Subsystem Design

具有 SPI、I2C 和 UART 的 IO 扩展器

1 说明

备注

TI 正在过渡到使用更具包容性的术语。本文档中包含的一些语言可能与之前用于某些技术领域的术语不同。

此子系统演示了如何使用 MSPM0 通过主机从串行外设接口 (SPI)、I2C 或通用异步接收器-发送器 (UART) 发出的通信命令来实现 IO 扩展器功能。当主机上的 GPIO 数量不足时，此扩展很有帮助。除了支持控制 GPIO 输出之外，子系统还可以通过 SPI、I2C 或 UART 回读 GPIO 状态。下图 展示了此例中使用的子系统和模块的基本架构。

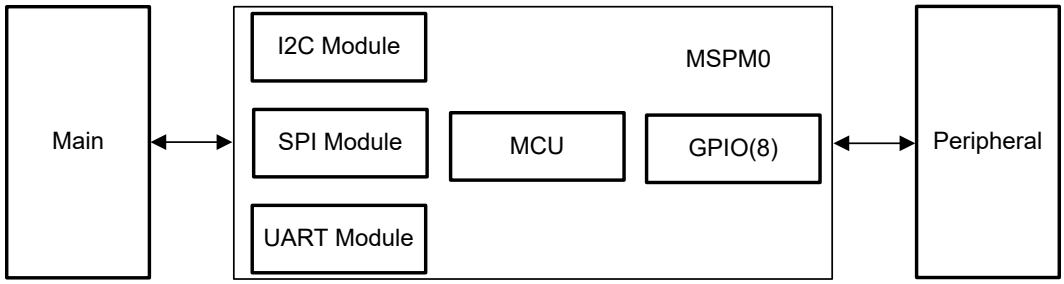

图 1-1. 子系统功能方框图

表 1-2 具有指向示例代码的链接。在此演示中，IO 控制数限制为 8。但是，用户可以通过参考此演示来进一步扩展 IO。

2 所需外设

表 2-1 展示了此应用中所需的外设和功能模块。

表 2-1. 所需外设

子块功能	外设使用	注释
串行外设接口	(1 个) SPI	在代码中称为 *SPI_0_INST*
I2C 接口	(1 个) I2C	在代码中称为 *I2C0_INST*
UART 接口	(1 个) UART	在代码中称为 *UART_0_INST*
GPIO	(8 个) GPIO	在代码中称为 GPIO_GRP_0

3 硬件设置

若要基于 MSPM0 来评估 IO 扩展器，需要以下硬件元件：

- “所需器件”中所示的 MSPM0 LaunchPad™ 开发套件
- 安装有 Microsoft® Windows® 7 或更高版本和 .NET Framework 4.5 的计算机。(或作为主器件的实际器件)
- USB2ANY 和兼容 GUI

在此子系统中，客户可灵活选择不同的通信接口，包括 I2C、SPI 或 UART。这可充分提高客户系统设计的灵活性。图 3-1 以 LP-MSPM0C1104 为例展示了此设计中的硬件连接。

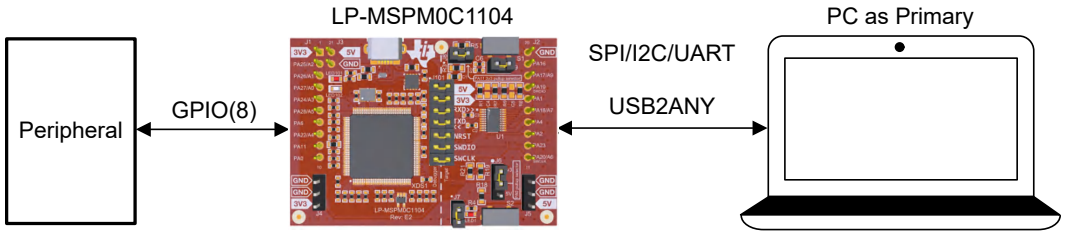

图 3-1. 硬件连接

表 3-1 展示了引脚配置，您还可以根据需要更改配置。SPI 通信配置为三线模式以节省 GPIO 资源。MSPM0L1306 和 MSPM0C1104 的引脚配置相同。

表 3-1. 引脚配置

模块	功能	引脚配置	注释
I2C 接口	SDA	PA0	地址：0x48，I2C 时钟频率：400kHz
	SDL	PA1	
串行外设接口	POSI	PA25	SPI 时钟频率：500kHz
	PISO	PA26	
	时钟	PA17	
UART 接口	RX	PA18	波特率：9600bps
	TX	PA23	
GPIO	GPIO	BIT0:PA2、BIT1:PA27、 BIT2:PA17、BIT3:PA24、 BIT4:PA4、BIT5:PA6、 BIT6:PA16、BIT7:PA22	

4 软件简介

图 4-1 展示了在 CCS 中开发的软件工程。该工程主要由三个部分组成。其他文件是 MSPM0 工程的默认文件。

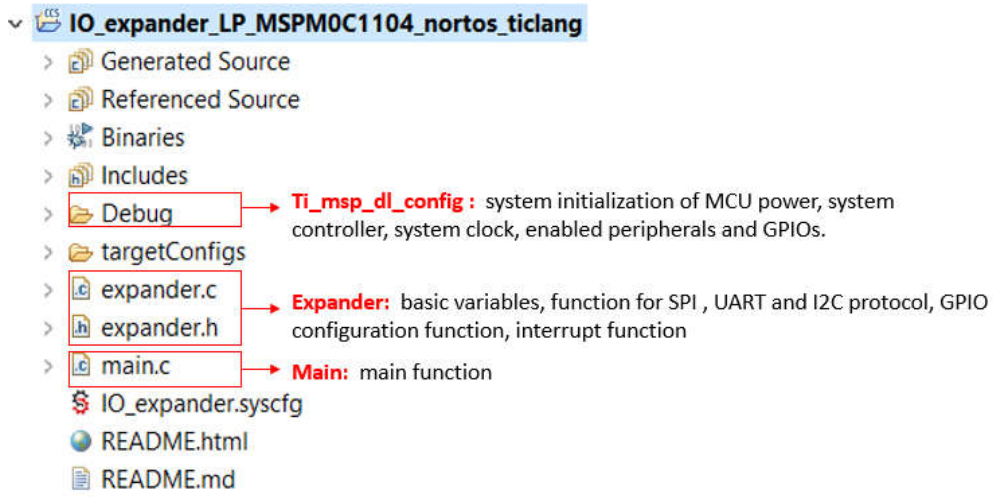

图 4-1. 软件工程视图

ti_msp_dl_config 部分由 SysConfig (图形代码生成工具) 生成，而 MSPM0 初始化则用于 MCU 电源、系统控制器、系统时钟、启用的外设和 GPIO 的系统初始化。

expander 部分声明基本变量、GPIO 配置函数和中断函数，*expander* 还包含一些适用于 SPI、I2C 和 UART 协议的基本函数。

main 部分包含最高系统功能代码，在系统初始化之后，MCU 等待来自主器件的命令并执行相应的 GPIO 操作。

4.1 代码简介

图 4-2 展示了此设计中的主函数代码。主函数初始化系统配置，然后进入循环以处理 IO 控制。代码支持用于 IO 控制的三个函数：gpioDirectionSet、gpioOutputCtl 和 gpioStateRead。另请参阅*协议简介*。

```
#include <expander.h>
#include "ti_msp_dl_config.h"

tDataType ioExpander;

int main(void)
{
    SYSCFG_DL_init();
    systemInit();

    while (1)
    {
        if(ioExpander.pktTyp != eNoPkt)
        {
            switch(ioExpander.pktTyp)
            {
            case eGpioDir:
                gpioDirectionSet();
                break;
            case eGpioOtp:
                gpioOutputCtl();
                break;
            case eGpioInp:
                gpioStateRead();
                break;
            default:
                break;
            }

            ioExpander.pktTyp = eNoPkt;
        }

    }
}
```

图 4-2. 主函数

默认情况下启用 UART、I2C 和 SPI。对于实际实现，取消注释 expander.h 文件中的定义，如 图 4-3 所示。

expander.h

```
31 */
32
33 #include <stdint.h>
34 #ifndef EXPENDER_H_
35 #define EXPENDER_H_
36
37 #define UART
38 #define I2C
39 #define SPI
40
```

图 4-3. 通信启用和禁用

所有通信命令接收都在相对中断中完成。使用 UART，命令传输在 gpioStateRead() 函数中完成。对于 SPI 和 I2C，命令传输在相对中断中完成。

4.2 协议简介

本节介绍此演示的支持命令。gpioDirectionSet 函数用于启用或禁用 GPIO 输出。因为 MSPM0 可以同时启用 GPIO 输入和输出，所以 GPIO 输入始终是启用的。然后，方向字节中的每个位都用于启用或禁用 GPIO 输出。*1* 表示启用 GPIO 输出和输入，而 *0* 表示仅启用 GPIO 输入。有关受控位和 GPIO 之间的关系，请参阅 表 3-1。

表 4-1. GPIO 输出启用命令

类型	接头	命令	方向	校验和
gpioDirectionSet	0x5A	0x01	1 个字节（1：OUT；0：IN）	1 字节

gpioOutputCtl 函数用于控制 GPIO 输出。然后，输出控制字节中的每个位都用于设置 GPIO 输出高电平或输出低电平。*1* 表示输出高电平，而 *0* 表示输出低电平。请记住，此函数仅在主器件发送 gpioDirectionSet 函数之后有效。有关受控位和 GPIO 之间的关系，请参阅 表 3-1。

表 4-2. GPIO 输出控制命令

类型	接头	命令	输出控制	校验和
gpioOutputCtl	0x5A	0x02	1 个字节（1：高电平；0：低电平）	1 字节

gpioStateRead 函数用于读取 GPIO 状态。主器件需要将此命令发送到辅助器件，然后辅助器件将 GPIO 状态发送回主器件。然后，辅助器件所发送的引脚状态字节中的每个位都用于显示 GPIO 状态。*1* 表示 GPIO 状态为高电平，而 *0* 表示 GPIO 状态为低电平。此命令还可用于在主器件发送 gpioOutputCtl 命令之后检查 GPIO 控制是否符合预期。有关受控位和 GPIO 之间的关系，请参阅 表 3-1。

表 4-3. GPIO 状态读取命令

类型	接头	命令	引脚状态	校验和
gpioStateRead	0x5A	0x03	1 个字节（1：高电平；0：低电平）	1 字节

图 4-4 展示了主器件和辅助器件上的命令发送条件，另请参阅 节 6。

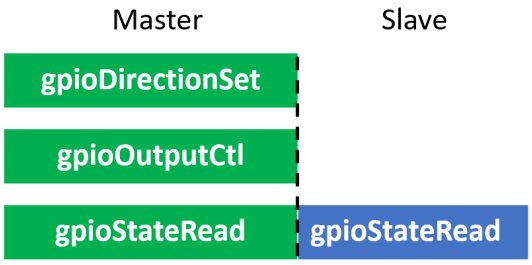

图 4-4. 命令发送条件

5 设计步骤

请完成以下步骤来评估该设计：

1. **准备必要资源：**

   请按照本文档前面讨论的内容准备相关的硬件资源和软件代码。使用真实主机系统将 PC 替换为 USB2ANY GUI。请注意 MSPM0C1104 的 I2C 设置，因为 PA1 重复用作复位引脚。对于软件，禁用 SysConfig 中的复位功能，对于硬件，移除 J9 连接器以避免下拉电容器影响 I2C 通信。使用额外的上拉电阻器。
2. **选择通信接口：**

   通过在 expander.h 中取消不需要的通信功能宏的注释来选择合适的通信接口（UART、SPI 或 I2C），如 图 4-3 中所示。现在，初始化通信接口并在 io_expander.syscfg（如果使用 UART）中设置通信参数，诸如波特率、数据位和停止位。

   确保正确调用 transmitPacket 和 receivePacket 函数，以在通信期间发送和接收数据包，并验证校验和以确保通信正确。
3. **开始使用：**

   通过引用 节 4.2 来写入执行 gpioDirectionSet、gpioOutputCtl 和 gpioStateRead 命令的函数。从主机向扩展模块发送相关命令，然后可以验证 GPIO 方向设置、输出值和输入值是否符合预期。

6 结果

基于 I2C 的协议用法： 图 6-1 展示了四种不同命令的用法，包括 GPIO 输出启用命令、GPIO 输出控制命令、GPIO 状态读取请求命令和 GPIO 状态读取响应命令。请注意，回读 GPIO 状态与 GPIO 输出控制命令中的稳定 GPIO 状态相同。

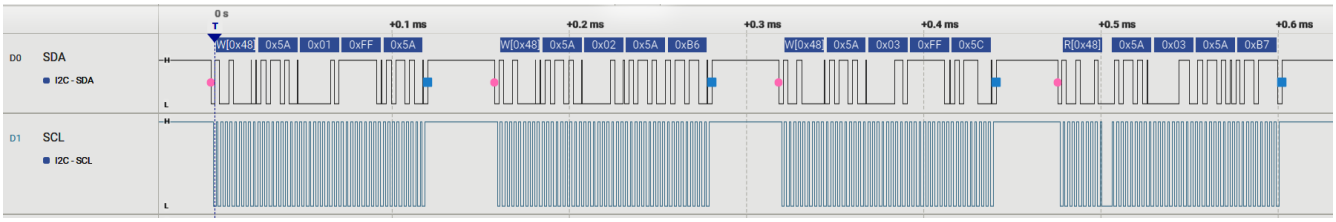

图 6-1. I2C 通信 1

图 6-2 展示了三种不同命令的用法，包括 GPIO 输出启用命令、GPIO 状态读取请求命令和 GPIO 状态读取响应命令。在此测试中，所有 GPIO 都稳定为输入状态。

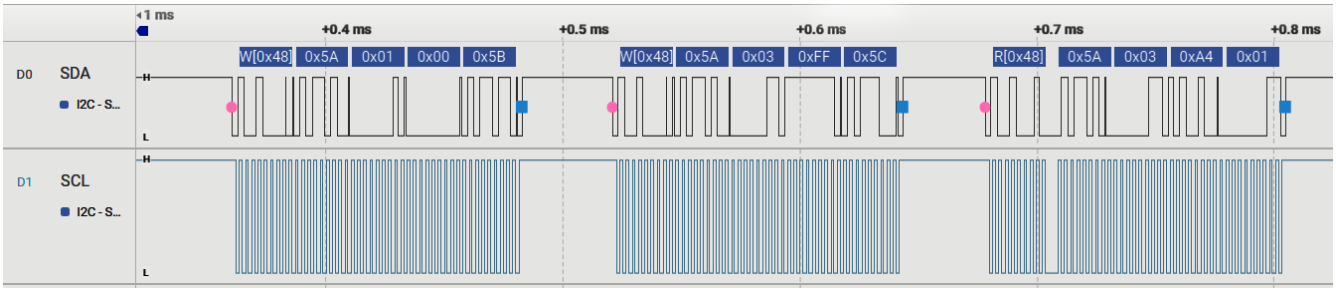

图 6-2. I2C 通信 2

基于 UART 的协议用法： 图 6-1 展示了四种不同命令的用法，包括 GPIO 输出启用命令、GPIO 输出控制命令、GPIO 状态读取请求命令和 GPIO 状态读取响应命令。

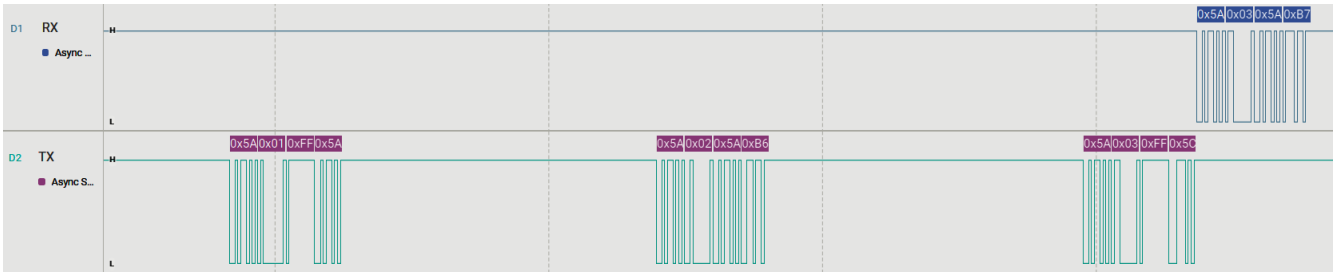

图 6-3. UART 通信

基于 SPI 的协议用法： 图 6-1 展示了四种不同命令的用法，包括 GPIO 输出启用命令、GPIO 输出控制命令、GPIO 状态读取请求命令和 GPIO 状态读取响应命令。

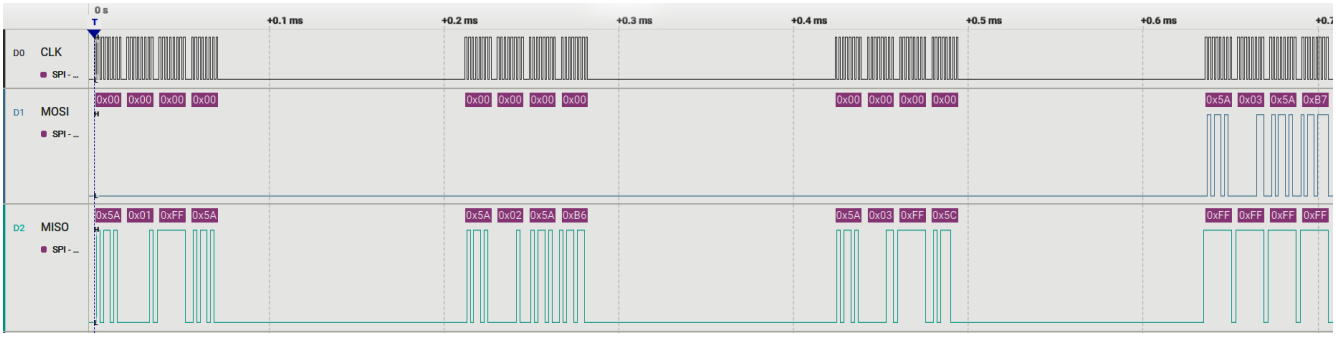

图 6-4. SPI 通信

7 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (October 2024) to Revision A (August 2025)	Page
• 删除了“兼容器件”部分	1

8 商标

LaunchPad™ is a trademark of Texas Instruments.
Microsoft® and Windows® are registered trademarks of Microsoft Corporation.
所有商标均为其各自所有者的财产。

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