Subsystem Design

I2C IO 扩展器

1 说明

该子系统示例演示了如何使用 MSPM0 及控制器配置 IO 扩展器。配置过程中会设置 PIN 方向、状态和读取状态。

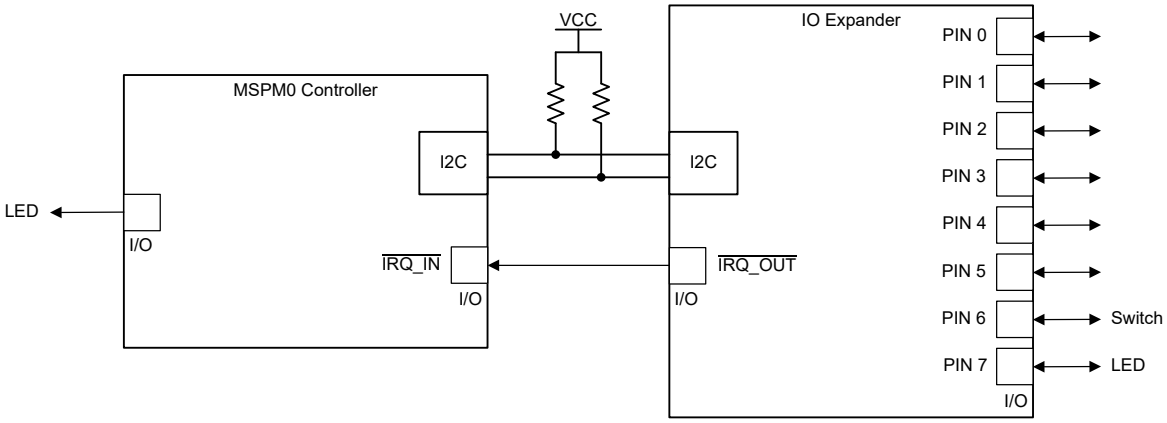

图 1-1. 子系统功能方框图

2 所需外设

表 2-1 和表 2-2 介绍了*所需*的集成外设。

表 2-1. 控制器所需的外设

子块功能	外设使用	注释
LED 及中断引脚	(2 个) GPIO	在代码中显示为 *LED* 和 *IRQ_IN*
I2C 控制器	(1 个) I2C	在代码中显示为 *I2C*

表 2-2. IO 扩展器所需的外设

子块功能	外设使用	注释
GPIO 及中断引脚	(9 个) GPIO	在代码中显示为 *PIN_0*、*PIN_1*、*PIN_2*、*PIN_3*、*PIN_4*、*PIN_5*、*PIN_6*、*PIN_7*, 和 *IRQ_OUT*
I2C 目标方	(1 个) I2C	在代码中显示为 *I2C*

3 设计步骤

1. 在 SysConfig 中配置控制器及外设 I2C 实例、GPIO 引脚、GPIO 开关和 GPIO LED。
2. 在 SysConfig 中设置 I2C 时钟速度。具有外部上拉电阻的 LaunchPad™ 开发套件默认为 400kHz。
3. 定义正确通信所需要的 I2C 数据包。
4. 创建一个切换 IO 扩展器输出及开关输入 (用于切换控制器上的 LED) 的演示。

4 设计注意事项

1. **检测引脚状态更改**：如果输入引脚改变状态，IO 扩展器利用中断来通知控制器请求读取。只要 GPIO 引脚观察到上升沿或下降沿，IO 扩展器也会更新本地保存的引脚状态。
2. **更改 GPIO 方向**：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。设置方向字节以启用或禁用 GPIO 输出。每个字节内的位位置决定输出控制。这使得所有引脚都可以使用上升沿和下降沿中断并能够读取所有引脚状态。

5 软件流程图

图 5-1 显示了 IO 扩展器的主函数代码。主函数初始化外设，然后进入循环以处理收到的 I2C 通信。

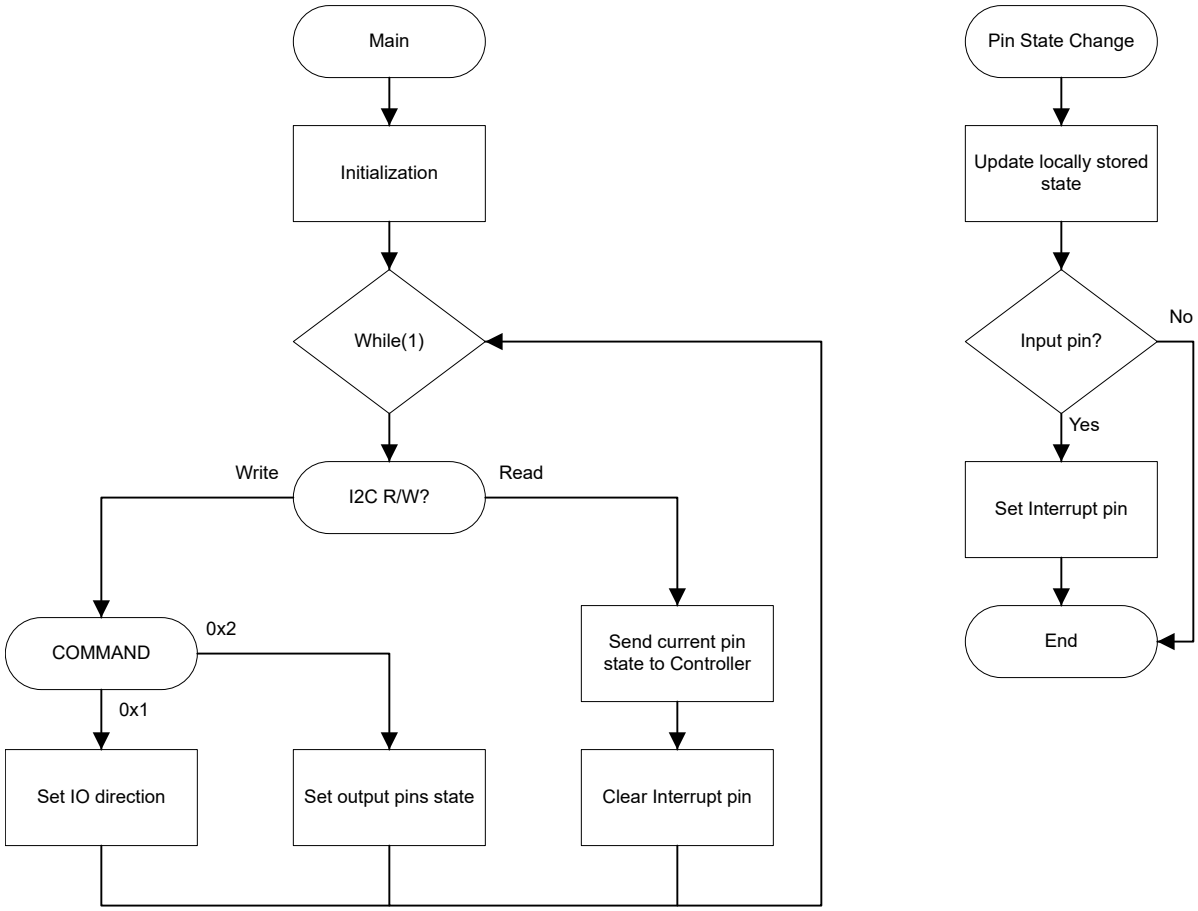

图 5-1. IO 扩展器软件流程图

图 5-2 展示了控制器中的主函数代码。主函数初始化外设，发送 I2C 命令以设置 IO 方向，并进入切换 IO 扩展器输出引脚的循环。

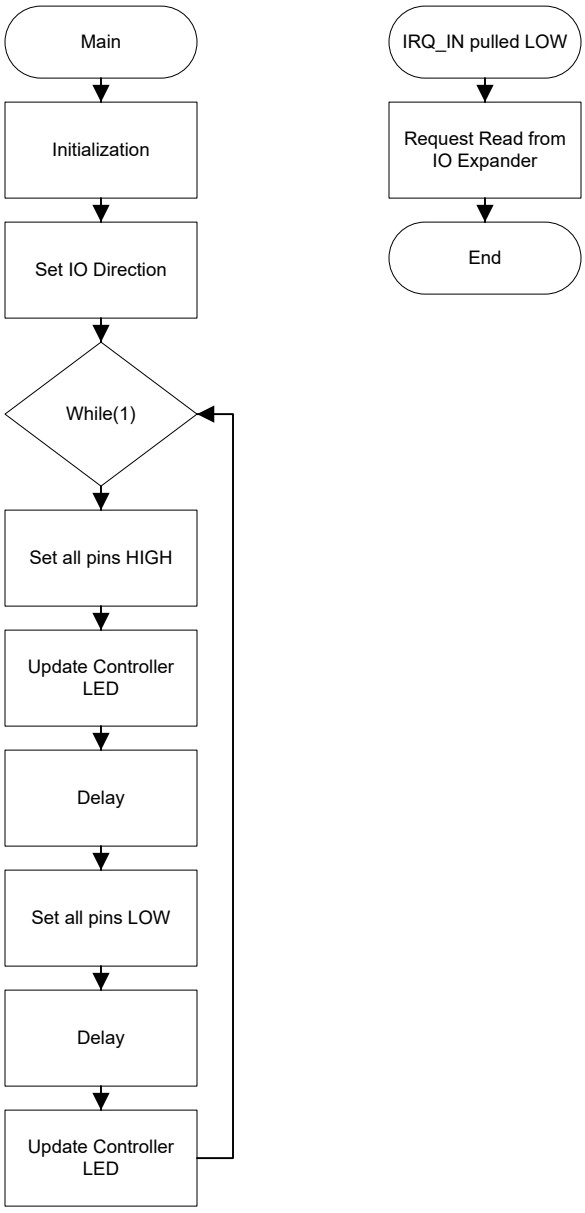

图 5-2. 控制器软件流程图

6 所需的 I2C 数据包

图 6-1 和图 6-2 展示了使用 I2C 接口正确通信所需的 I2C 数据包。

- **COMMAND**：包含用于更改 IO 方向或设置输出引脚状态的命令的字节。
  - **0x1**：设置 GPIO 方向
  - **0x2**：设置输出引脚状态
- **DATA**：包含引脚配置的字节
  - **COMMAND = 0x1**：1 表示输出，0 表示输入
  - **COMMAND = 0x2**：1 表示输出高电平，0 表示输出低电平

表 6-1. I2C 数据包分解

功能	命令	DATA
设置 GPIO 方向	0x1	1：输出
		0：输入
设置输出引脚状态	0x2	1：高
		0：低

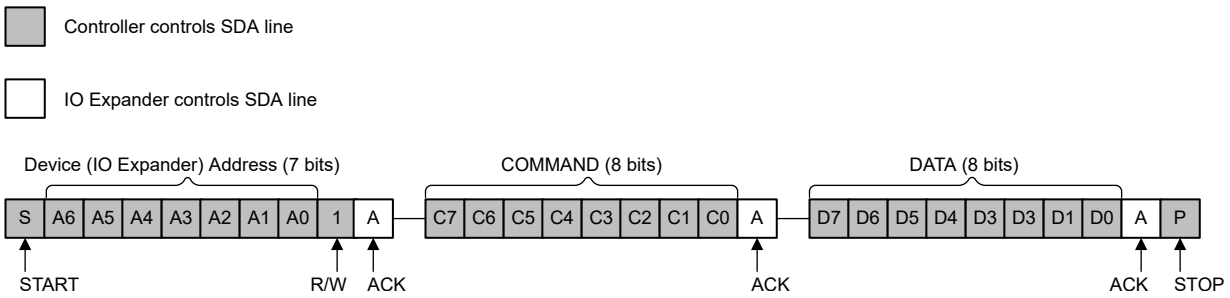

图 6-1. I2C 写入数据包

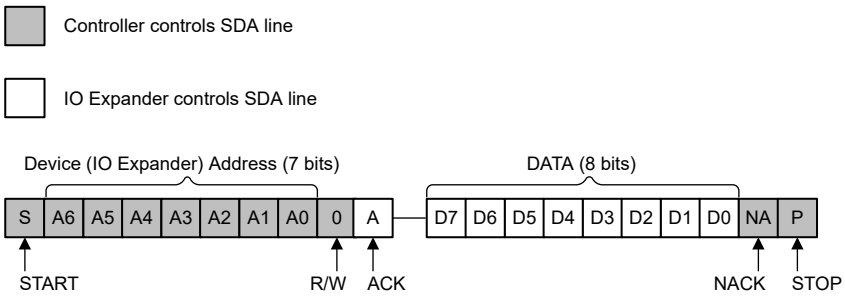

图 6-2. I2C 读取数据包

7 应用代码

只要引脚观察到上升沿或下降沿，代码示例就会更新存储的 GPIO 状态。IRQ_OUT 引脚仅在输入引脚观察到上升沿或下降沿时才会被拉低，并且一旦控制器请求读取，就会被拉高。TX FIFO 在接收到一个 I2C 启动条件时被填充，并在接收到 I2C 停止条件时被清除：

```
//I2C INT
void I2C_INST_IRQHandler(void)
{
    switch (DL_I2C_getPendingInterrupt(I2C_INST)) {
        case DL_I2C_IIDX_TARGET_START:
            /* Fill TX FIFO with current pin state */
            DL_I2C_fillTargetTXFIFO(I2C_INST, &gGpioState, 1);
            break;
        case DL_I2C_IIDX_TARGET_RXFIFO_TRIGGER:
            /* Store received data in buffer */
            while (DL_I2C_isTargetRXFIFOEmpty(I2C_INST) != true) {
                receivePacket(DL_I2C_receiveTargetData(I2C_INST));
            }
            break;
        case DL_I2C_IIDX_TARGET_TX_DONE:
            /* Pull interrupt pin high when Controller reads from IO Expander */
            DL_GPIO_setPins(GPIO_GRP_0_PORT, GPIO_GRP_0_IRQ_OUT_PIN);
            break;
        case DL_I2C_IIDX_TARGET_STOP:
            /* Flush TX FIFO */
            DL_I2C_flushTargetTXFIFO(I2C_INST);
            break;
        default:
            break;
    }
```

```
}

void GPIOA_IRQHandler(void)
{
    /* Store the current pin state */
    gGpioState = (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_7_PIN) << 7) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_6_PIN) << 6) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_5_PIN) << 5) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_4_PIN) << 4) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_3_PIN) << 3) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_2_PIN) << 2) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_1_PIN) << 1) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_0_PIN));

    /* Loop through all pins */
    for (int i = 0; i < 8; i++) {
        /* Check if the current pin state changed */
        if (((gGpioState >> i) & 0x1) != ((gLastGpioState >> i) & 0x1)) {
            /* If the pin is an Input */
            if (((gGpioDirection >> i) & 0x1) == 0) {
                /* Pull interrupt pin LOW when an input pin state changes */
                DL_GPIO_clearPins(GPIO_GRP_0_PORT, GPIO_GRP_0_IRQ_OUT_PIN);
            }
        }
    }
    gLastGpioState = gGpioState;
}
```

控制器 IRQ_IN 中断在检测到下降沿时触发，在该情况下会请求读取 IO 扩展器：

```
void GPIOA_IRQHandler(void)
{
    /* Set LED HIGH */
    DL_GPIO_clearPins(GROUP0_PORT, GROUP0_LED_PIN);

    /* Request a read from the IO Expander */
    gRxLen = I2C_RX_PACKET_SIZE;
    gRxCount = 0;

    /* Wait until the I2C bus is idle */
    while (!(DL_I2C_getControllerStatus(I2C_INST) & DL_I2C_CONTROLLER_STATUS_IDLE));

    /* Start a read operation */
    gI2cControllerStatus = I2C_STATUS_RX_STARTED;
    DL_I2C_startControllerTransfer(I2C_INST, I2C_IO_EXPANDER_ADDRESS,
                                   DL_I2C_CONTROLLER_DIRECTION_RX, gRxLen);
    /* Enable RX FIFO trigger interrupt */
    DL_I2C_enableInterrupt(I2C_INST, DL_I2C_INTERRUPT_CONTROLLER_RXFIFO_TRIGGER);
}
```

8 其他资源

- 德州仪器 (TI)，下载 MSPM0 SDK
- 德州仪器 (TI)，详细了解 SysConfig
- 德州仪器 (TI)，MSPM0C LaunchPad™ 开发套件
- 德州仪器 (TI)，MSPM0L LaunchPad™ 开发套件
- 德州仪器 (TI)，MSPM0G LaunchPad™ 开发套件
- 德州仪器 (TI)，MSPM0 Academy

9 E2E

请访问 TI E2E™ 支持论坛查看讨论并发布新主题，以获得在设计中使用 MSPM0 器件的技术支持。

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。

最后更新日期：2025 年 10 月