Subsystem Design
CAN to I2C Bridge

Design Description

This subsystem demonstrates how to build a CAN-I2C bridge. CAN-I2C bridge allows a device to send/receive information on one interface and receive/send the information on the other interface Download the code for this example.Two example codes are provided to support I2C to work in controller mode or target mode respectively.

Figure 1-1 shows a functional diagram of this subsystem. Please note that one line is added for IO interrupt to implement message transmission from I2C target to I2C controller.

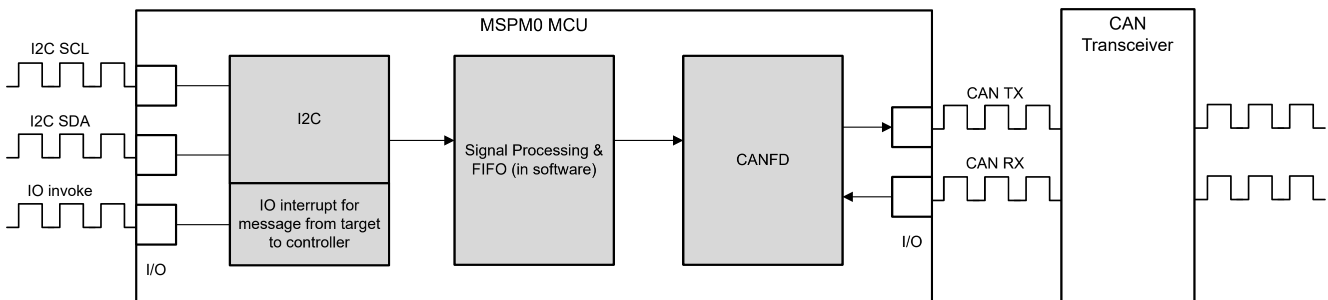

Figure 1-1. Subsystem Functional Block Diagram

Required Peripherals

This application requires CANFD and I2C.

Table 1-1. Required Peripherals

Sub-block Functionality	Peripheral Use	Notes
CAN interface	(1x) CANFD	Called *MCAN0_INST* in code
I2C interface	(1x) I2C	Called *I2C_INST* in code

Design Steps

1. Determine the basic setting of CAN interface, including CAN mode, bit timing, message RAM configuation and so on. Consider which setting is fixed and which setting is changed in the application. In example code, CANFD is used with 250kbit/s arbitration rate and 2Mbit/s data rate.
    a. Key features of the CAN-FD peripheral include:
        i. Dedicated 1KB message SRAM with ECC
        ii. Configurable transmit FIFO, transmit queue and event FIFO (up to 32 elements)
        iii. Up to 32 dedicated transmit buffers and 64 dedicated receive buffers. Two configurable receive FIFOs (up to 64 elements each)
        iv. Up to 128 filter elements
    b. If CANFD mode is enabled:
        i. Full support for 64-byte CAN-FD frames
        ii. Up to 8Mbit/s bit rate
    c. If CANFD mode is disabled:
        i. Full support for 8-byte classical CAN frames

Copyright © 2025 Texas Instruments Incorporated

ii. Up to 1Mbit/s bit rate

2. Determine the CAN frame, including data length, bit rate switching, identifier, data and so on. Consider which part is fixed and which part need to be changed in the application. In example code, identifier, data length and data can change in different frames, while others are fixed. Note that users need to modify the code if protocol communication is required.

```
/**
 *  @brief  Structure for MCAN Rx Buffer element.
 */
typedef struct {
    /*! Identifier */
    uint32_t id;
    /*! Remote Transmission Request
     *   0 = Received frame is a data frame
     *   1 = Received frame is a remote frame
     */
    uint32_t rtr;
    /*! Extended Identifier
     *   0 = 11-bit standard identifier
     *   1 = 29-bit extended identifier
     */
    uint32_t xtd;
    /*! Error State Indicator
     *   0 = Transmitting node is error active
     *   1 = Transmitting node is error passive
     */
    uint32_t esi;
    /*! Rx Timestamp */
    uint32_t rxts;
    /*! Data Length Code
     *   0-8  = CAN + CAN FD: received frame has 0-8 data bytes
     *   9-15 = CAN: received frame has 8 data bytes
     *   9-15 = CAN FD: received frame has 12/16/20/24/32/48/64 data bytes
     */
    uint32_t dlc;
    /*! Bit Rat Switching
     *   0 = Frame received without bit rate switching
     *   1 = Frame received with bit rate switching
     */
    uint32_t brs;
    /*! FD Format
     *   0 = Standard frame format
     *   1 = CAN FD frame format (new DLC-coding and CRC)
     */
    uint32_t fdf;
    /*! Filter Index */
    uint32_t fidx;
    /*! Accepted Non-matching Frame
     *   0 = Received frame matching filter index FIDX
     *   1 = Received frame did not match any Rx filter element
     */
    uint32_t anmf;
    /*! Data bytes.
     *   Only first dlc number of bytes are valid.
     */
    uint16_t data[DL_MCAN_MAX_PAYLOAD_BYTES];
} DL_MCAN_RxBufElement;
```

3. Determine the basic setting of I2C interface, including I2C mode, bus speed, target address, FIFO and so on. Consider which setting is fixed and which setting is changed in the application. One example code is used for I2C controller with 400kHz bus speed, the other one is used for I2C target with address 0x48.
   a. Key features of the I2C peripheral include:
      i. Configurable as a controller or a target with a bit rate up to 1Mbps
      ii. Independent 8-byte FIFOs for reception and transmission
      iii. Dual target address capability, glitch suppression
      iv. Independent controller and target interrupt generation, and hardware support for DMA
      v. Controller operation with arbitration, clock synchronization, multiple controller support

4. Determine the I2C message format. Typically I2C is transmitted in bytes. To achieve high-level communication, users can implement frame communication through software. If necessary, users can also introduce specific communication protocols. In example code, the message format is < 55 AA ID1 ID2 ID3

Copyright © 2025 Texas Instruments Incorporated

ID4 Length Data1 Data2 ...>. Users can send data through I2C as the same format. 55 AA is the header. ID area is 4 bytes. Length area is 1 byte, indicating the data length. Note that if users need to modify the I2C packet form, the code for frame acquisition and parsing also need to be modified.

Table 1-2. I2C Packet Form

Header	Address	Data Length	Data
0x55 0xAA	4 bytes	1 byte	(Data Length) bytes

5. Determine the bridge structure, including what messages need to be converted, how to convert messages and so on.
    a. Consider whether the bridge is one-way or two-way. Typically each interface has two functions: receiving and sending. Consider whether only some functions need to be included (such as I2C reception and CAN transmission). In example code, CAN-I2C bridge is a two-way structure. Since the receiving and transmitting of the I2C target are controlled by the I2C controller, the I2C target cannot initiate transmission to the I2C controller. To achieve communication from the target to the controller, a line is added to this design. The target's IO pull-down notifies the controller that there is information to be sent.
    b. Consider what information to convert and the corresponding carrier(variable, FIFO). In example code, identifier, data and data length are convert from one interface to the other interface. There are two FIFOs defined in code as shown in Figure 1-2.

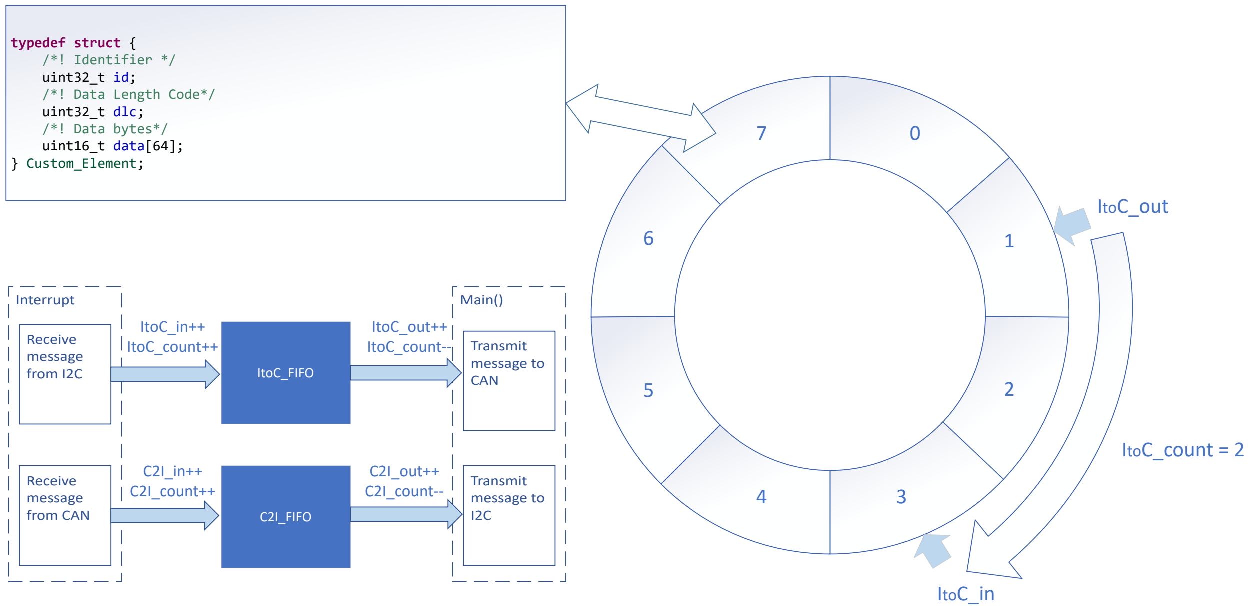

Figure 1-2. Bridge Structure

6. (Optionally) Consider priority design, congestion situation, error handling, and so on.

Copyright © 2025 Texas Instruments Incorporated

Design Considerations

1. Consider the information flow in the application to determine the information to be received or sent by each interface, the protocols to be followed, and design appropriate information transfer carriers to connect different interfaces.
2. The recommendation is to test the interface separately first, and then implement the overall bridge function. In addition, consider the handling of abnormal situations, such as communication failure, overload, frame format error, and so on.
3. The recommendation is to implement interface functions through interrupts to make sure of timely communication. In example code, interface functions are usually implemented in the interrupt, and the transfer of information is completed in the main() function.

Software Flowchart

Figure 1-3 shows the code flow diagram for *CAN-I2C bridge* which explains how the messages received in one interface and sent in the other interface. The *CAN-I2C bridge* can be divided into four independent tasks: receive from I2C, receive from CAN, transmit through CAN, transmit through I2C. Two FIFOs implement bidirectional message transfer and message caching.

Note that I2C is a communication method that I2C controller control the transmit and receive. In general, I2C target cannot initiate communication. For I2C target-to-controller communication, I2C target can pull down the IO when messages needed to be sent, as shown in Figure 1-3. I2C controller can initiate I2C read command in IO interrupt when IO is detected low, as shown in Figure 1-4. In this demo, I2C can be configured as I2C target or controller.

Copyright © 2025 Texas Instruments Incorporated

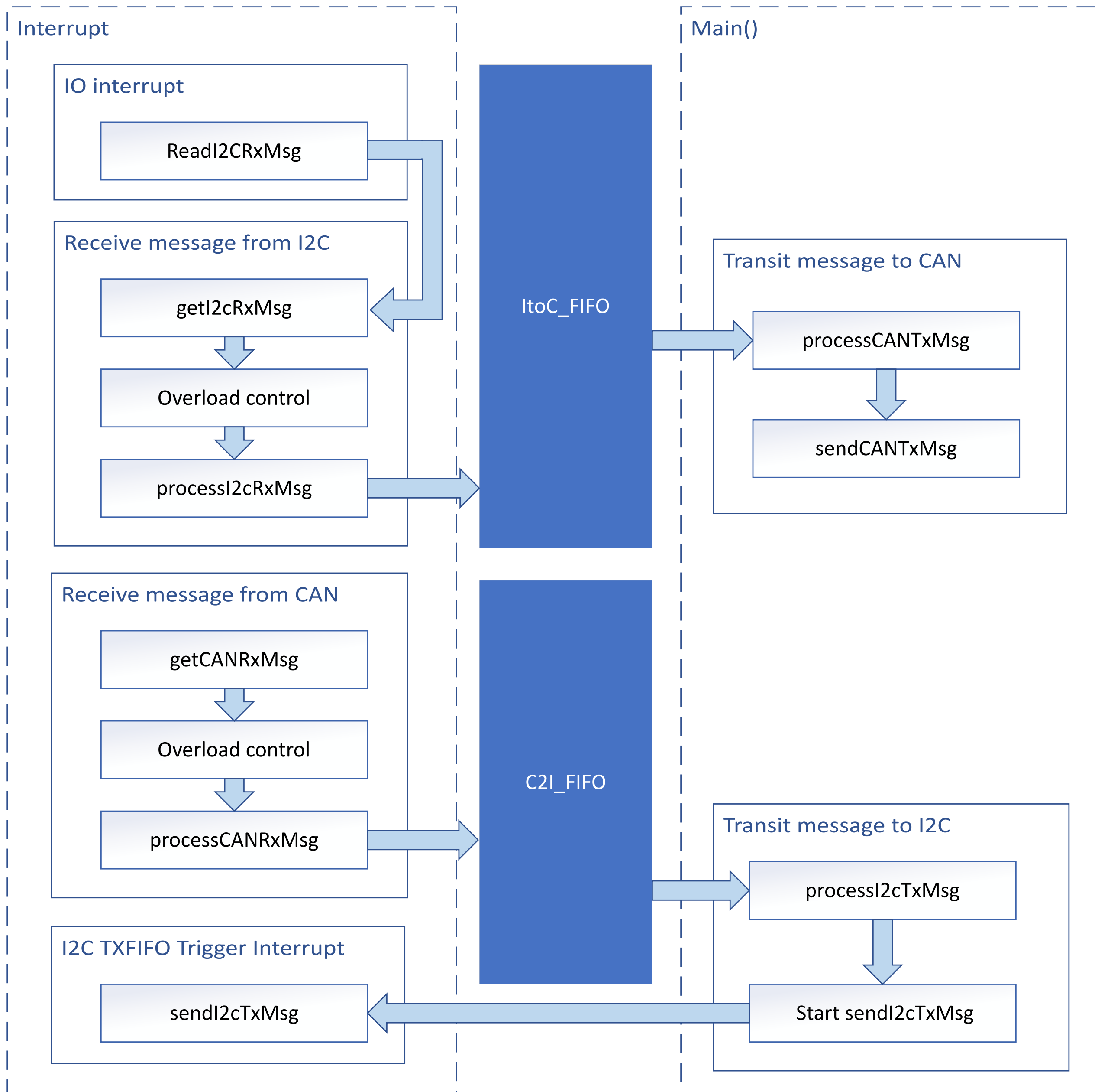

Figure 1-3. Application Software Flowchart of CAN-I2C (I2C Controller) Bridge

Copyright © 2025 Texas Instruments Incorporated

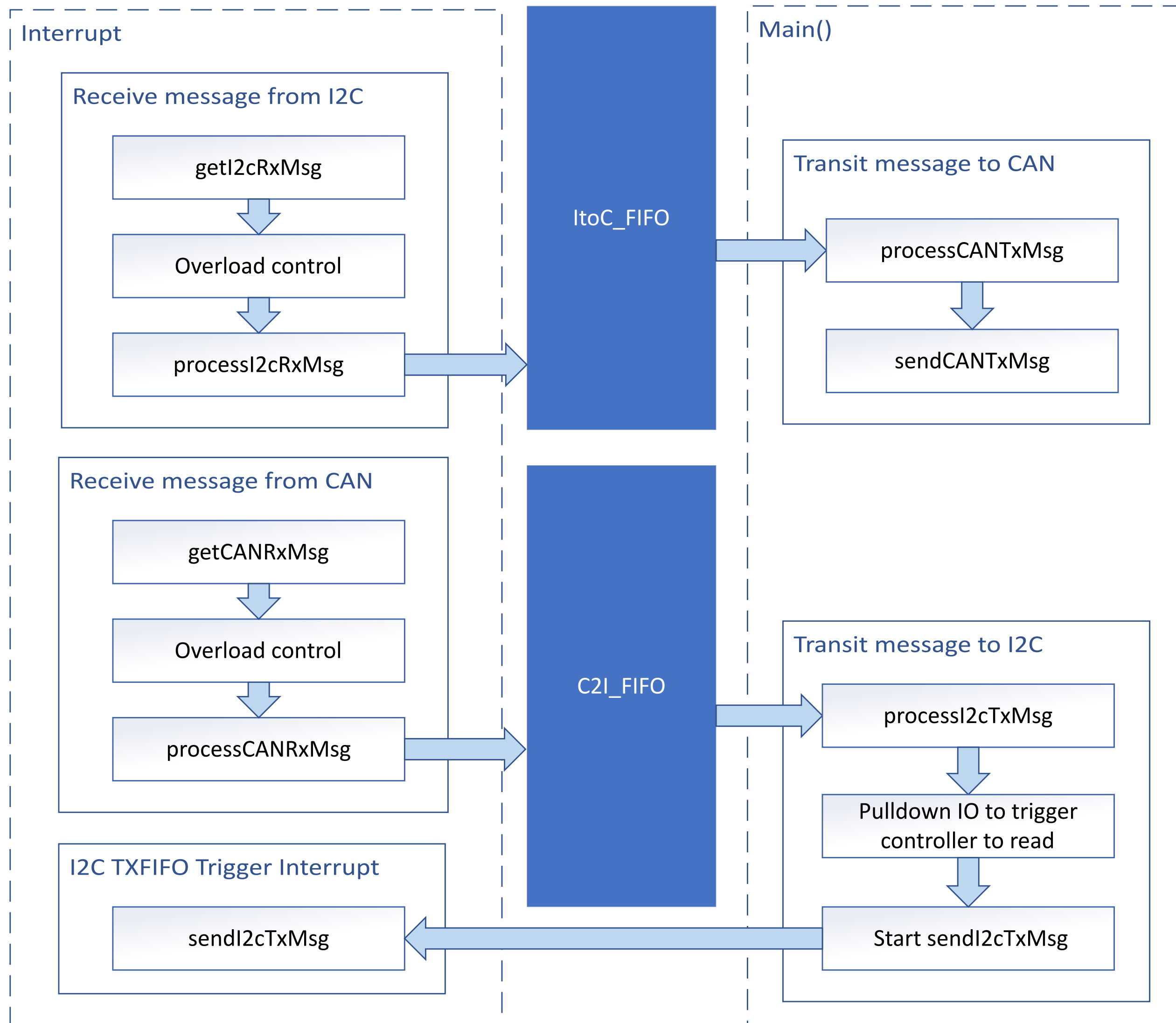

Figure 1-4. Application Software Flowchart of CAN-I2C (I2C Target) Bridge

Copyright © 2025 Texas Instruments Incorporated

Device Configuration

This application makes use of TI System Configuration Tool (SysConfig) graphical interface to generate the configuration code for the CAN and I2C. Using a graphical interface to configure the device peripherals streamlines the application prototyping process.

The code for what is described in Figure 1-3 can be found in the files from example code as shown in Figure 1-5.

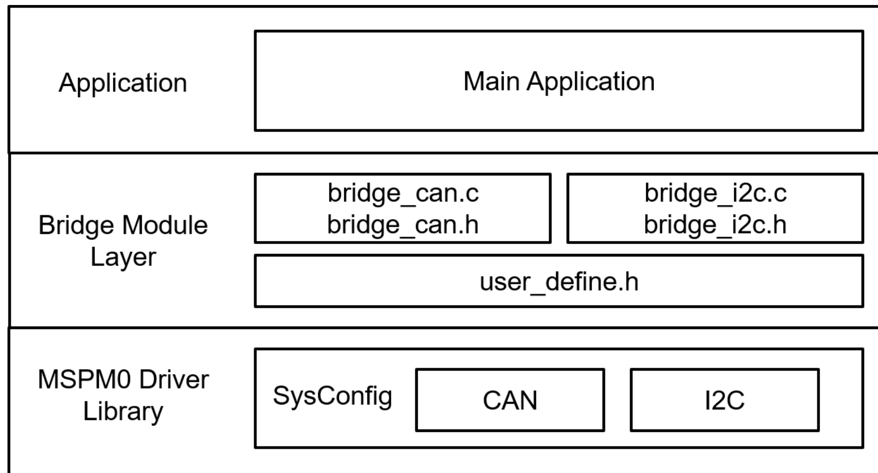

Figure 1-5. File Structure

Application Code

The following code snippet shows where to modify the interface function. Functions in table are categorized into different files. Functions for I2C receive and transmit are included in bridge_i2c.c and bridge_i2c.h. Functions for CAN receive and transmit are included in bridge_can.c and bridge_can.h. Structure of FIFO element is defined in user_define.h.

Users can easily separate functions by file. For example, if only I2C functions are needed, users can reserve bridge_i2c.c and bridge_i2c.h to call the functions.

See the MSPM0 SDK and DriverLib documentation for the basic configuration of peripherals.

Table 1-3. Functions and Descriptions

Tasks	Functions	Description	Location
I2C receive	readI2CRxMsg_controller()	Send a read request to slave (I2C master only)	bridge_i2c.c bridge_i2c.h
	getI2CRxMsg_controller()	Get the received I2C message (I2C master only)	
	getI2CRxMsg_target()	Get the received I2C message (I2C slave only)	
	processI2cRxMsg()	Convert the received I2C message format and store it into gI2C_RX_Element	
I2C transmit	processI2cTxMsg()	Convert the gI2C_TX_Element format to be sent through I2C	
	sendI2CTxMsg_controller()	Send message through I2C (I2C master only)	
	sendI2CTxMsg_target()	Send message through I2C (I2C slave only)	
CAN receive	getCANRxMsg()	Get the received CAN message	bridge_can.c bridge_can.h
	processCANRxMsg()	Convert the received CAN message format and store the message into gCAN_RX_Element	
CAN transmit	processCANTxMsg()	Convert the gCAN_TX_Element format to be sent through CAN	
	sendCANTxMsg()	Send message through CAN	

Copyright © 2025 Texas Instruments Incorporated

Custom_Element is the structure defined in user_define.h. Custom_Element is used as the structure of FIFO element, output element of I2C/CAN transmit and input element of I2C/CAN receive. Users can modify the structure according to the need.

```
typedef struct {
    /*! Identifier */
    uint32_t id;
    /*! Data Length Code*/
    uint32_t dlc;
    /*! Data bytes*/
    uint16_t data[64];
} Custom_Element;
```

For FIFO, there are 2 global variables used as FIFO. 6 global variables are used to trace the FIFO.

```
Custom_Element ItoC_FIFO[ItoC_FIFO_SIZE];
Custom_Element C2I_FIFO[C2I_FIFO_SIZE];
uint16_t ItoC_in = 0;
uint16_t ItoC_out = 0;
uint16_t ItoC_count = 0;
uint16_t C2I_in = 0;
uint16_t C2I_out = 0;
uint16_t C2I_count = 0;
```

Results

By using CAN analyzer, users can send and receive messages on the CAN side. As a demonstration, two launchpads can be used as two CAN-I2C bridges (one I2C master and one I2C slave) to form a loop. When the CAN analyzer sends CAN messages through master launchpad, the analyzer can receive CAN messages from the slave launchpad.

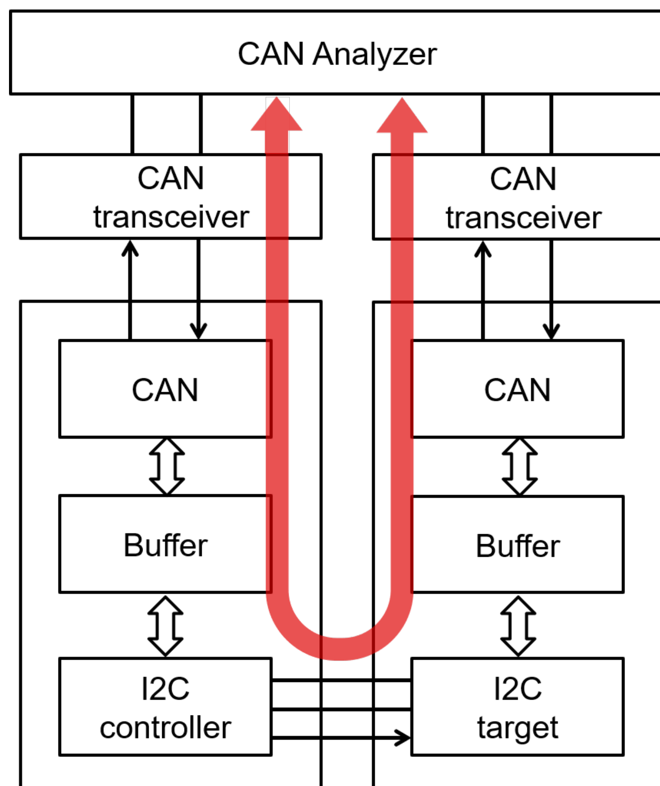

Figure 1-6. Demonstration

Copyright © 2025 Texas Instruments Incorporated

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

Figure 1-7. Messages Sent and Received by CAN Analyzer for the Demo

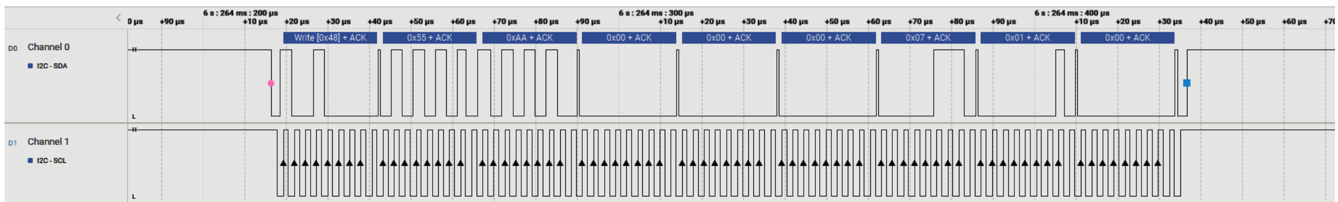

Figure 1-8. PC Terminal Program of Logic Analyzer

Additional Resources

- Texas Instruments, Download the MSPM0 SDK
- Texas Instruments, Learn more about SysConfig
- Texas Instruments, MSPM0 G-Series 80-MHz Microcontrollers, technical reference manual
- Texas Instruments, MSPM0G LaunchPad development kit
- Texas Instruments, MSPM0 CAN academy
- Texas Instruments, MSPM0 I2C academy

1 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (January 2024) to Revision A (August 2025) **Page**

- Removed Compatible Devices section... 1

Trademarks

All trademarks are the property of their respective owners.

Copyright © 2025 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated