

Subsystem Design

CAN to SPI Bridge



Yuhao Zhao

Design Description

This subsystem demonstrates how to build a CAN-SPI bridge. CAN-SPI bridge allows a device to send or receive information on one interface and receive or send the information on the other interface [Download the code for this example](#). The subsystem supports SPI to work in controller mode or peripheral mode.

Figure 1-1 shows a functional diagram of this subsystem.

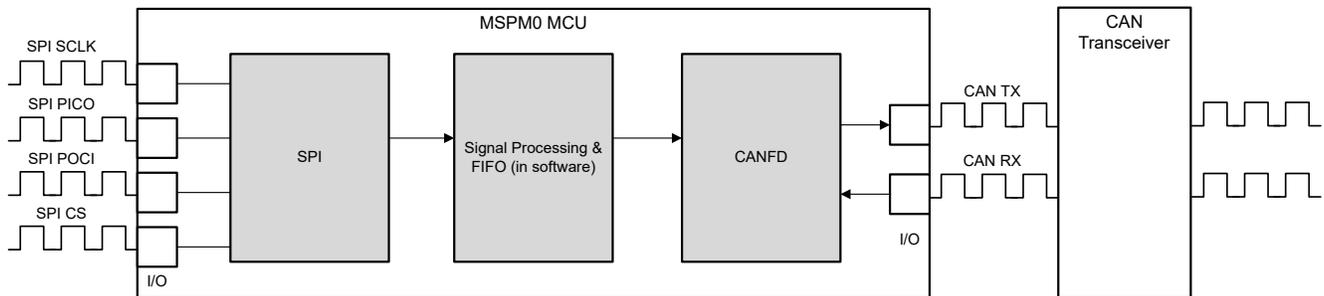


Figure 1-1. Subsystem Functional Block Diagram

Required Peripherals

This application requires CANFD and SPI.

Table 1-1. Required Peripherals

Sub-block Functionality	Peripheral Use	Notes
CAN interface	(1x) CANFD	Called <i>MCAN0_INST</i> in code
SPI interface	(1x) SPI	Called <i>SPI_0_INST</i> in code

Compatible Devices

Based on the requirements in Table 1-1, this example is compatible with the devices in Table 1-2. The corresponding EVM can be used for prototyping.

Table 1-2. Compatible Devices

Compatible Devices	EVM
MSPM0G35xx	LP-MSPM0G3507

Design Steps

1. Determine the basic setting of CAN interface, including CAN mode, bit timing, message RAM configuration 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
 - 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 SPI interface, including SPI mode, bit rate, frame size, FIFO, and so on. Consider which setting is fixed and which setting is changed in the application. In example code, SPI can be set as controller or peripheral. SPI operates at 500k bit rate in controller mode.
- a. Key features of the SPI include:
 - i. Configurable as a controller or a peripheral
 - ii. Programmable clock bit rate and prescaler

- iii. Separate transmit (TX) and receive (RX) first-in first-out buffers (FIFOs);
 - iv. Supports PACKEN feature and single bit parity
 - v. Programmable data frame size and programmable SPI mode
 - vi. Interrupts for transmit and receive FIFOs, overrun and timeout interrupts, and DMA done
4. Determine the SPI frame. Typically SPI 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 ID4 Length Data1 Data2 ...>. Users can send data to the CAN bus from the terminal by entering data 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 SPI frame, the code for frame acquisition and parsing also need to be modified.

Table 1-3. SPI Frame 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 SPI reception and CAN transmission). In example code, CAN-SPI bridge is a two-way structure.
 - 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](#).

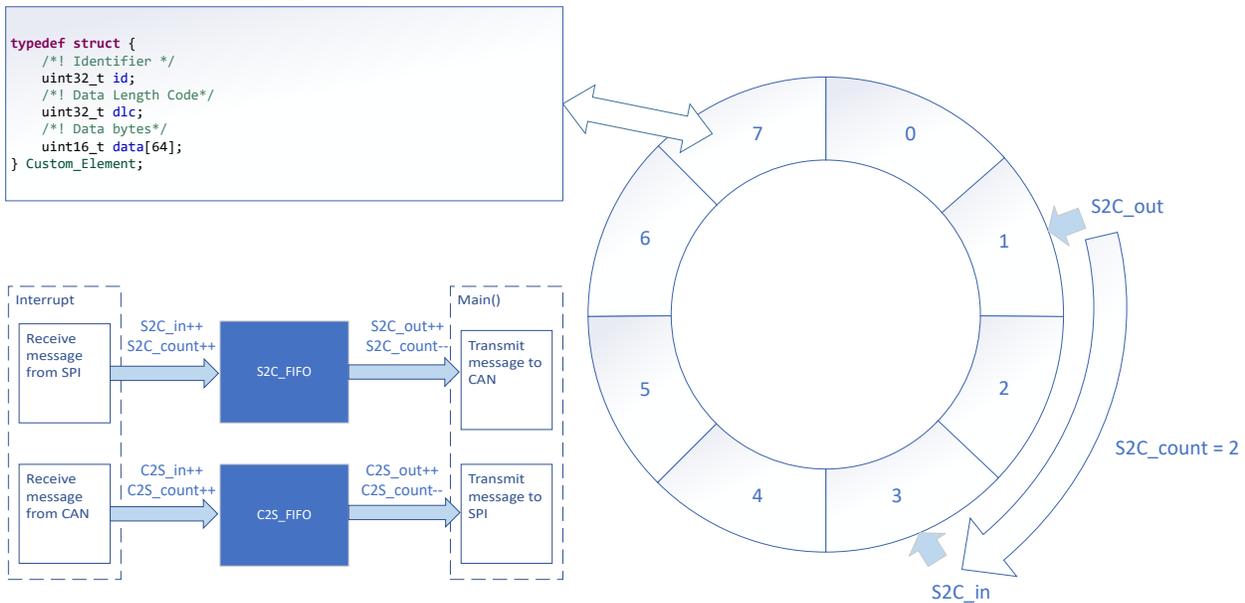


Figure 1-2. Bridge Structure

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

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.

- 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

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

Note that SPI is a communication method that sends and receives at the same time. When the controller initiates sending a byte, the controller expects to receive a byte. In the design of this article, SPI RX interrupt is not only used for SPI receive, but also used to fill the TX data into SPI TX FIFO. If SPI works in controller mode, SPI communication starts immediately after SPI TX FIFO is stored by data. If SPI works in peripheral mode, SPI can wait for the controller to initiate communication after data is stored. In this demo, users can select the mode of SPI.

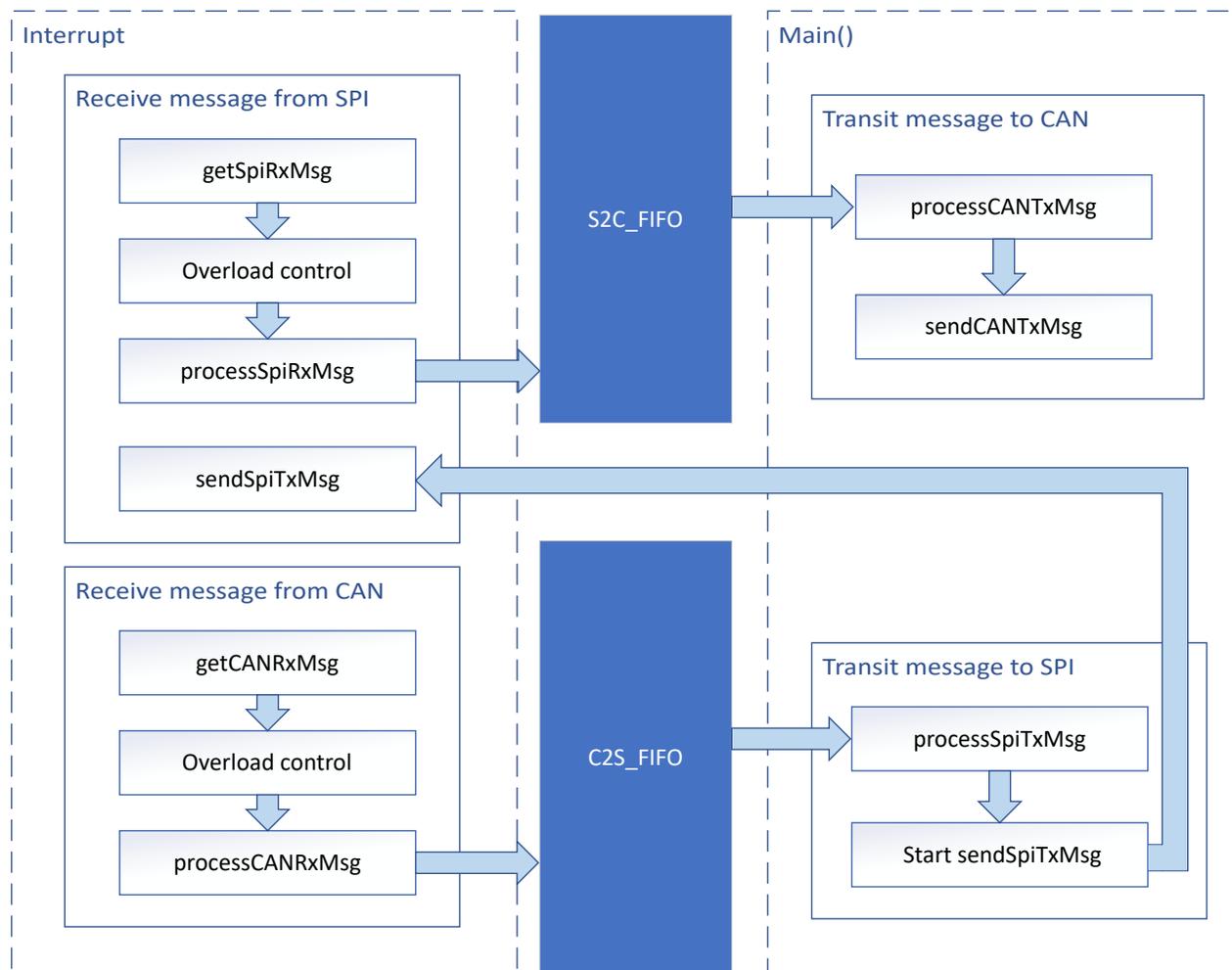


Figure 1-3. Application Software Flowchart

Device Configuration

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

The user can configure the SPI to be controller or peripheral in the Sysconfig.

The code for what is described in [Figure 1-3](#) can be found in the files from example code as shown in [Figure 1-4](#).

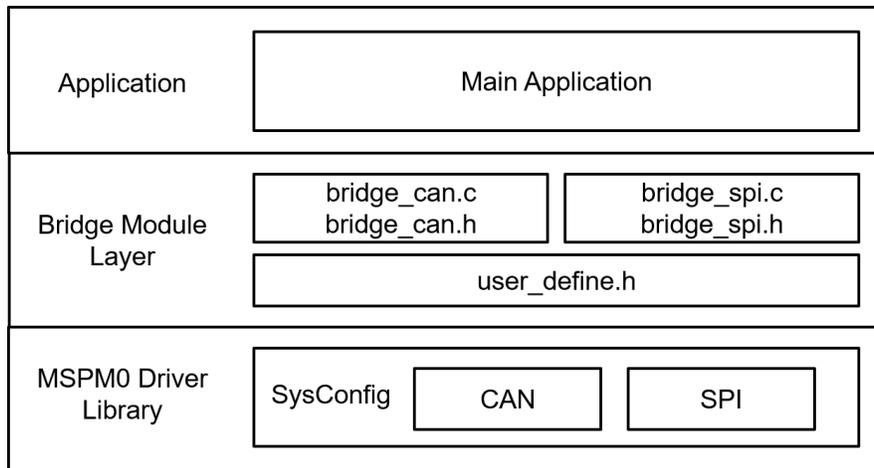


Figure 1-4. File Structure

Application Code

The following code snippet shows where to modify the interface function. Functions in table are categorized into different files. Functions for SPI receive and transmit are included in `bridge_spi.c` and `bridge_spi.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 SPI functions are needed, users can reserve `bridge_spi.c` and `bridge_spi.h` to call the functions.

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

Table 1-4. Functions and Descriptions

Tasks	Functions	Description	Location
SPI receive	<code>getSpiRxMsg()</code>	Get the received SPI message	<code>bridge_spi.c</code> <code>bridge_spi.h</code>
	<code>processSpiRxMsg()</code>	Convert the received SPI message format and store it into <code>gSPI_RX_Element</code>	
SPI transmit	<code>processSpiTxMsg()</code>	Convert the <code>gSPI_TX_Element</code> format to be sent through SPI	<code>bridge_spi.c</code> <code>bridge_spi.h</code>
	<code>sendSpiTxMsg()</code>	Send message through SPI	
CAN receive	<code>getCANRxMsg()</code>	Get the received CAN message	<code>bridge_can.c</code> <code>bridge_can.h</code>
	<code>processCANRxMsg()</code>	Convert the received CAN message format and store the message into <code>gCAN_RX_Element</code>	
CAN transmit	<code>processCANTxMsg()</code>	Convert the <code>gCAN_TX_Element</code> format to be sent through CAN	<code>bridge_can.c</code> <code>bridge_can.h</code>
	<code>sendCANTxMsg()</code>	Send message through CAN	

Custom_Element is the structure defined in user_define.h. Custom_Element is used as the structure of FIFO element, output element of SPI/CAN transmit and input element of SPI/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 S2C_FIFO[S2C_FIFO_SIZE];
Custom_Element C2S_FIFO[C2S_FIFO_SIZE];
uint16_t S2C_in = 0;
uint16_t S2C_out = 0;
uint16_t S2C_count = 0;
uint16_t C2S_in = 0;
uint16_t C2S_out = 0;
uint16_t C2S_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-SPI bridges(one SPI controller and one SPI peripheral) to form a loop. When the CAN analyzer sends CAN messages through controller launchpad, it will receive CAN messages from the peripheral launchpad.

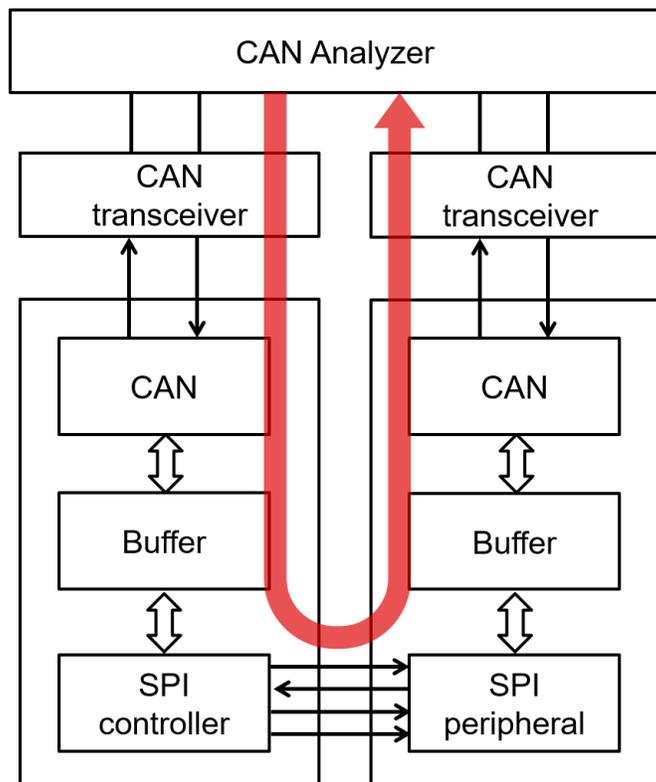


Figure 1-5. Demonstration

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALI		
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...

Figure 1-6. Messages Sent and Received by CAN Analyzer for the Demo

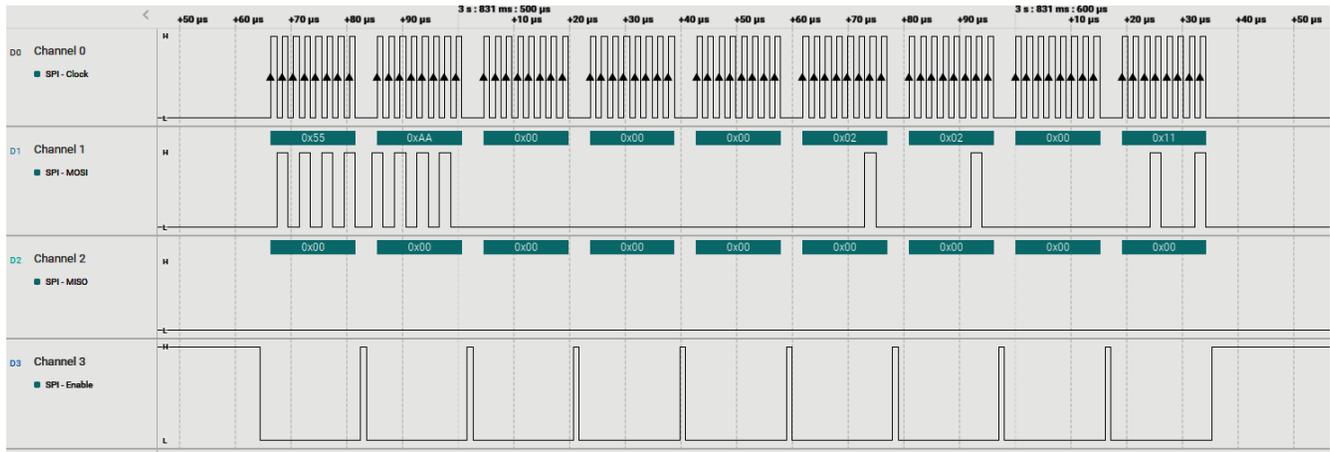


Figure 1-7. PC Terminal Program of Logic Analyzer

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 SPI academy](#)

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](https://www.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 © 2024, Texas Instruments Incorporated