参考資料

Application Note

TXE81XX SPI I/O エクスパンダ ファミリのプログラム方法

Tyler Townsend

概要

このアプリケーション ノートでは、TXE81XX ファミリの I/O エクスパンダのプログラミング方法を紹介します。

目次

商標

1 はじめに

TXE81XX デバイス ファミリは、TI 初の SPI から GPIO へのエクスパンダです。このアプリケーション ノートでは、これらのデバイスをプログラムする方法についての詳細と、24 ビットの SPI ワード構造の説明を提供します。また、レジスタ マップについても注目するので、I/O エクスパンダ内の一部の共通レジスタの目的を説明するのに役立ちます。

2 設定および構成

TXE81XX SPI から GPIO へのエクスパンダ ファミリは、以下の 4 線式 SPI インターフェイス経由で制御されます。MISO (マスタ入力 / スレーブ出力) ライン、MOSI (マスタ出力 / スレーブ入力) ライン、SCLK (クロック) ライン、および CS (チップ セレクト) ライン。現在に至るまで、用語は以下のように再定義されています。

MISO → POCI (ペリフェラル出力 / コントローラ入力)

MOSI→PICO (ペリフェラル入力 / コントローラ出力)

SCLK は変わりません

CS は変わりません

TXE81XX は以下の用語に従います。

SDI (シリアル データ入力) → MOSI / PICO

SDO (シリアル データ出力) → MISO / POCI

SCLK → クロック

CS → チップ セレクト

この例では、M0 LaunchPad LP-MSPM0C1104 を使用して、以下の接続で TXE8124 をプログラムしています。

VCC = 3.3V → (ピン 2)

PA11 → SCLK (ピン 29)

PA16 → MISO/POCI/SDO (ピン 1)

PA18 →MOSI/PICO/SDI (ピン 30)

PA2 → /CS (ピン 28)

GND → (ピン 3)

SPI 設定は、500kHz の SPI クロック、CPOL (クロック アイドル極性) = 0 (Low)、CPHA (クロック位相) = 0 (立ち上がりエッジ / 先端エッジ) に設定されます。

図 2-1 は、TXE8124 と MSPM0 の間の物理的な接続に使用される完全なブロック図です。

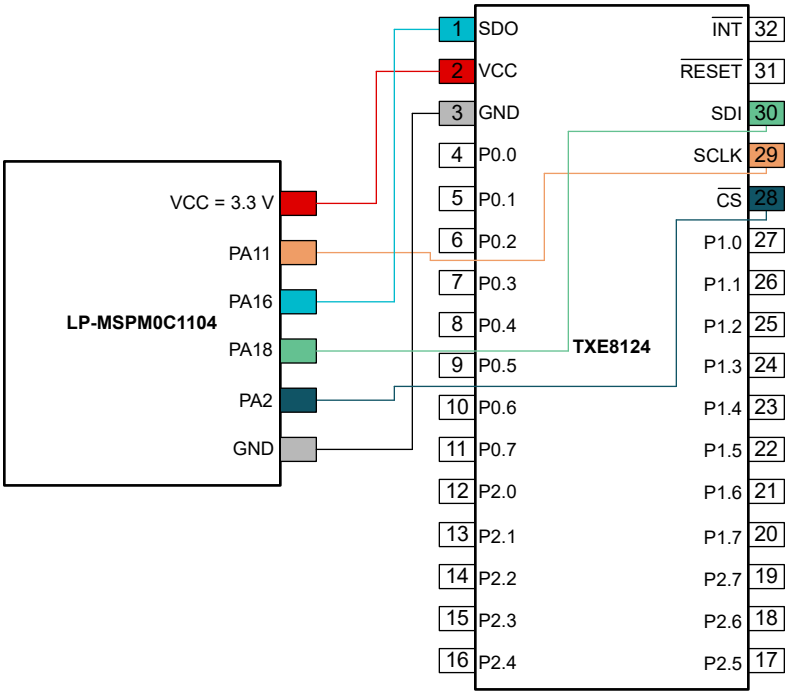

図 2-1. LP-MSPM0C1104 および TXE8124 の配線方法

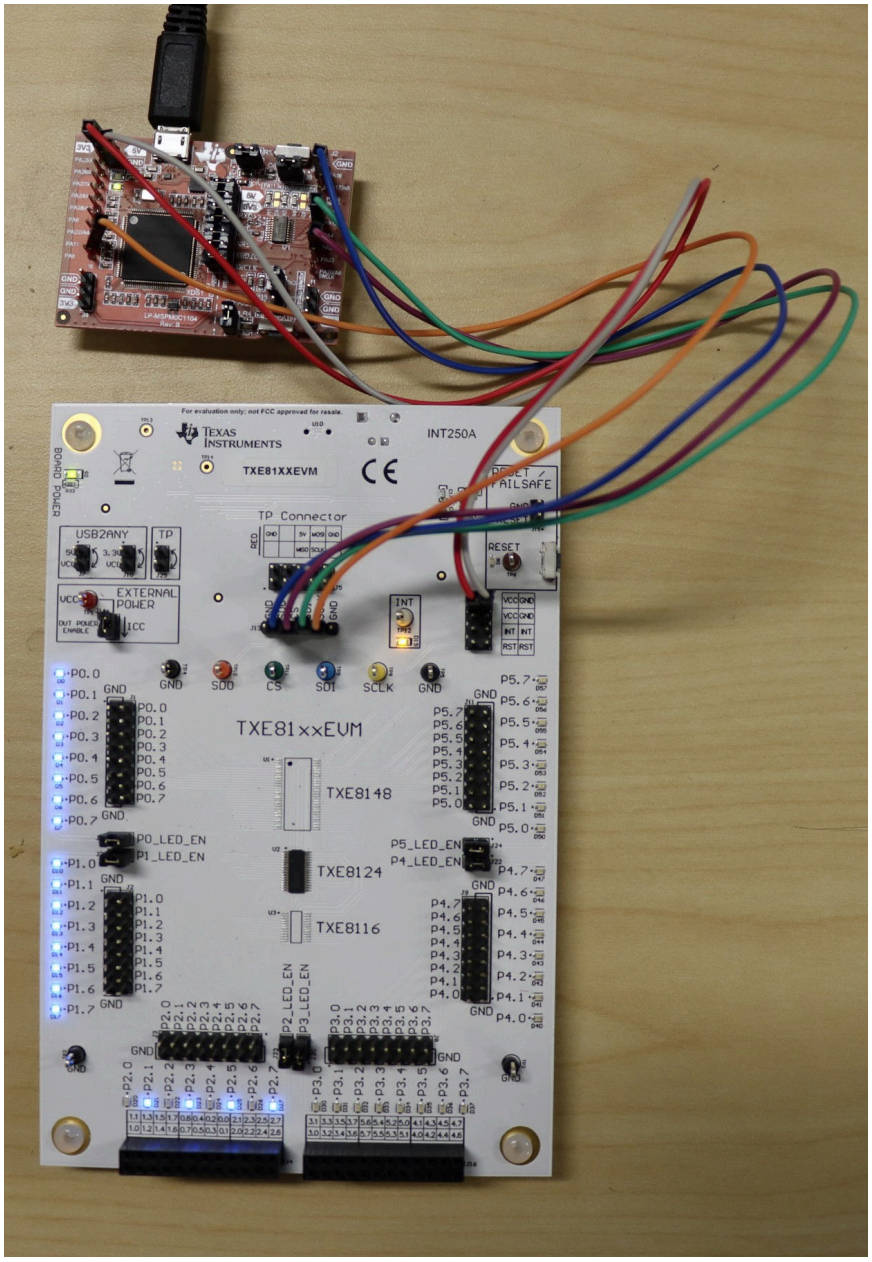

図 2-2. LP-MSPM0C1104 と TXE81XXEVM の接続例

3 TXE81XX 24 ビット SPI ワードの定義

TXE81XX は、次の 24 ビット ワード構造を使用して、デバイスとの間で SPI 書き込みおよび読み取りを開始します。

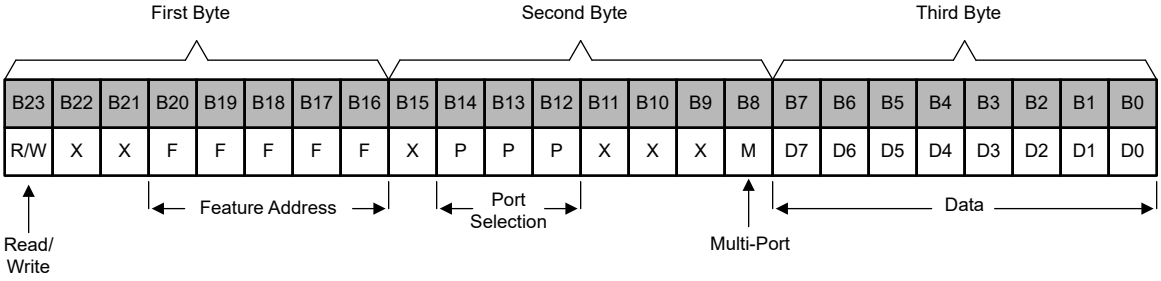

図 3-1. TXE81XX 24 ビット SPI ワード

SPI ワードは 24 ビット長で、SDI において MSB ファーストでデータをシフトする必要があります。

B23 : (R/W) 読み取りまたは書き込みビット (R = 1、W = 0)

B22 ～ B21 : (X) 未使用

B20 ～ B16 : 機能アドレス (5 ビット)

B15 : (X) 未使用

B14 ～ B12 : ポート選択 (ポート 0 = 000、ポート 1 = 001、ポート 2 = 010)

B11 ～ B9 : (X) 未使用

B8 : マルチポート ビット (マルチポートが有効化 = 1、マルチポートが無効化 = 0)

B7 ～ B0 : データ バイト

4 SPI 書き込み手順

SPI 書き込みコマンドには、指定されたレジスタへのデータの書き込みと、SDO (全二重) での以前のレジスタ データの内容の読み取りも含まれます。

1. チップ セレクト (/CS) を Low に駆動します。これにより、内部シフト レジスタが有効になります
2. 24 ビットのデータを SDI において MSB ファーストでデバイスに移行します。クロック (SCLK) の立ち上がりエッジの間、データは安定している必要があります。
3. MSB ビット (B23) は、これが書き込み動作であることを示す「0」である必要があります。
4. 16 ビットのステータスが SDO で送信されます。最初の 2 ビットは (2 ビットのバイナリ表記で)「11」です (ステータス セグメントであることを示します)。次の 6 ビット、B21 ～ B16 は、故障ステータス レジスタ のビット D5 ～ D0 です。最後の 8 ビット、B15 ～ B8 はすべて 0 です。
5. レジスタの以前のデータは SDO (B7 ～ B0) で読み取られ、データ バイトは SDI (B7 ～ B0) のレジスタに書き込まれます。
6. 最後のデータビットが転送された後、転送するデータがない場合は SCLK を low に駆動します。
7. 書き込みサイクルを終了するためにチップ セレクト (/CS) をデアサート (High に駆動) します

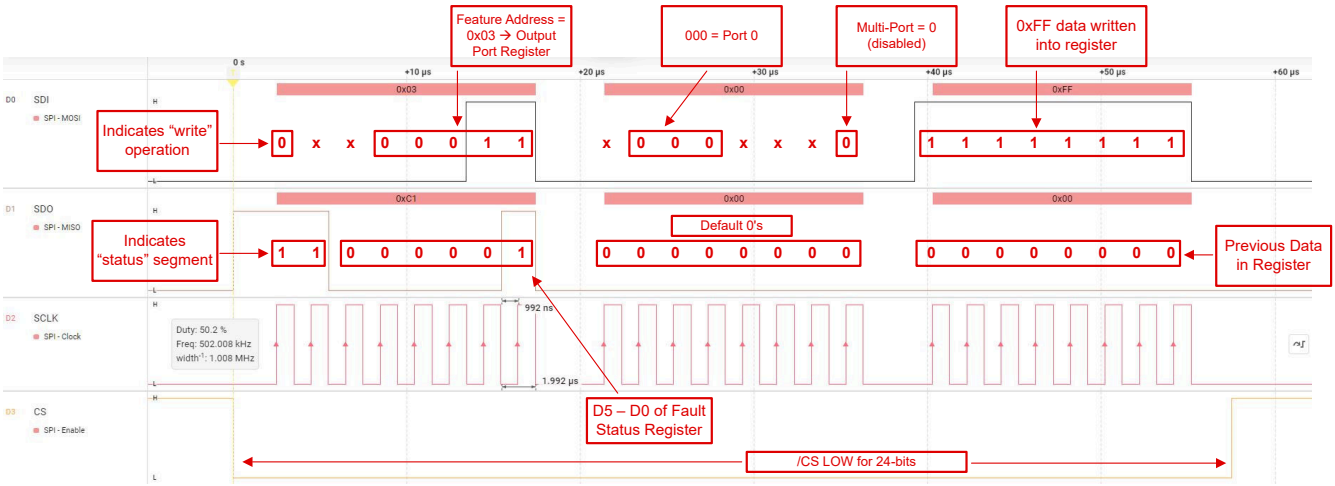

図 4-1. Saleae ロジック アナライザを使用した SPI 書き込みの例

5 コーディング例

Code Composer Studio の現行バージョンは、TI Developer Zone の Common Actions セクションの下からダウンロードできます。

Code Composer Studio IDE をダウンロードして開くと、Resource Explorer で提供されているソフトウェア サンプルを参照して、サンプル コードを見つけることができます。

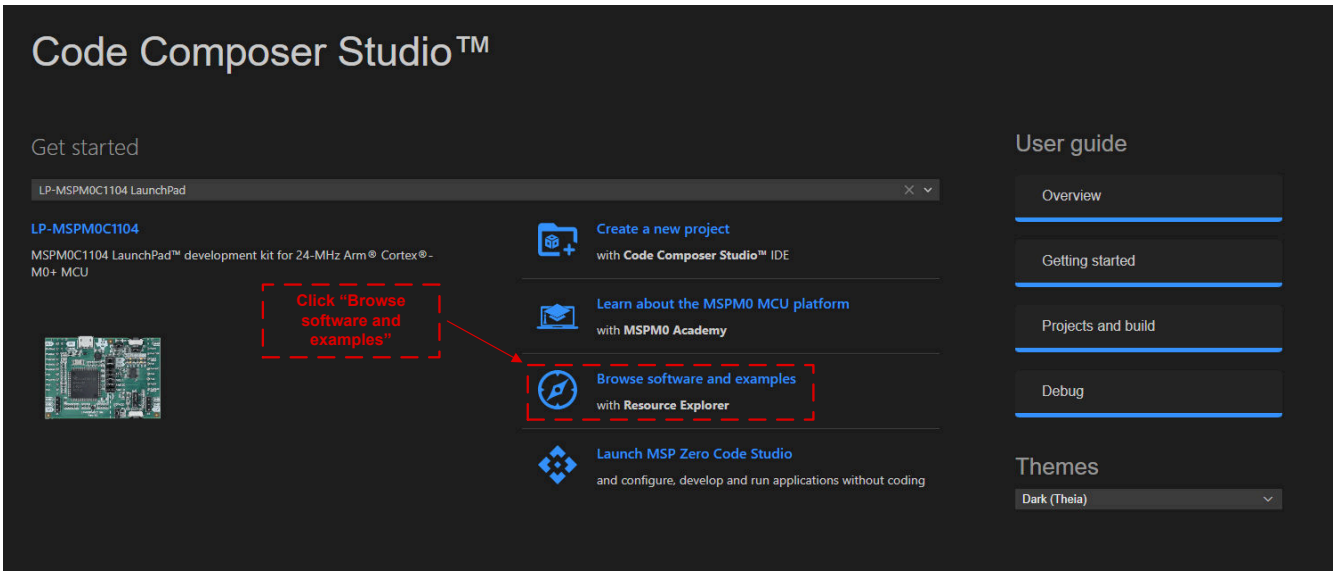

図 5-1. Code Composer Studio IDE で、ソフトウェアとサンプルを参照する

Resource Explorer を開いた後、以下のディレクトリを辿ってサンプル コードを見つけます。

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

3 つのドットをクリックし、「Import to CCS IDE」(CCS IDE にインポート) を選択して、サンプル プロジェクトを CCS IDE にインポートできます。

インポートが完了したら、最初に「spi_controller_internal_loopback_poll.syscfg」ファイルを選択します。

この例の SPI 設定は、SYSCONFIG GUI で変更する必要があります。コードを正しくセットアップするため、SPI の下で以下の構成を選択します。

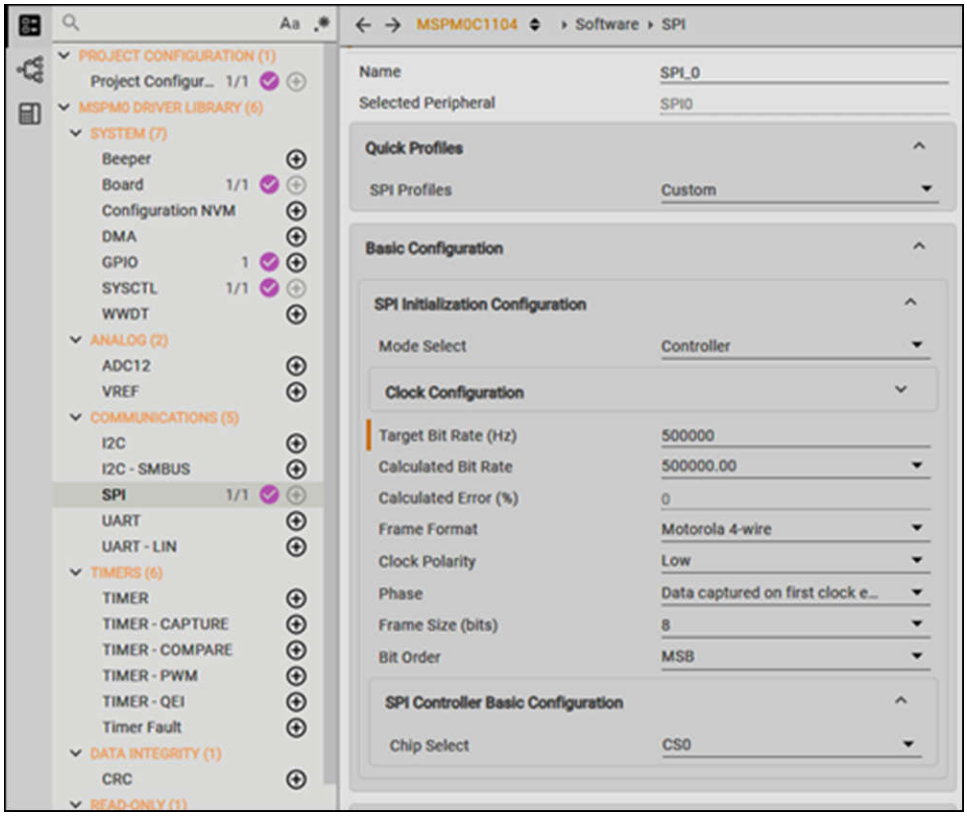

図 5-2. SYSCONFIG のセットアップ

SPI_0 は、使用されている SPI ペリフェラルの名前です。カスタム プロファイルが作成されます。モードが、ターゲット ビット レート 500kHz のコントローラとなるように選択されます。フレーム フォーマットは Motorola 4 線式 SPI です。クロック極性は Low に設定されています。クロック位相：最初のクロック エッジ (立ち上がりエッジ) でキャプチャされたデータです。フレーム サイズは 8 ビットに設定されています。ビット順序は MSB ファーストです。チップ セレクトは CS0 – PA11 ですが、この例では PA2 上の GPIO を使用して、TXE8124 デバイスの 3 バイト フレーミング構造をキャプチャします。この例では、CS0 – PA11 は無視されます。

「Advanced Configuration」(高度な構成) タブで、「Enable Internal Loopback」(内部ループバックの有効化) がオフになっていることを確認します。

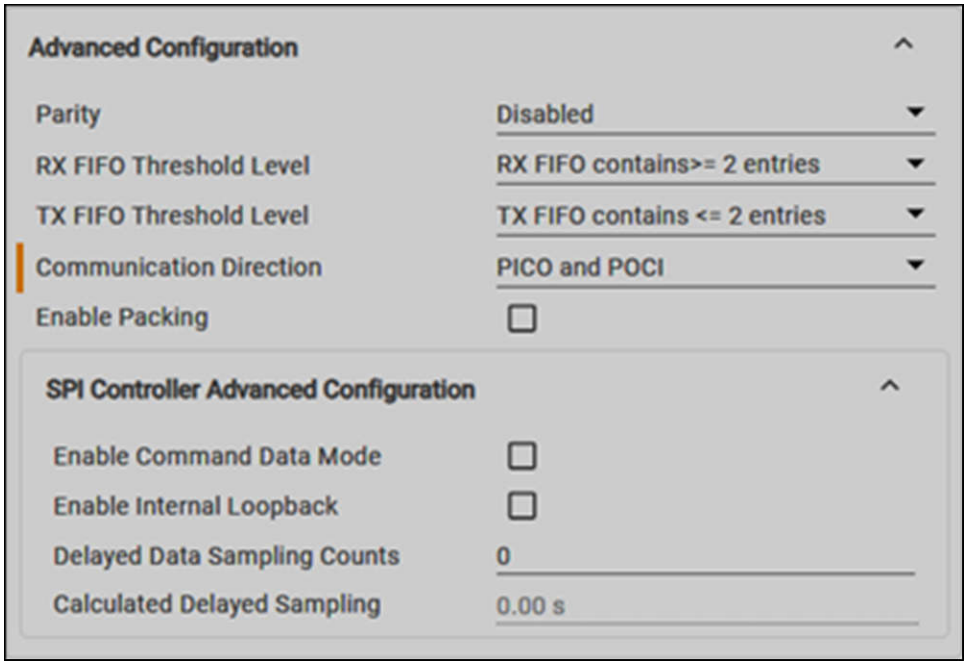

図 5-3.「Advanced Configuration」(高度な構成) タブの下にある「Enable Internal Loopback」(内部ループバックの有効化) がオフになっている

SYSCONFIG GUI は、M0 ドライバ ライブラリ内にコードを書き込む必要なしに、SPI 構成を設定できます。これにより、SPI ドライバが設定され、TXE81XX のサンプル関数に集中できるる時間が増えます。

6 サンプル コード

以下のサンプル コードは、MSPM0 SDK に含まれる spi_controller_internal_loopback_poll.c サンプル コードにコピーして使用できます。この .c ファイルは、TXE81XX の 24 ビット ワード方式で動作する SPI 書き込みおよび読み取り関数を含めるように変更されました。

```
/*SPI の接続:
VCC = 3.3V
CPOL = 0
CPHA = 0
クロック周波数 = 500kHz
SCLK = PA11
MISO = PA16
MOSI = PA18
/CS = PA2
*/
#include "ti_msp_dl_config.h" // MSP ドライバ ライブラリ
//TXE81xx D の TXE81xx レジスタ マップを定義する
#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)
#define fail_safe_redun (0x18)
#define fail_safe_fault (0x19)
#define software_reset (0x1A)
//各ポートを定義する
//TXE8116：ポート 0 / ポート 1
//TXE8124：ポート 0 / ポート 1 / ポート 2
#define port0 (0x00)
#define port1 (0x01)
#define port2 (0x02)
#define DELAY_500khz (50) //500kHz での SPI 通信のチップ セレクト ( /CS) をチューニングするための遅延サイクル
//関数の定義
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 (); //SPI ドライバを初期化する

read_val = SPI_Transfer(config_reg, port0, 0xFF, false); //構成レジスタ 0 に書き込む：すべての出力を設定する
//ポート 0 に出力を毎秒書き込む
while(1){
read_val = SPI_Transfer(output_reg, port0, 0xFF, false); //0xFF：LED がオフ
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);

read_val = SPI_Transfer(output_reg, port0, 0xAA, false); //0xAA：代替 LED
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);
read_val = SPI_Transfer(output_reg, port0, 0x55, false); //0x55：逆の状態に切り替える
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);

read_val = SPI_Transfer(output_reg, port0, 0x00, false); //0x00：LED がオン
read_val = SPI_read(output_reg, port0);
delay_cycles(24000000);
}
}
```

```
//SPI_Transfer は、8 ビット レジスタに設定された前のバイトを書き込み、返す
uint8_t SPI_Transfer(uint8_t feat_addr, uint8_t port, uint8_t data_byte, bool multi_port){
uint8_t write_byte = feat_addr & 0x1F; //B20 ～ B16 のみを保持する
uint8_t port_byte = (port << 4) & 0x70; //B14 ～ B12 のみを保持し、まずポート番号を上位ニブルにシフトする
uint8_t m_bit;
uint8_t status_byte1 = 0x00; // 最初の 2 ビットは (2 ビットのバイナリ表記で)「11」(ステータス セグメントを示す)、次の 6 ビットは故障フラグ レジスタの D5 ～ D0 ビット
uint8_t status_byte2 = 0x00; // 8 ビットはすべて 0
uint8_t reg_data_out = 0x00; //現在のレジスタ データ
if (multi_port == true){
m_bit = 0x01; //マルチポート = 1 = 有効化
} else {
m_bit = 0x00; //マルチポート = 0 = 無効化
}
DL_GPIO_clearPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN |
GPIO_LEDS_USER_TEST_PIN); //チップ セレクト (/CS) を Low に設定する
DL_SPI_transmitData8(SPI_0_INST, write_byte); //送信 B23 ～ B16
DL_SPI_transmitData8(SPI_0_INST, port_byte + m_bit); //送信 B15 ～ B8
DL_SPI_transmitData8(SPI_0_INST, data_byte); //送信 B7 ～ B0
status_byte1 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //SDO の 1 番目のバイトのデータの読み取り
status_byte2 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //SDO の 2 番目のバイトのデータの読み取り
reg_data_out = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //SDO の 3 番目のバイトのデータの読み取り:これが返されるレジスタ データ (以前にレジスタで設定されたバイト)
delay_cycles(DELAY_500khz); //チップ セレクトまでの遅延
DL_GPIO_setPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN | GPIO_LEDS_USER_TEST_PIN); //チップ セレクト (/CS) を High に設定する
return reg_data_out; //以前のレジスタ バイトを返す
}
// 故障フラグ レジスタのレジスタとステータス ビットを直接読み取る関数を作成する
uint8_t SPI_read (uint8_t feat_addr, uint8_t port){
uint8_t read_byte = (feat_addr & 0x1F) + 0x80; //B20 ～ B16 のみを保持し、「読み取り」用に B23 = 1 にする
uint8_t port_byte = (port << 4) & 0x70; //B14 ～ B12 のみを保持し、まずポート番号を上位ニブルにシフトする
uint8_t status_byte1 = 0x00; // 最初の 2 ビットは (2 ビットのバイナリ表記で)「11」(ステータス セグメントを示す)、次の 6 ビットは故障フラグ レジスタの D5 ～ D0 ビット
uint8_t status_byte2 = 0x00; // 8 ビットはすべて 0
uint8_t reg_data_out = 0x00; //現在のレジスタ データ
```

```
DL_GPIO_clearPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN |
GPIO_LEDS_USER_TEST_PIN); //チップ セレクト (CS) を Low に設定する

DL_SPI_transmitData8(SPI_0_INST, read_byte); //読み取りコマンド バイトを送信する

DL_SPI_transmitData8(SPI_0_INST, port_byte); //ポート選択を指定する

DL_SPI_transmitData8(SPI_0_INST, 0x00); //MISO の 8 ビットのレジスタ データを読み取るための 0x00 ダミー データ

status_byte1 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //ステータスの最初の 8 ビット

status_byte2 = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //ステータスの 2 番目の 8 ビット

reg_data_out = DL_SPI_receiveDataBlocking8 (SPI_0_INST); //レジスタ データ バイト

delay_cycles(DELAY_500khz); //チップ セレクトまでの遅延

DL_GPIO_setPins(GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN | GPIO_LEDS_USER_TEST_PIN); //チップ セレクト (CS) を High に設定する

return reg_data_out; //レジスタ バイトを返す

}
```

7 まとめ

SPI – GPIO エクスパンダの TXE ファミリは、24 ビットのワード形式を使用しています。TXE デバイスへの書き込みは全二重動作であり、SDI および SDO 上のレジスタからデータをそれぞれ読み取りおよび書き込みできます。TXE を直接読み取ると、SDO のレジスタからデータが出力されます。提供されるサンプル コードは、TXE に対する単純な読み取りまたは書き込みコマンドのための最適化されていない設計で、IC の動作方法に関する一般的な書式を示します。

8 参考資料

- テキサス インスツルメンツ、MSPM0 SDK、Resource Explorer。

重要なお知らせと免責事項

テキサス・インスツルメンツは、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、テキサス・インスツルメンツ製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した テキサス・インスツルメンツ製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、テキサス・インスツルメンツはその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。テキサス・インスツルメンツや第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、テキサス・インスツルメンツおよびその代理人を完全に補償するものとし、テキサス・インスツルメンツは一切の責任を拒否します。

テキサス・インスツルメンツの製品は、テキサス・インスツルメンツの販売条件、または ti.com やかかる テキサス・インスツルメンツ製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。テキサス・インスツルメンツがこれらのリソースを提供することは、適用されるテキサス・インスツルメンツの保証または他の保証の放棄の拡大や変更を意味するものではありません。

お客様がいかなる追加条項または代替条項を提案した場合でも、テキサス・インスツルメンツはそれらに異議を唱え、拒否します。

郵送先住所:Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月