Application Note

*如何对 **TXE81XX SPI I/O** 扩展器系列进行编程*

Tyler Townsend

摘要

本应用手册详细阐述了 TXE81XX 系列 I/O 扩展器的编程方法。

内容

商标

所有商标均为其各自所有者的财产。

1 简介

TXE81XX 器件系列是 TI 的首款 SPI 转 GPIO 扩展器。本应用手册详细介绍了这些器件的编程方法，并对 24 位 SPI 字结构进行了解释。本文还讲解了寄存器映射，并说明了该 I/O 扩展器中部分常用寄存器的功能用途。

2 设置和配置

TXE81XX SPI 转 GPIO 扩展器系列通过 4 线制 SPI 接口进行控制：MISO（主输入从输出）、MOSI（主输出从输入）、SCLK（时钟）和 CS（芯片选择）线路。到目前为止，术语已重新定义为如下所示。

MISO → POCI（外设输出控制器输入）

MOSI → PICO（外设输入控制器输出）

SCLK 保持不变

CS 保持不变

TXE81XX 所采用的术语如下。

SDI（串行数据输入）→ MOSI/PICO

SDO（串行数据输出）→ MISO/POCI

SCLK → 时钟

CS → 芯片选择

在本例中，使用 M0 LaunchPad LP-MSPM0C1104 通过以下连接对 TXE8124 进行编程。

VCC = 3.3V →（引脚 2）

PA11 → SCLK（引脚 29）

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

PA16 → MISO/POCI/SDO（引脚 1）

PA18 → MOSI/PICO/SDI（引脚 30）

PA2 → /CS（引脚 28）

GND →（引脚 3）

对 500kHz SPI 时钟设置 SPI 参数：CPOL（时钟空闲极性）= 0（低电平），CPHA（时钟相位）= 0（上升沿 / 前沿）。

图 2-1 是用于 TXE8124 和 MSPM0 之间物理连接的完整方框图。

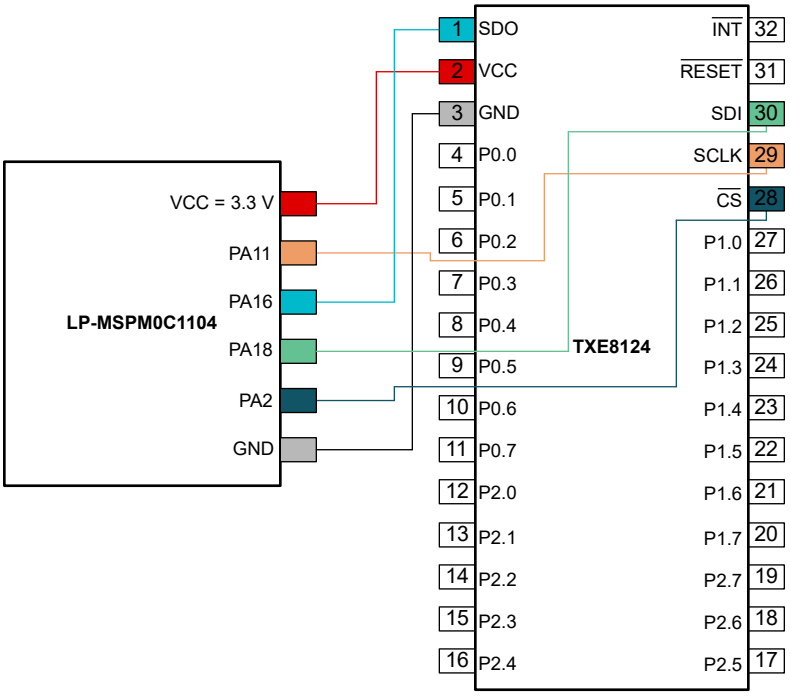

图 2-1. LP-MSPM0C1104 和 TXE8124 的接线方案

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

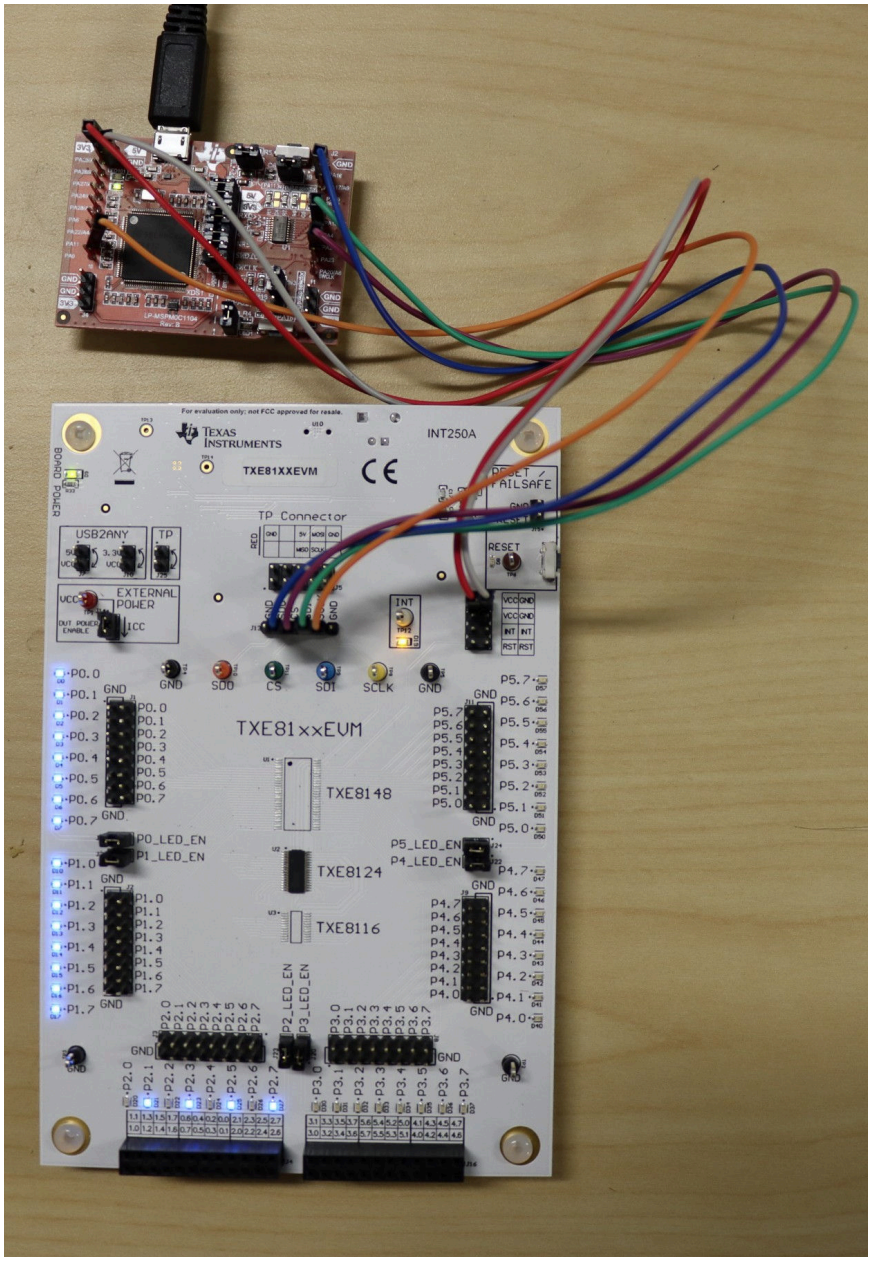

图 2-2. LP-MSPM0C1104 连接至 TXE81XXEVM 示例

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

3 TXE81XX 24 位 SPI 字定义

TXE81XX 使用以下 24 位字结构来启动 SPI 对器件读写操作。

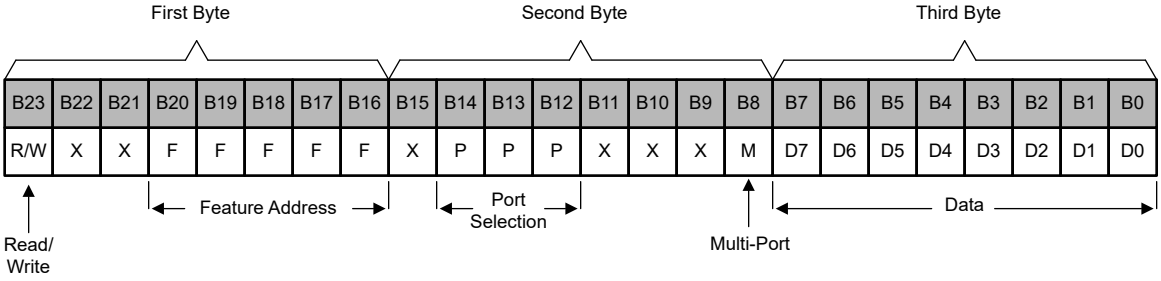

图 3-1. TXE81XX 24 位 SPI 字

SPI 字的长度为 24 位，需要在 SDI 上以 MSB 优先的方式移入数据。

B23 → R/W = 读取或写入位（R = 1，W = 0）

B22 - B21 = (X) 不用考虑

B20 - B16 = 特性地址（5 位）

B15 = (X) 不用考虑

B14 - B12 = 端口选择（端口 0 = 000，端口 1 = 001，端口 2 = 010）

B11 - B9 = (X) 不用考虑

B8 = 多端口位（多端口启用 = 1，多端口禁用 = 0）

B7 - B0 = 数据字节

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

4 SPI 写入步骤

SPI 写入命令涉及将数据写入指定的寄存器，同时在 SDO 上读取之前的寄存器数据内容（全双工）。

1. 将 /CS 驱动为低电平。这将启用内部移位寄存器
2. 在 SDI 上将 24 位数据以 MSB 优先的方式移入器件。数据在时钟 (SCLK) 的上升沿必须稳定。
3. MSB 位 (B23) 必须为“0”，表示这是写入操作。
4. 16 位状态在 SDO 上发送。前 2 位是 2’b11（表示这是一个状态段）。接下来的 6 位 B21-B16 是故障状态寄存器的 D5 至 D0 位。最后 8 位 B15-B8 全为 0。
5. 寄存器中先前的数据在 SDO (B7-B0) 上读取，而数据字节将写入 SDI (B7-B0) 上的寄存器。
6. 在传输最后一位数据后，如果没有更多数据要传输，则将 SCLK 驱动为低电平。
7. 将 /CS 置为无效（将其驱动为高电平），结束写入周期。

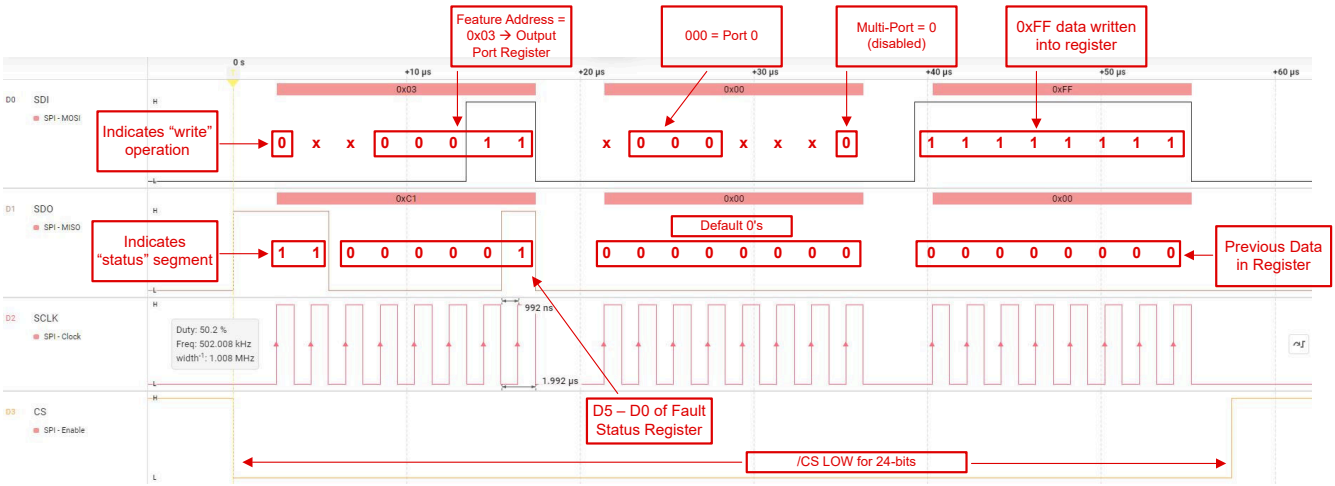

图 4-1. 使用 Saleae 逻辑分析仪的 SPI 写入示例

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

5 代码示例

当前版本的 Code Composer Studio 可从 TI 开发人员专区的 *Common Actions* 部分下方下载。

下载并打开 Code Composer Studio IDE 后，可通过浏览 Resource Explorer 中提供的软件示例找到示例代码。

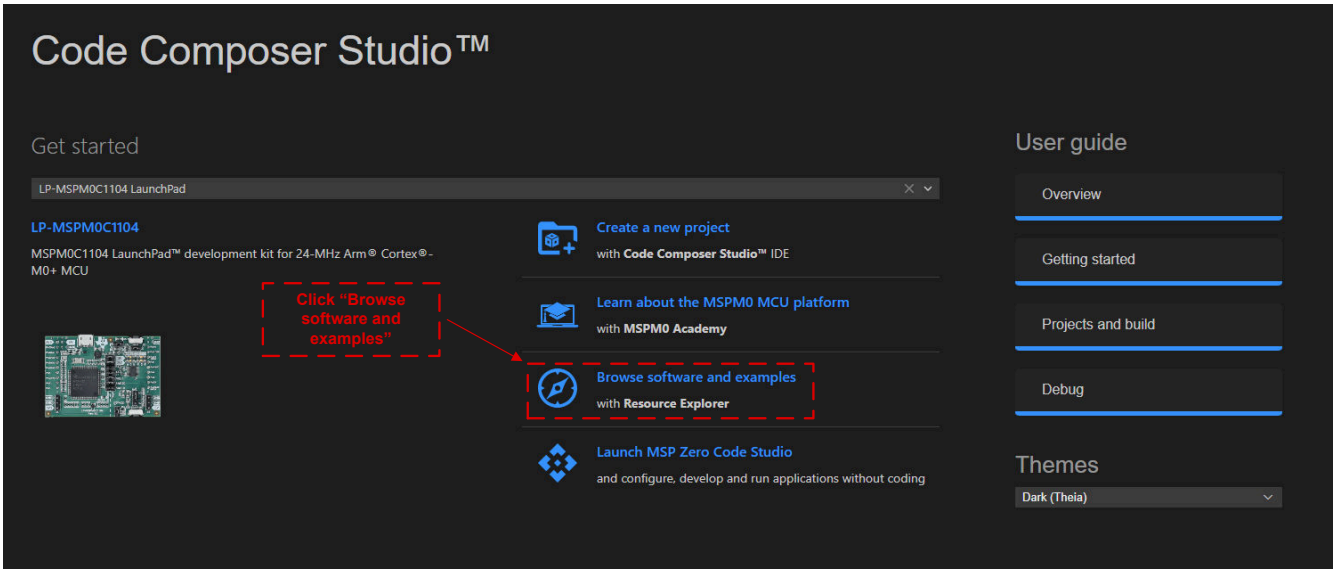

图 5-1. 在 Code Composer Studio IDE 中，选择 Browse Software and Examples

打开 Resource Explorer 后，按照以下目录查找示例代码：

Arm^R-Based Microcontrollers → Embedded Software → MSPM0 SDK – 2.04.00.06 → Examples → Development Tools → LP-MSPM0C1104 LaunchPad → DriverLib → spi_controller_internal_loopback_poll → No RTOS → GCC Compiler → spi_controller_internal_loopback_poll

点击 3 个点并选择“Import to CCS IDE”，即可将示例项目导入到 CCS IDE 中。

导入后，首先选择“spi_controller_internal_loopback_poll.syscfg”文件。

示例中的 SPI 设置可通过 SYSCONFIG GUI 进行修改。在 SPI 下方，选择以下配置以正确设置代码。

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

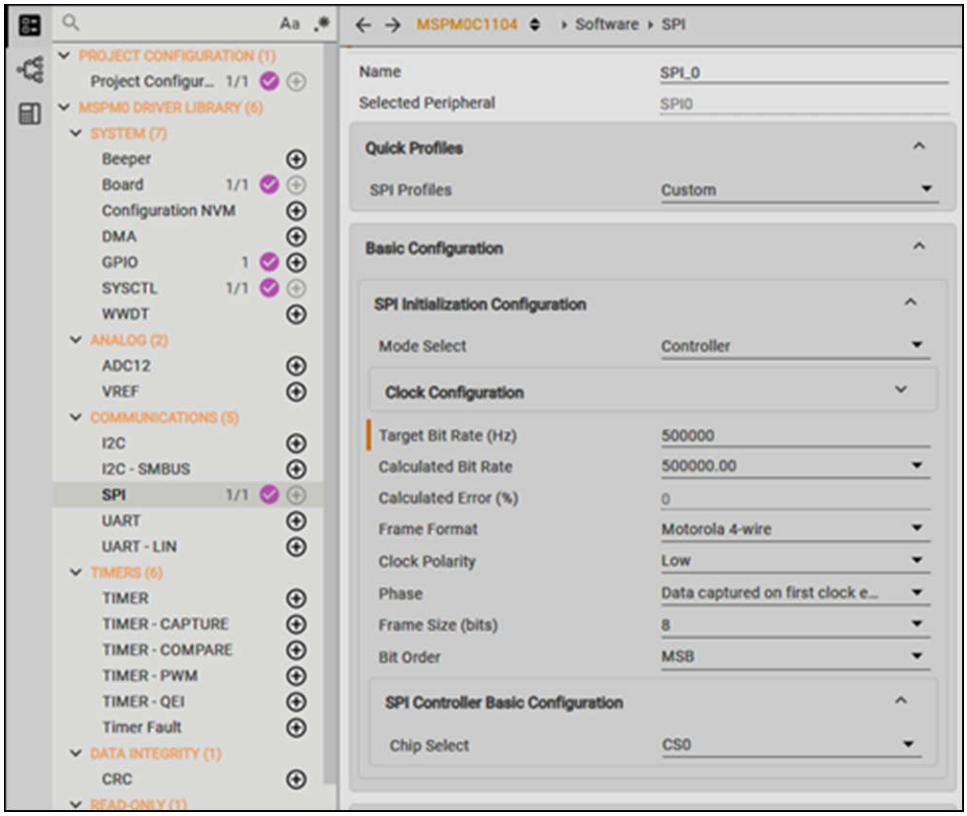

图 5-2. SYSCONFIG 设置

SPI_0 是所用 SPI 外设的名称。系统会创建一个自定义配置文件。模式应选择 *Controller*，目标比特率则应设为 500kHz。帧格式是 Motorola 4 线制 SPI。时钟极性设置为低电平。时钟相位 – 数据在第一个时钟沿（上升沿）采集。帧大小设置为 8 位。位顺序为 MSB 优先。芯片选择为 CS0 – PA11，本例使用 PA2 上的 GPIO 采集 TXE8124 器件的 3 字节帧结构。本示例忽略 CS0 – PA11。

在 *Advanced Configuration* 选项卡中进行以下选择，并确保 *Enable Internal Loopback* 未选中。

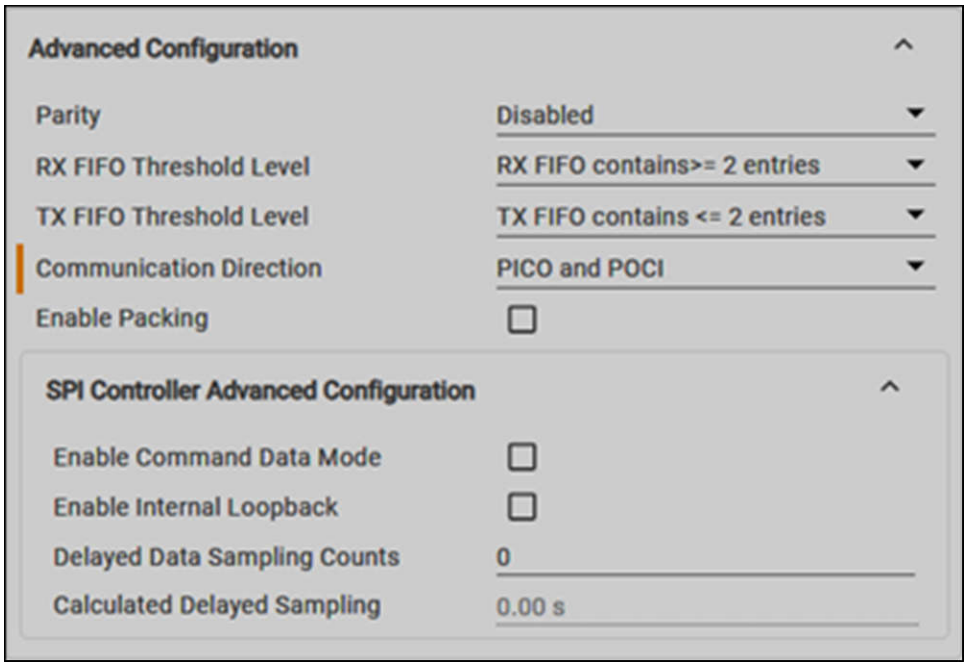

图 5-3. 在 Advanced Configuration 选项卡下，Enable Internal Loopback 处于禁用状态

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

SYSCONFIG GUI 会设置 SPI 配置，无需在 M0 驱动程序库中编写代码。这将设置 SPI 驱动程序，以便腾出更多时间专注于 TXE81XX 示例功能的开发。

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

6 示例代码

以下示例代码可直接覆盖替换 MSPM0 SDK 中的 spi_controller_internal_loopback_poll.c 示例代码。此 .c 文件已经过修改，包含适配 TXE81XX 器件 24 位字方案的 SPI 读写函数。

```
/*SPI Connections:
VCC = 3.3V
CPOL = 0
CPHA = 0
Clock Frequency = 500kHz
SCLK = PA11
MISO = PA16
MOSI = PA18
/CS = PA2
*/
#include "ti_msp_dl_config.h" //MSP driver library
//Define the TXE81xx Register Map from the TXE81xx d
#define scratch_reg (0x00)
#define device_ID (0x01)
#define input_reg (0x02)
#define output_reg (0x03)
#define config_reg (0x04)
#define pol_reg (0x05)
#define pp_od_sel (0x06)
#define od_conf (0x07)
#define pu_pd_en (0x08)
#define pu_pd_sel (0x09)
#define bus_holder (0x0A)
#define smart_int (0x0B)
#define int_mask (0x0C)
#define glitch_filter_en (0x0D)
#define int_flag_status (0x0E)
#define int_port_status (0x0F)
#define fail_safe_en_reg_1 (0x12)
#define fail_safe_en_reg_2 (0x13)
#define fail_safe_dir_config1 (0x14)
#define fail_safe_dir_config2 (0x15)
#define fail_safe_out1 (0x16)
#define fail_safe_out2 (0x17)
```


English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

```
#define fail_safe_redun (0x18)
#define fail_safe_fault (0x19)
#define software_reset (0x1A)
//define each port
//TXE8116 - Port 0 / Port 1
//TXE8124 - Port 0 / Port 1 / Port 2
#define port0 (0x00)
#define port1 (0x01)
#define port2 (0x02)
#define DELAY_500khz (50) //delay cycles to tune /CS for SPI comms at 500kHz
//Function Definitions
uint8_t SPI_Transfer(uint8_t feat_addr, uint8_t port, uint8_t data_byte, bool multi_port);
uint8_t SPI_read (uint8_t feat_addr, uint8_t port);
int main(void)
{
volatile uint8_t read_val = 0x00;
SYSCFG_DL_init(); //initalize SPI driver

read_val = SPI_Transfer(config_reg, port0, 0xFF, false); //write to configuration register 0 - set all output
//write outputs every second on port 0
while(1){
read_val = SPI_Transfer(output_reg, port0, 0xFF, false); //0xFF - LED's off
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);

read_val = SPI_Transfer(output_reg, port0, 0xAA, false); //0xAA - alternate LED's
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);
read_val = SPI_Transfer(output_reg, port0, 0x55, false); //0x55 - alternate the other way
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);

read_val = SPI_Transfer(output_reg, port0, 0x00, false); //0x00 - LED's ON
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);
}
}
//The SPI_Transfer both writes and returns the previous byte set in the 8-bit register
uint8_t SPI_Transfer(uint8_t feat_addr, uint8_t port, uint8_t data_byte, bool multi_port){
```


English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

```
uint8_t write_byte = feat_addr & 0x1F; //only keep B20 - B16
uint8_t port_byte = (port << 4) & 0x70; //only keep B14 - B12, first shift port number to upper nibble
uint8_t m_bit;
uint8_t status_byte1 = 0x00; //first 2 bits are 2'b11 (indicating status segment), next 6 bits are D5 to D0 bits of the fault flag register
uint8_t status_byte2 = 0x00; //8 bits are all 0
uint8_t reg_data_out = 0x00; //the current register data
if (multi_port == true){
m_bit = 0x01; // multi-port = 1 = enabled
} else {
m_bit = 0x00; // multi-port = 0 = disabled
}
DL_GPIO_clearPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN |
GPIO_LEDS_USER_TEST_PIN); //set /CS LOW
DL_SPI_transmitData8(SPI_0_INST, write_byte); //Transmit B23 - B16
DL_SPI_transmitData8(SPI_0_INST, port_byte + m_bit); //Transmit B15 - B8
DL_SPI_transmitData8(SPI_0_INST, data_byte); //Transmit B7 - B0
status_byte1 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //reading data on SDO 1st byte
status_byte2 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //reading data on SDO 2nd byte
reg_data_out = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //reading data on SDO 3rd byte - this is the register data returned (previously set byte in register)
delay_cycles(DELAY_500khz); //delay to time chip select
DL_GPIO_setPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN |
GPIO_LEDS_USER_TEST_PIN); //set /CS HIGH
return reg_data_out; //return the previous register byte
}
//create a function to directly read a register and the status bits for the fault flag register
uint8_t SPI_read (uint8_t feat_addr, uint8_t port){
uint8_t read_byte = (feat_addr & 0x1F) + 0x80; //only keep B20 - B16 and make B23 = 1 for a "READ"
uint8_t port_byte = (port << 4) & 0x70; //only keep B14-B12, first shift port number to upper nibble
uint8_t status_byte1 = 0x00; //first 2 bits are 2'b11 (indicating status segment), next 6 bits are D5 to D0 bits of the fault flag register
uint8_t status_byte2 = 0x00; //8 bits are all 0
uint8_t reg_data_out = 0x00; //the current register data

DL_GPIO_clearPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN |
GPIO_LEDS_USER_TEST_PIN); //set CS LOW
DL_SPI_transmitData8(SPI_0_INST, read_byte); //send read command byte
DL_SPI_transmitData8(SPI_0_INST, port_byte); //specify the port selection
DL_SPI_transmitData8(SPI_0_INST, 0x00); //0x00 dummy data to shift out 8 bits of register data on MISO
```


English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

```
status_byte1 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //first 8 bits of status
status_byte2 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //second 8 bits of status
reg_data_out = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //register data byte
delay_cycles(DELAY_500khz); //delay to time chip select
DL_GPIO_setPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN |
GPIO_LEDS_USER_TEST_PIN); //set CS HIGH
return reg_data_out; //return register byte
}
```


English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

7 总结

TXE 系列 SPI 转 GPIO 扩展器使用 24 位字格式。写入 TXE 器件是一种全双工操作，此操作可分别在 SDI 和 SDO 上的寄存器中写入和读取数据。直接读取 TXE 会导致从 SDO 上的寄存器输出数据。提供的示例代码为非优化设计，仅用于实现对 TXE 的简单读写命令，并展示该 IC 的通用操作格式。

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

8 参考资料

- 德州仪器 (TI)，*MSPM0 SDK*，Resource Explorer。

English Document: SCEA154
Copyright © 2026 Texas Instruments Incorporated

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、与某特定用途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安全、安保法规或其他要求。

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。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 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。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

版权所有 © 2026，德州仪器 (TI) 公司

最后更新日期：2025 年 10 月