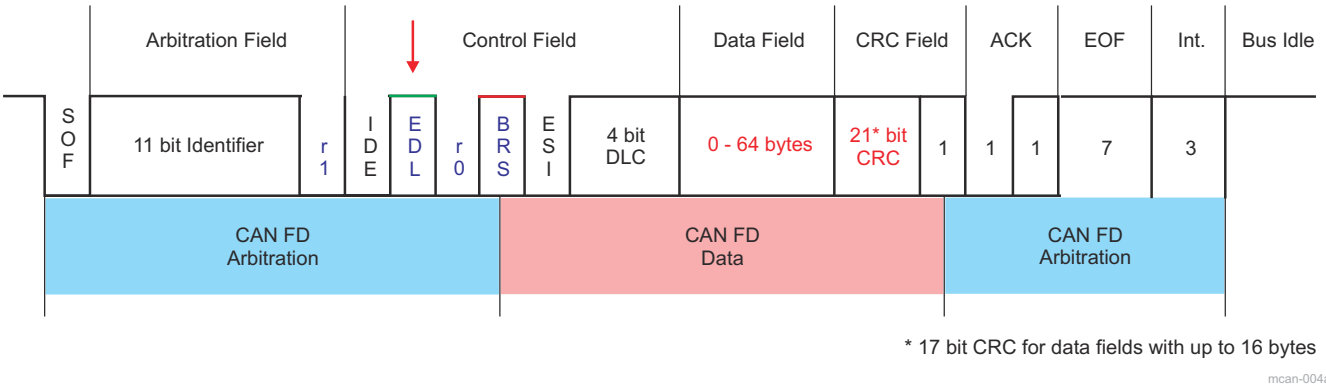

图 2-2. CAN FD 帧

表 2-1. CAN 数据包格式

	ID 区域	数据
协议传输	4/1/0 个字节	（数据长度）字节

对于 I2C 协议传输，基于串行帧信息来识别消息。表 2-2 中列出了 I2C 消息格式。

表 2-2. I2C 数据包格式

	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	—	—	—	主件到从件 — （数据长度）字节 从机到主机 — (I2C_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 区域默认占用四个字节以匹配 CAN ID，我们可将其配置为一个字节或 ID 区域不存在。数据长度区域占用一个字节。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，在 I2C 这种通信方法中，由 I2C 主机来控制消息传输和接收。通常，I2C 从件无法发起通信。对于 I2C 从机到主机通信，I2C 从机可以在需要发送消息时下拉 IO，如 图 2-4 中所示。当检测到 IO 为低电平时，I2C 主机可以在 IO 中断中启动 I2C 读取命令，如 图 2-3 中所示。

在 I2C 透明传输中，通过 I2C 停止中断来标识消息，如 图 2-4 中所示。所有字节均视为纯数据。加载数据包信息（例如 ID）的默认值。

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。图 2-5 显示了 FIFO 元素的格式。在 FIFO 元素的格式中，有 *origin_id*、*destination_id*、*data length* 和 *data*。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 I2C 传输均在主函数中执行。当检测到 FIFO 不为空时，将提取 FIFO 元素。消息经格式化后发出。对于 CAN，CAN 帧为固定格式，如 表 2-1 中所示。对于 I2C，消息以 表 2-2 中所列格式发送。

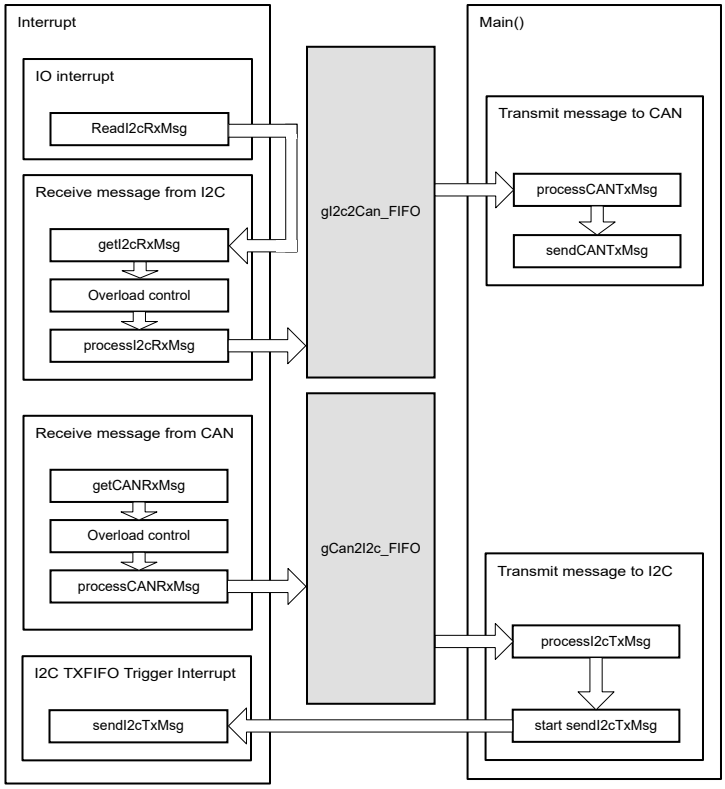

图 2-3. CAN-I2C（I2C 控制器）桥接器的结构：协议和透明

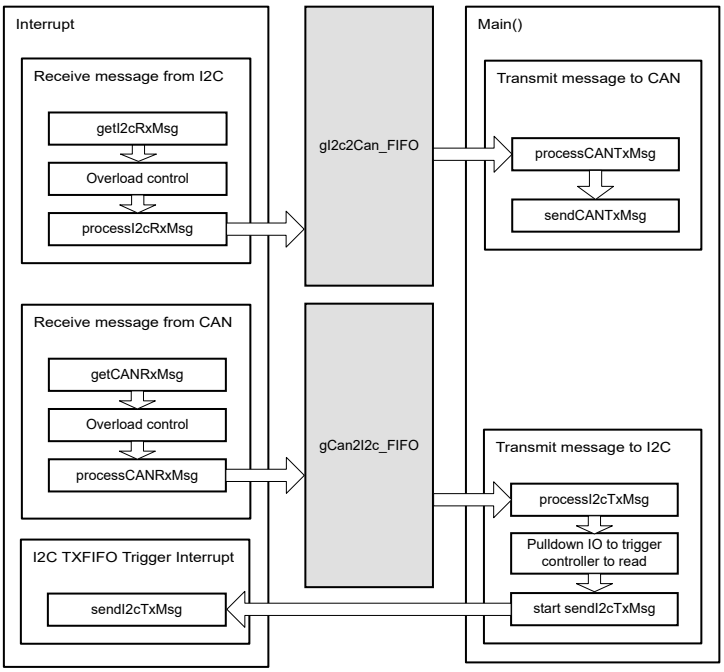

图 2-4. CAN-I2C（I2C 目标）桥接器的结构：协议和透明

图 2-5 展示了 FIFO 的结构。每个 FIFO 使用三个全局变量来指示 FIFO 状态。对于 *gI2c2Can_FIFO*，*gI2c2Can_FIFO.fifo_in* 表示写入位置，*gI2c2Can_FIFO.fifo_out* 表示读取位置，*gI2c2Can_FIFO.fifo_Count* 表示 *gI2c2Can_FIFO* 中的元素数量。

如果 *gI2c2Can_FIFO* 为空，则 *gI2c2Can_FIFO.fifo_in* 等于 *gI2c2Can_FIFO.fifo_out*，*gI2c2Can_FIFO.fifo_count* 为零。

执行 *processI2cRxMsg()* 时，来自 I2C 的新消息将存储到 *gI2c2Can_FIFO* 中。因此，*gI2c2Can_FIFO.fifo_in* 会移动到下一个位置，并且 *gI2c2Can_FIFO.fifo_count* 会递增 1。

从 *gI2c2Can_FIFO* 向 CAN 传输消息时，*gI2c2Can_FIFO.fifo_out* 会移动到下一个位置，而 *gI2c2Can_FIFO.fifo_count* 减 1。*gCan2I2c_FIFO* 与 *gI2c2Can_FIFO* 类似。

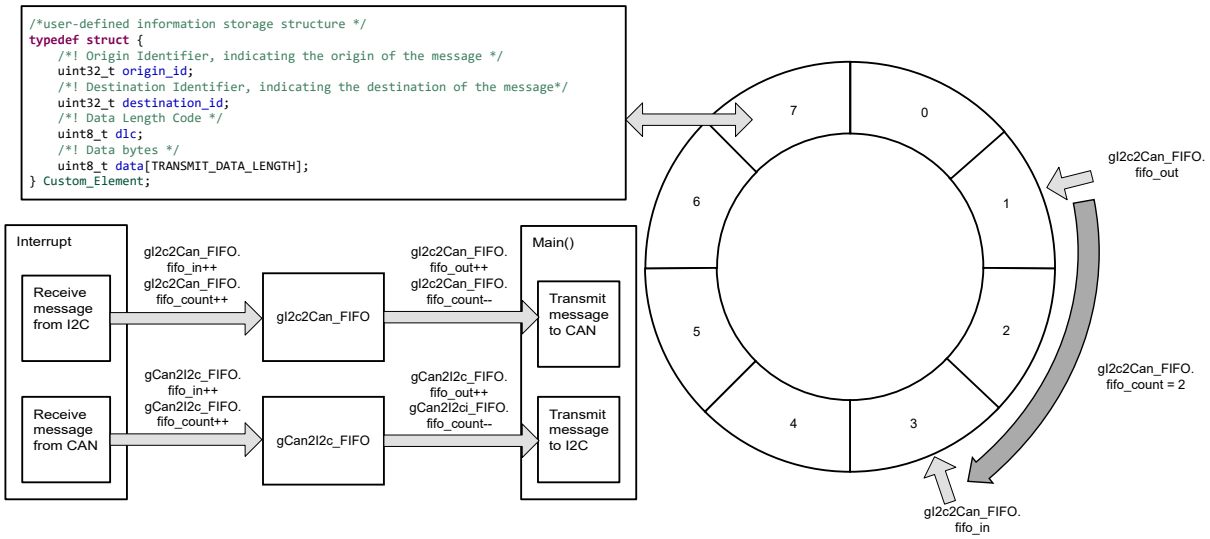

图 2-5. FIFO 的结构

3 软件说明

3.1 软件功能

这些函数是根据 图 2-3 和 图 2-4 设计。表 3-1 列出了这些函数。

表 3-1. 函数和说明

任务	函数	说明	位置
I2C 接收	readI2CRxMsg_controller( )	向从器件发送读取请求（仅限 I2C 主器件）	bridge_i2c.c bridge_i2c.h
	getI2CRxMsg_controller()	获取接收到的 I2C 消息（仅限 I2C 主机）（协议）	
	getI2CRxMsg_controller_ transparent()	获取接收到的 I2C 消息（仅限 I2C 主机）（透明）	
	getI2CRxMsg_target()	获取接收到的 I2C 消息（仅限 I2C 从机）（协议）	
	getI2CRxMsg_target_ transparent()	获取接收到的 I2C 消息（仅限 I2C 从机）（透明）	
	processI2CRxMsg_controller()	转换接收到的 I2C 消息格式（协议），并将消息存储到 gI2C_RX_Element 中（仅限 I2C 主机）	
	processI2CRxMsg_controller_ transparent()	转换接收到的 I2C 消息格式（透明），并将消息存储到 gI2C_RX_Element 中（仅限 I2C 主机）	
	processI2CRxMsg_target( )	转换接收到的 I2C 消息格式（协议），并将消息存储到 gI2C_RX_Element 中（仅限 I2C 从机）	
	processI2CRxMsg_target_ transparent()	转换接收到的 I2C 消息格式（透明），并将消息存储到 gI2C_RX_Element 中（仅限 I2C 从机）	
I2C 传输	processI2CTxMsg_controller()	转换要通过 I2C 发送的 gI2C_TX_Element 格式（协议）（仅限 I2C 主机）	
	processI2CTxMsg_controller_ transparent()	转换要通过 I2C 发送的 gI2C_TX_Element 格式（透明）（仅限 I2C 主机）	
	processI2CTxMsg_target()	转换要通过 I2C 发送的 gI2C_TX_Element 格式（协议）（仅限 I2C 从机）	
	processI2CTxMsg_target_ transparent()	转换要通过 I2C 发送的 gI2C_TX_Element 格式（透明）（仅限 I2C 从机）	
	sendI2CTxMsg_controller( )	通过 I2C 发送消息（仅限 I2C 主器件）	
	sendI2CTxMsg_target()	通过 I2C 发送消息（仅限 I2C 从器件）	
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 I2C，此示例中同时支持透明传输和协议传输，通过定义 I2C_TRANSPARENT 或 I2C_PROTOCOL 进行切换。

在透明传输中，用户可以为从 I2C 从机传输到 I2C 主机的消息配置默认数据长度 (I2C_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 1 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 I2C_TRANSPARENT
#ifdef I2C_TRANSPARENT
/* The format of I2C:
     * Transparent transmission - Data1 Data2 ...*/
/* data length for I2C master receiving or I2C slave transmitting*/
#define I2C_TRANSPARENT_LENGTH  (8)
#else
/* The format of I2C:
     * if I2C_ID_LENGTH = 4, format is 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2 ...
     * if I2C_ID_LENGTH = 1, format is 55 AA ID Length Data1 Data2 ...
     * if I2C_ID_LENGTH = 0, format is 55 AA Length Data1 Data2 ...*/
//#define I2C_ID_LENGTH  (0)
//#define I2C_ID_LENGTH  (1)
#define I2C_ID_LENGTH  (4)
#endif

/* default address for I2C master receiving */
#define I2C_TARGET_ADDRESS (0x48)
```

表 3-2. 在不同 I2C 模式下接收或发送消息的字节数

参数	I2C 接口：主机		I2C 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+I2C_ID_LENGT+1+Length) 字节	(2+I2C_ID_LENGT+1+Length) 字节	(2+I2C_ID_LENGT+1+Length) 字节	(2+I2C_ID_LENGT+1+Length) 字节
透明传输	(I2C_TRANSPARENT_LENGTH) 字节	(Length) 字节	I2C 停止中断标识消息结束	(I2C_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 1) 来在数据区域中添加另一个 ID。在本示例中，我们为 I2C 地址添加了一个单字节 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 I2C_TRANSPARENT	定义/未定义	启用 I2C 透明传输。
#define I2C_PROTOCOL	定义/未定义	启用 I2C 协议传输。
#defineI2C_TRANSPARENT_LENGTH (8)		从 I2C 从机传输到 I2C 主机之消息的默认数据长度。仅在定义了 I2C_TRANSPARENT 时可用。在本例中，默认值为八个字节。
#define I2C_TARGET_ADDRESS (0x48)		从 I2C 从机传输到 I2C 主机之消息的默认 I2C 从机地址。在本例中，默认值为 0x48
#define I2C_ID_LENGTH (4)	0/1/4	可选 I2C ID 长度，与协议中的 ID 区域相关。仅在定义了 I2C_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，默认值为一个字节
#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 C2I_FIFO_SIZE (8)		CAN 到 I2C FIFO 的大小。请注意 SRAM 的使用量。
#define ItoC_FIFO_SIZE (8)		I2C 到 CAN FIFO 的大小。请注意 SRAM 的使用量。
#define DEFAULT_I2C_ORIGIN_ID (0x00)		I2C 源 ID 的默认值
#define DEFAULT_I2C_DESTINATION_ID (0x00)		I2C 目标 ID 的默认值
#define DEFAULT_CAN_ORIGIN_ID (0x00)		CAN 源 ID 的默认值
#define DEFAULT_CAN_DESTINATION_ID (0x48)		CAN 目标 ID 的默认值

3.3 自定义元件的结构

Custom_Element 结构在 *user_define.h* 中定义，图 2-5 所示就是此结构。

*源标识符*用于指示消息的来源。以下是示例 (CAN_ID_LENGTH =1, I2C_ID_LENGTH =4)。

- 示例 1 - CAN 接口接收和传输
  1. 当 CAN-I2C 桥接器接收到 CAN 消息时，来自 CAN 帧的 ID 是*源标识符*，指示消息来自何处。
  2. 当 CAN-I2C 桥接器传输 CAN 消息时，*源标识符*是 CAN 中的 1 字节 ID（CAN_ID_LENGTH 默认设置为 1），指示消息来自何处。

- 示例 2 - I2C 接口接收和传输（I2C 协议传输）
  1. 当 CAN-I2C 桥接器接收到 I2C 消息（I2C 协议传输）时，如果 I2C 用作主机，则 I2C_TARGET_ADDRESS 是*源标识符*，指示消息来自何处。如果 I2C 用作从机，而 I2C 主机没有地址，则 DEFAULT_I2C_ORIGIN_ID 为*源标识符*。
  2. 当 CAN-I2C 桥接器传输 I2C 消息（I2C 协议传输）时，*源标识符*是 I2C 数据中的 4 字节 ID（I2C _ID_LENGTH 默认设置为 4），指示消息来自何处。

- 示例 3 - I2C 接口接收和传输（I2C 透明传输）
  1. 当 CAN-I2C 桥接器接收到 I2C 消息（I2C 透明传输）时，如果 I2C 用作主机，则 I2C_TARGET_ADDRESS 是*源标识符*，指示消息来自何处。如果 I2C 用作从机，并且 I2C 主机没有地址，则 DEFAULT_I2C_ORIGIN_ID 为*源标识符*。
  2. 当 CAN-I2C 桥接器传输 I2C 消息（I2C 透明传输）时，*源标识符*将被忽略（透明传输没有 ID 区域）。

*目标标识符*指示消息的目标地址。以下是示例 (CAN_ID_LENGTH =1, I2C_ID_LENGTH =4)。

- 示例 1 - CAN 接口接收和传输
  1. 当 CAN-I2C 桥接器接收到 CAN 消息时，来自 CAN 数据区域的 1 字节 ID（CAN_ID_LENGTH 默认设置为 1）是*目标标识符*，指示消息的目标地址（I2C 地址）。
  2. 当 CAN-I2C 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 位和 29 位均受支持。

- 示例 2 - I2C 接口接收和传输（I2C 协议传输）
  1. 当 CAN-I2C 桥接器接收到 I2C 消息（I2C 协议传输）时，来自 I2C 数据的 4 字节 ID 是*目标标识符*（I2C_ID_LENGTH 默认设置为 4）。CAN 传输需要 ID 信息。
  2. 当 CAN-I2C 桥接器传输 I2C 消息（I2C 协议传输）时，如果 I2C 用作主机，则*目标标识符*是 I2C 地址。如果 I2C 用作从机，则*目标标识符*将被忽略。（使用 IO 触发主机来生成消息。）

- 示例 3 - I2C 接口接收和传输（I2C 透明传输）
  1. 当 CAN-I2C 桥接器接收到 I2C 消息（I2C 透明传输）时，DEFAULT_I2C_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-I2C 桥接器传输 I2C 消息（I2C 透明传输）时，如果 I2C 用作主机，则*目标标识符*是 I2C 地址。如果 I2C 用作从机，则*目标标识符*将被忽略（使用 IO 触发主机来传输消息）。

```
/*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 是在 *user_define.h* 中定义的结构。图 2-5 中也显示了 CUSTOM_FIFO。

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

gCan2I2c_FIFO 和 *gI2c2Can_FIFO* 在 *main.c* 中定义。请注意 SRAM 的使用量，它与 *C2I_FIFO_SIZE*、*ItoC_FIFO_SIZE* 以及 *Custom_Element* 的大小有关。

```
/* Variables for ItoC_FIFO
 * ItoC_FIFO is used to temporarily store message from I2C to CAN */
Custom_Element gItoC_FIFO[ItoC_FIFO_SIZE];
Custom_FIFO gI2c2Can_FIFO = {0, 0, 0, gItoC_FIFO};

/* Variables for C2I_FIFO
 * C2I_FIFO is used to temporarily store message from CAN to I2C */
Custom_Element gC2I_FIFO[C2I_FIFO_SIZE];
Custom_FIFO gCan2I2c_FIFO = {0, 0, 0, gC2I_FIFO};
```

3.5 I2C 接收和传输（透明传输）

通常，I2C 主机控制 I2C 通信，I2C 从机无法触发从机到主机的通信。在本设计中，使用了另一个 IO。从机的 IO 下拉会通知主机有需要发送的信息。用户可以根据需要，修改引脚或删除 IO 功能。

对于 I2C 接收，*bridge_i2c.c* 中定义了三个全局变量。

```
uint8_t gI2CReceiveGroup[I2C_RX_SIZE];
Custom_Element gI2C_RX_Element;
uint16_t gGetI2cRxMsg_Count;
```

下面是 I2C 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 *readI2CRxMsg_controller()* 向 I2C 从机发送 I2C_TRANSPARENT_LENGTH) 字节的读取请求。
2. 调用 *getI2CRxMsg_controller_transparent ()*，将消息存储到 *gI2cReceiveGroup* 中。当接收到 (I2C_TRANSPARENT_LENGTH) 个字节时，、消息接收完成。
3. 调用 *processI2CRxMsg_controller_transparent ()*，从 *gI2cReceiveGroup* 中提取数据，并将消息存储到 *gI2C_RX_Element* 中。
4. 将 *gI2C_RX_Element* 放入 *gI2c2Can_FIFO* 中。

下面是 I2C 从机接收的过程。

1. 调用 getI2CRxMsg_target_transparent () 将消息存储到 *gI2cReceiveGroup* 中。当 I2C 停止中断发生（I2C 停止条件）时，消息接收完成。
2. 调用 processI2CRxMsg_target_transparent ()，从 *gI2cReceiveGroup* 中提取数据，并将数据存储在 *gI2C_RX_Element* 中。
3. 将 *gI2C_RX_Element* 放入 *gI2c2Can_FIFO* 中。

对于 I2C 传输，*bridge_i2c.c* 中定义了四个全局变量。

```
uint8_t gI2cTransmitGroup[I2C_TX_SIZE];
Custom_Element gI2C_TX_Element;
uint32_t gTxLen, gTxCount;
```

下面是 I2C 主机/从机传输的过程。

1. 从 *gCan2I2c_FIFO* 获取 *gI2C_TX_Element*。
2. 调用 *processI2CTxMsg_controller_transparent()* 和 *processI2CTxMsg_target_ transparent()*，从 *gI2C_TX_Element* 接收数据，并将消息存储到 *gI2cTransmitGroup* 中。
3. 调用 *sendI2CTxMsg_controller()* 和 *sendI2CTxMsg_target()*，通过 I2C 传输 *gI2cTransmitGroup*。对于 I2C 从机，使用 IO 触发主机以读取从机中的数据，并且仅发送 (I2C_TRANSPARENT_LENGTH) 个字节。

主机模式和从机模式的函数都包含在 *bridge_i2c.c* 中。

3.6 I2C 接收和传输（协议传输）

通常，I2C 主机控制 I2C 通信，I2C 从机无法触发从机到主机的通信。在本设计中，使用了另一个 IO。从机的 IO 下拉会通知主机要发送信息。用户可以根据需要修改引脚或删除 IO 功能。

对于 I2C 接收，*bridge_i2c.c* 中定义了两个全局变量。

```
uint8_t gI2CReceiveGroup[I2C_RX_SIZE];
Custom_Element gI2C_RX_Element;
```

下面是 I2C 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 *readI2CRxMsg_controller()* 向 I2C 从机发送读取请求。
2. 调用 *getI2CRxMsg_controller()* 以检测标头，从而将完整消息存储在 *gI2cReceiveGroup* 中。
3. 调用 processI2CRxMsg_controller()，从 *gI2cReceiveGroup* 中提取数据，并将数据存储在 *gI2C_RX_Element* 中
4. 将 *gI2C_RX_Element* 放入 *gI2c2Can_FIFO* 中。

下面是 I2C 从机接收的过程。

1. 调用 *getI2CRxMsg_target()*，将消息存储到 *gI2cReceiveGroup* 中。
2. 调用 processI2CRxMsg_target()，从 *gI2cReceiveGroup* 中提取数据，并将数据存储到 *gI2C_RX_Element* 中。
3. 将 *gI2C_RX_Element* 放入 *gI2c2Can_FIFO* 中。

对于 I2C 传输，*bridge_i2c.c* 中定义了四个全局变量。

```
uint8_t gI2cTransmitGroup[I2C_TX_SIZE];
Custom_Element gI2C_TX_Element;
uint32_t gTxLen, gTxCount;
```

下面是 I2C 主机和从机传输的过程。

1. 从 *gCan2I2c_FIFO* 获取 *gI2C_TX_Element*。
2. 调用 processI2CTxMsg_controller()/processI2CTxMsg_target()，从 *gI2C_TX_Element* 获取数据、并将消息存储到 *gI2cTransmitGroup* 中。
3. 调用 *sendI2CTxMsg_controller()* 和 *sendI2CTxMsg_target()*，通过 I2C 传输 *gI2cTransmitGroup*。对于 I2C 从机，使用 IO 触发主机以读取从机中的数据。

主机模式和从机模式的函数都包含在 *bridge_i2c.c* 中。

3.7 CAN 接收和传输

对于 CAN 接收，在 *bridge_can.c* 中定义了 2 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 *gCan2I2c_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 *gI2c2Can_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 ()* 以通过 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。I2C 接收和传输函数包含在 *bridge_i2c.c* 和 *bridge_i2c.h* 中。CAN 接收和传输函数包含在 *bridge_can.c* 和 *bridge_can.h* 中。FIFO 元素的结构在 *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 I2C 函数，用户可以保留 *bridge_i2c.c* 和 *bridge_i2c.h* 以调用相应函数。

为了进行外设的基本配置，该项目集成了 SysConfig 配置文件。用户可以使用 SysConfig 修改外设的基本配置。

需要此功能的应用必须包含 CAN 模块 API 和 I2C 模块 API。所有 API 文件都包含在 SDK 中。

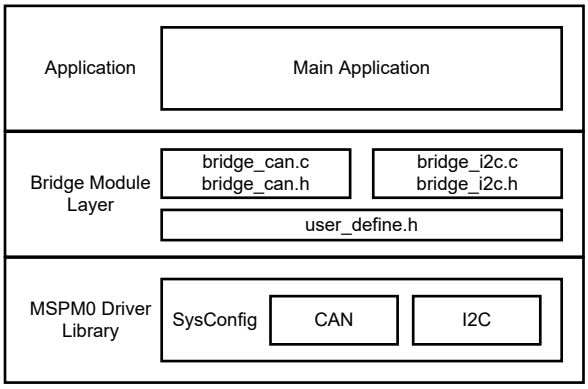

图 3-1. 软件所需的文件

表 3-4 详细说明了 CAN-I2C 桥接器解决方案在闪存大小和 RAM 大小方面的占用空间。下表和下图是在使用 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 越大意味着高速缓存容量越大，但需要的 RAM 空间也越大。有关详细信息，请参阅“应用程序方面”中的相关内容。用户可以根据实际数据长度来配置数据字段大小。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-I2C 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-I2C 主桥接器 （协议传输 ItoC_FIFO_SIZE=8 C2S_FIFO_SIZE=8 数据大小 = 12 字节）	6352	1428
CAN-I2C 从桥接器 （协议传输 ItoC_FIFO_SIZE=8 C2I_FIFO_SIZE=8 数据大小 = 12 字节）	6264	1428
CAN-I2C 主桥接器 （协议传输 ItoC_FIFO_SIZE=8 C2I_FIFO_SIZE=8 数据大小 = 64 字节）	6440	2572
CAN-I2C 从桥接器 （协议传输 ItoC_FIFO_SIZE=8 C2I_FIFO_SIZE=8 数据大小 = 64 字节）	6360	2572
CAN-I2C 主桥接器 （协议传输 ItoC_FIFO_SIZE=30 C2I_FIFO_SIZE=30 数据大小 = 12 字节）	6456	2484
CAN-I2C 从桥接器 （协议传输 ItoC_FIFO_SIZE=30 C2I_FIFO_SIZE=30 数据大小 = 12 字节）	6368	2484

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-I2C 桥接器（一个 I2C 主机和一个 I2C 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，也可以自从机 LaunchPad 接收 CAN 消息。图 4-1 所示为基本结构。请注意，构建 CAN 总线需要 CAN 收发器。图 4-2 显示该演示中 CAN 分析仪发送和接收的消息。

随附演示使用两个 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。对于 LaunchPad，PA12 用于 CAN 传输，PA13 用于 CAN 接收。PA12 和 PA13 必须连接到 TCAN1046EVM 的 TX 引脚和 RX 引脚。PB2 用于 I2C SCL（串行时钟线）。PB3 用于 I2C SDA（串行数据线）。PB20 用于 I2C 从机到主机的 IO 触发。

由于 TCAN1046 支持电平转换，因此 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。

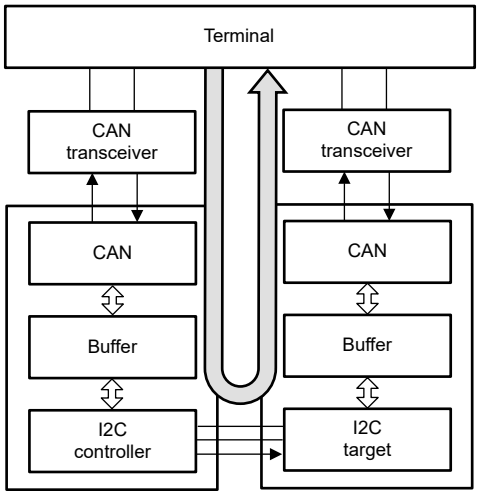

图 4-1. 随附演示的基本结构

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALL		
0	0.000000	Device0	0	0x1	StandardFrame	CANFD Accelerate	Tx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
1	0.000900	Device0	1	0x1	StandardFrame	CANFD Accelerate	Rx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
2	75.392500	Device0	1	0x2	StandardFrame	CANFD Accelerate	Tx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
3	75.393400	Device0	0	0x2	StandardFrame	CANFD Accelerate	Rx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
4	96.807600	Device0	1	0x3	StandardFrame	CANFD Accelerate	Tx	12	00 11 22 33 44 53 66 77 88 99 AA BB
5	96.808400	Device0	0	0x3	StandardFrame	CANFD Accelerate	Rx	12	00 11 22 33 44 53 66 77 88 99 AA BB
6	111.433500	Device0	0	0x4	StandardFrame	CANFD Accelerate	Tx	8	00 11 22 33 44 53 66 77
7	111.434100	Device0	1	0x4	StandardFrame	CANFD Accelerate	Rx	8	00 11 22 33 44 53 66 77
8	127.068700	Device0	1	0x5	StandardFrame	CANFD Accelerate	Tx	4	00 11 22 33
9	127.069200	Device0	0	0x5	StandardFrame	CANFD Accelerate	Rx	4	00 11 22 33
10	137.580700	Device0	0	0x6	StandardFrame	CANFD Accelerate	Tx	4	00 11 22 33
11	137.581200	Device0	1	0x6	StandardFrame	CANFD Accelerate	Rx	4	00 11 22 33
12	160.259200	Device0	0	0x7	StandardFrame	CANFD Accelerate	Tx	1	00
13	160.259700	Device0	1	0x7	StandardFrame	CANFD Accelerate	Rx	1	00

图 4-2. 演示中 CAN 分析仪发送和接收的消息 (CAN_ID_LENGTH = 0)

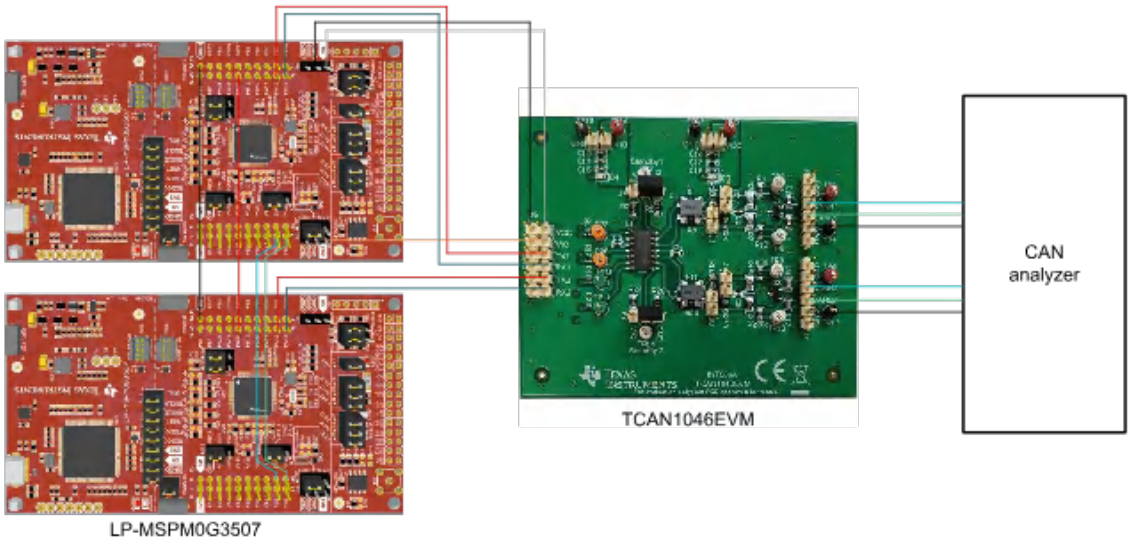

图 4-3. 演示的硬件连接

5 应用程序方面

本节介绍了 CAN-I2C 桥接器设计在应用程序方面的特性，以及如何对其进行配置以满足应用程序需求。

5.1 结构灵活

节 3.2 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 I2C 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传输单独运行。消息将通过 FIFO 传送。用户可以更改结构。例如，用户可以使消息遵循特定的格式，甚至遵循特定的通信协议。根据 图 2-3，该结构可以拆分为单向传输。

5.2 I2C 的可选配置

I2C 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。该设计提供一个适用于 CAN-I2C 桥接器（I2C 主机）的代码和另一个适用于 CAN-I2C（I2C 从机）的代码。

此外，用户还可以配置 I2C 模块的各种函数。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）。

提供的代码包含可选的 CAN ID 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-I2C 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-I2C 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。

至少需要使用三个 LaunchPad。LaunchPad 上的每个 CAN 通信都需要一个收发器。LaunchPad 和收发器之间的连接如 图 4-3 中所示。

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。

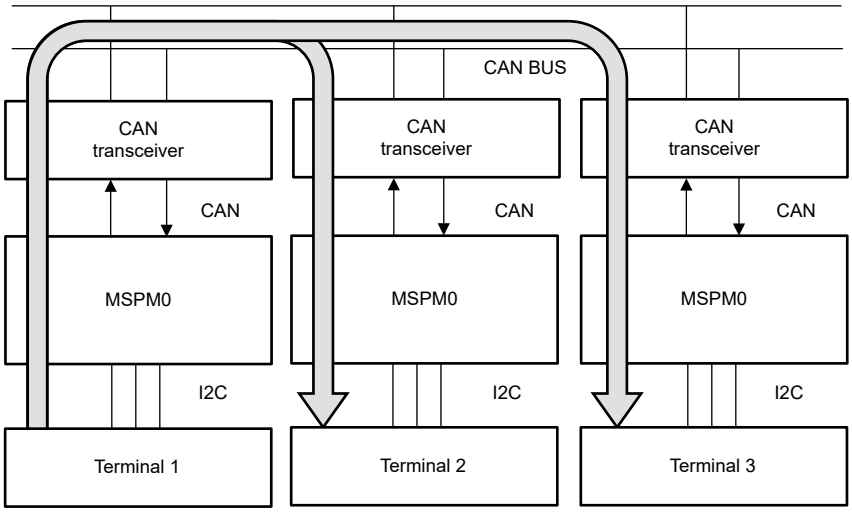

图 5-1. 多节点通信的基本结构

6 总结

本文介绍了 CAN 转 I2C 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 可以用作 CAN 和 I2C 之间的转换器，允许在一个接口上传输和接收信息，并在另一个接口上接收和发送信息。

7 参考资料

- 使用 MSPM0 MCU 在 CAN 和 UART 之间建立桥接的解决方案
- 使用 MSPM0 MCU 在 CAN 和 SPI 之间建立桥接的解决方案
- 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。
- 下载 MSPM0 SDK
- 德州仪器 (TI)，SysConfig 工具，配置工具。

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