*Subsystem Design*

# *USB 转 SPI 桥接器*

## 1 说明

本子系统演示如何将 MSPM0 器件实现为 USB 通信设备类 - 抽象控制模型 (USB CDC-ACM) 到串行外设接口（SPI）的桥接器。当连接至 USB 主机时，该器件会枚举为虚拟 COM 端口。在 Windows®、macOS® 或 Linux® 上，无需安装自定义驱动程序。

该桥接器将 USB 主机数据作为写事务转发到 SPI 总线，并将 SPI 从设备的读取数据返回给 USB 主机。由于 SPI 是全双工总线，桥接器在读事务期间会在 MOSI 上发送虚拟字节以产生 SPI 时钟，同时从 MISO 上接收数据。SPI 从器件通过专用的 GPIO 引脚（数据就绪信号）指示数据可用，该信号触发 SPI 读事务。此示例支持两个目标平台：MSPM0G5187（LaunchPad™、LQFP-64）和 MSPM0C5116（LaunchPad、LQFP-48）。这两个平台运行相同的桥接应用代码，但具有针对各自器件的 SysConfig 外设分配。

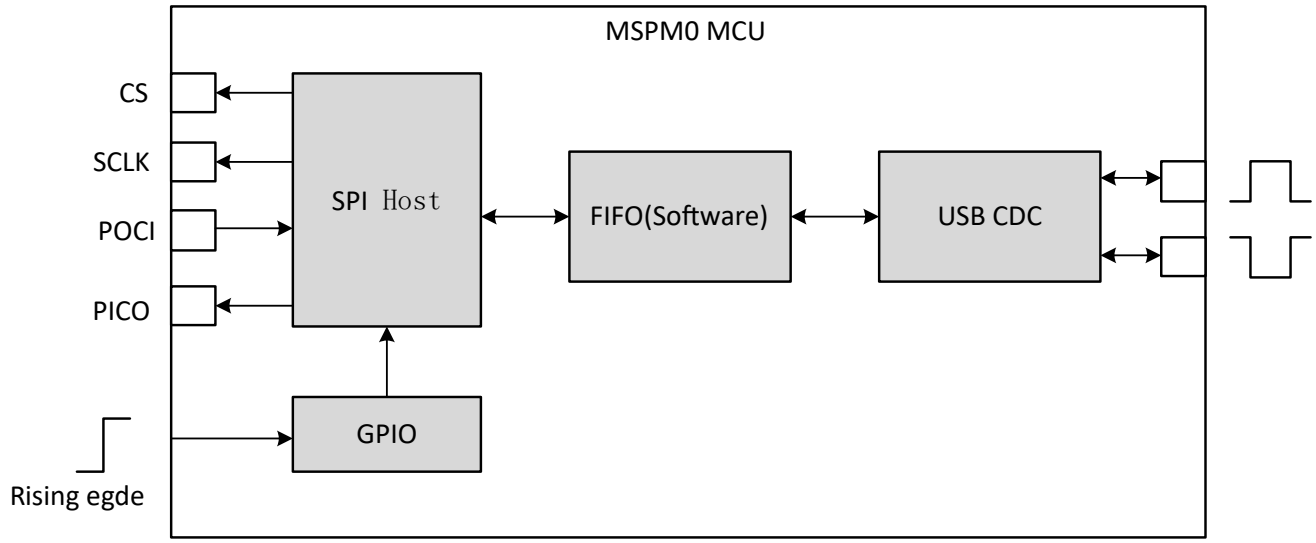


**图 1-1. 系统功能方框图**

## 2 所需外设

两个平台使用相同的外设类型。SysConfig 会根据具体器件分配对应的实例，但会生成一致的宏名称（SPI_0_INST、DMA_CH0_CHAN_ID、DMA_CH1_CHAN_ID），因此应用代码无需修改即可在两个器件上直接编译。

**表 2-1. 外设信息**

| 外设 | 功能 | MSPM0G5187 | MSPM0C5116 |
|---|---|---|---|
| USB | USB 全速 CDC 器件 | USBFS0 (USB_0) | USBLC0 (USB_0) |
| SPI | SPI 控制器至目标设备 | UC2 (SPI_0_INST) | UC18 (SPI_0_INST) |
| DMA (CH1) | SPI TX 加速 | DMA_CH1 | DMA_CH1 |
| DMA (CH0) | SPI RX 加速 | DMA_CH0 | DMA_CH0 |
| GPIO | 目标数据就绪通知输入 | PB25 | PB24 |

## 3 兼容器件

- MSPM0G5187
- MSPM0G11x
- MSPM0C511x

## 4 设计步骤

1. 在 SysConfig 中设置 TinyUSB 模块。添加一个 CDC 实例 (CDC_0)。SysConfig 会自动分配三个端点：通知 IN (EP_IN2)、数据 IN (EP_IN1) 和数据 OUT (EP_OUT1)。
2. 配置 USB 时钟树。两个平台使用不同的时钟路径。在 MSPM0G5187 上：使能 PLL 和 USB FLL，将 UDIV 设置为 2，并选择 HSCLKMUX_SYSPLL0。在 MSPM0C5116 上：仅使能 USB FLL（无需 PLL），将 UDIV 设置为 2，并选择 HSCLKMUX_USBFLL。
3. 在 SysConfig 中设置 SPI 模块。
   a. 使能 SPI 控制器模式。
   b. 设置比特率：MSPM0G5187 默认 8MHz，MSPM0C5116 默认 4MHz。
   c. 将 TX 和 RX FIFO 阈值均设置为 DL_SPI_RX/TX_FIFO_LEVEL_ONE_FRAME。
   d. 将 DMAEvent1 配置为 DL_SPI_DMA_INTERRUPT_RX，并分配 DMA_CH0，工作模式为*固定地址到块地址*。
   e. 将 DMAEvent2 配置为 DL_SPI_DMA_INTERRUPT_TX，并分配 DMA_CH1，工作模式为*块地址到固定地址*。
   f. 在中断配置选项卡中，使能 DMA_DONE_TX 和 DMA_DONE_RX 中断。
4. 在 SysConfig 中设置 GPIO 模块。
   a. 添加一个名为 SPI_SIGNAL 的输入 GPIO 引脚，配置为上升沿中断并使能下拉电阻。此引脚连接到 SPI 目标器件的数据就绪输出。
   b. 使能该输入中断。在 MSPM0G5187 上，将该 GPIO 映射到中断组 1。

```
void GROUP1_IRQHandler(void)
{
    switch (DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1)) {
        case GPIO_INT_IIDX:
            switch (DL_GPIO_getPendingInterrupt(GPIO_PORT)) {
                case GPIO_SPI_SIGNAL_IIDX:
                    // Slave signaled data ready - set flag for main loop
                    SPISlaveDataReady = true;
                    break;

                default:
                    break;
            }
            break;

        default:
            break;
    }
}
```

在 MSPM0C5116 上，直接使用 GPIOB 中断。

```
void GPIOB_IRQHandler(void)
{
    switch (DL_GPIO_getPendingInterrupt(GPIO_PORT)) {
        case GPIO_SPI_SIGNAL_IIDX:
            // Slave signaled data ready - set flag for main loop
            SPISlaveDataReady = true;
            break;

        default:
            break;
    }
}
```

c. 在 SysConfig 中设置 SysTick。使能 SysTick 并设置与 CPU 频率匹配的周期：MSPM0G5187 (80MHz) 设为 80000 个周期，MSPM0C5116 (48MHz) 设为 48000 个周期。这两个设置值均可产生 1ms 的周期。TinyUSB 的 board_millis() 需要此 1ms 周期。

# 5 设计注意事项

1. *缓冲区大小设置：*USB_FRAME_PACKAGE 默认值为 64 字节，与 USB 全速模式的最大数据包大小相匹配。每个 FIFO 队列最多可容纳 8 个任务 (USB_TX_BUF_SIZE / SPI_TX_BUF_SIZE)，每个方向对应 512 字节。增加队列深度可以处理突发流量，但会占用更多 RAM 资源。

```
// ===== Queue Configuration =====
#define USB_TX_BUF_SIZE     8               // Max tasks in SPI→USB queue
#define SPI_TX_BUF_SIZE     8               // Max tasks in USB→SPI queue

// ===== USB/SPI Parameters =====
#define USB_FRAME_PACKAGE   64              // USB Full Speed max packet size (bytes)
```

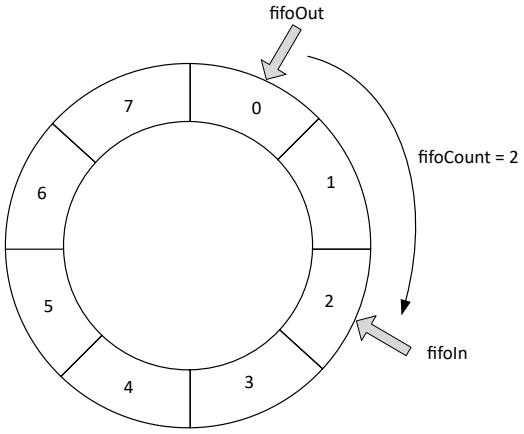


**图 5-1. 缓冲区结构**

```
// ===== Single Task Structure =====
// Represents one 64-byte unit of data (one USB frame)
typedef struct {
    uint8_t buffer[USB_FRAME_PACKAGE];   // Data payload (up to 64 bytes)
    uint16_t length;                     // Actual data length in buffer
} TxElement_t;

// ===== FIFO Queue Structure =====
// Circular queue to manage multiple tasks
typedef struct {
    uint16_t fifoIn;                     // Write pointer (next task to add)
    uint16_t fifoOut;                    // Read pointer (next task to process)
    uint16_t fifoCount;                  // Number of pending tasks in queue
    TxElement_t *fifoPointer;            // Pointer to task array
} TxFIFO_t;
```

2. *GPIO 数据就绪信号。*当 SPI 目标设备有数据准备就绪供控制器读取时，它会拉高专用 GPIO 输出（高电平有效）。桥接器通过 GPIO 中断检测上升沿，并设置 SPISlaveDataReady 标志。主循环轮询该标志，并调用 spi2usb_pushTask() 启动 SPI 读事务。目标器件必须保持 GPIO 为高电平，直到控制器完成传输。GPIO 引脚使用内部下拉电阻，因此当目标设备无数据时，线路空闲为低电平。
3. *全双工 SPI 读取：*SPI 是同步全双工总线。要从目标设备接收数据，控制器必须同时发送字节以产生 SPI 时钟。桥接器将接收任务缓冲区的内容作为虚拟 PICO 数据发送（它不是有效载荷）。目标设备忽略虚拟字节；实际接收的数据出现在 POCI 上，并由 DMA_CH0 捕获。在 SPI 读取期间，TX DMA 中断 (DMA_DONE_TX) 被禁用，因此只有 DMA_DONE_RX 标记读事务的完成。
4. *SPI TX 的 DMA：*DMA 通道 CH1 运行在*块地址到固定地址*模式，并直接从任务缓冲区装载 SPI TX FIFO。DMA 完成装载 FIFO 后，DMA_DONE_TX 中断触发。桥接器使能 TX_EMPTY 中断，该中断在最后一个字节物理离开 SPI 总线时触发。只有到此时，桥接器才会释放 TX 任务并清除 spiInProgress。
5. *SPI RX 的 DMA：*DMA 通道 CH0 运行在*固定地址到块地址*模式，并将 SPI RX FIFO 中的输入字节捕获到任务缓冲区。当所有 64 个字节均已接收完成时，DMA_DONE_RX 中断触发。该处理程序将任务标记为可进行 USB 传输，并清除 SPISlaveDataReady 和 spiInProgress。

6. *溢出保护：*当 USB 到 SPI 队列已满时，传入的 USB 数据会被读入一个虚拟缓冲区，以防止 TinyUSB RX 缓冲区停滞。当 SPI 到 USB 队列已满时，spi2usb_pushTask() 会立即返回，而不启动新的 SPI 读取。

## 6 软件流程图

图 6-1 展示了此示例的代码流程。

*USB 到 SPI 方向：*

1. 每当 USB 主机发送数据时，TinyUSB 都会调用 tud_cdc_rx_cb()。该回调函数会调用 usb2spi_pushTask()，从 USB CDC 接口读取最多 64 个字节，并将其作为任务加入 gUSB2SPITask 队列。
2. 在主循环中，当 gUSB2SPITask.fifoCount > 0 时，usb2spi_popTask() 会刷新 SPI TX FIFO，使用任务缓冲区地址和长度配置 DMA 通道 CH1，使能 DMA_DONE_TX 中断，并启动 SPI 写传输。
3. 当 DMA 完成装载 TX FIFO 时，DMA_DONE_TX 中断触发。该中断处理程序会清除并使能 TX_EMPTY 中断。当最后一个字节被时钟移出时，TX_EMPTY 中断触发。该处理程序调用 fifoCompleteRead() 释放任务，并清除 spiInProgress。

*SPI 到 USB 方向：*

1. SPI 目标设备拉高数据就绪 GPIO 引脚（上升沿）。GPIO 中断处理程序将 SPISlaveDataReady 置为 true 并立即返回。
2. 在主循环中，当 SPISlaveDataReady 被置位且 spiInProgress 为 false 时，spi2usb_pushTask() 会刷新 RX 和 TX FIFO，为接收缓冲区配置 DMA 通道 CH0，配置 DMA 通道 CH1 以在 MOSI 上发送虚拟数据，然后启动两个 DMA。显式禁用 TX_DONE_TX 中断，以便仅由 DMA_DONE_RX 指示完成。
3. 当所有 64 个字节均已接收完成时，DMA_DONE_RX 中断触发。该处理程序调用 fifoCompleteWrite() 将任务标记为可进行 USB 传输，清除 SPISlaveDataReady，并清除 spiInProgress。
4. 在主循环中，当 gSPI2USBTask.fifoCount > 0 时，spi2usb_popTask() 会检查 TinyUSB CDC TX FIFO 是否有足够空间，写入缓冲数据，将其刷新到主机，并调用 fifoCompleteRead() 释放任务。

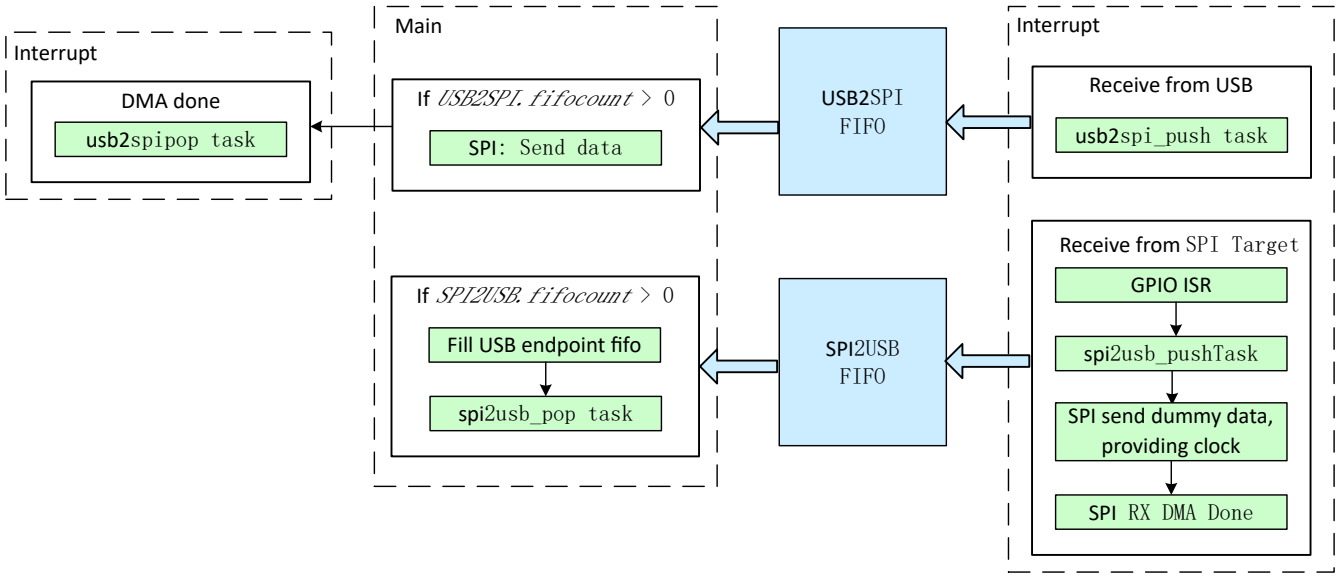


**图 6-1. 应用软件流程图**

# 7 应用代码

*USB 到 SPI：*

```
/*********************************************************
* Function name  : usb2spi_popTask
* Description    : USB -> SPI. Pop task from queue and transmit data through
*                  SPI to slave device. Waits if SPI TX already in progress.
*                  Called from main loop when fifoCount > 0.
* Parameter      : None (modifies gUSB2SPITask, spiInProgress)
* Return         : None
*********************************************************/
void usb2spi_popTask(void)
{
    if(spiInProgress) return;
    if(gUSB2SPITask.fifoCount == 0) return;

    // Current process task
    TxElement_t *pTask = &gUSB2SPITask.fifoPointer[gUSB2SPITask.fifoOut];

    //Flush TXFIFO
    DL_SPI_flushTXFIFO(SPI_0_INST);

    /* Setup DMA and start SPI write */
    DL_SPI_clearInterruptStatus(SPI_0_INST, DL_SPI_INTERRUPT_DMA_DONE_TX);
    DL_SPI_enableInterrupt(SPI_0_INST, DL_SPI_INTERRUPT_DMA_DONE_TX);
    DL_DMA_setSrcAddr(DMA, DMA_CH1_CHAN_ID, (uint32_t)(pTask->buffer));
    DL_DMA_setDestAddr(DMA, DMA_CH1_CHAN_ID, (uint32_t)(&SPI_0_INST->spi->TXDATA));
    DL_DMA_setTransferSize(DMA, DMA_CH1_CHAN_ID, pTask->length);
    DL_DMA_enableChannel(DMA, DMA_CH1_CHAN_ID);

    spiInProgress = true;
}
```

```
/* ===== USB RX -> SPI TX ===== */

/*********************************************************
* Function name  : usb2spi_pushTask
* Description    : USB -> SPI. Read data from USB CDC interface and push
*                  into gUSB2SPITask queue. Each task holds one USB frame
*                  (up to 64 bytes). Called from tud_cdc_rx_cb callback.
* Parameter      : None (modifies gUSB2SPITask)
* Return         : None
*********************************************************/
void usb2spi_pushTask(void)
{
    // Check space for new task
    if(gUSB2SPITask.fifoCount >= SPI_TX_BUF_SIZE) return;

    // Get current task slot in queue
    TxElement_t *pTask = &gUSB2SPITask.fifoPointer[gUSB2SPITask.fifoIn];

    // Read up to 64 bytes from USB CDC
    uint32_t cnt = tud_cdc_n_read(0, pTask->buffer, 64);
    if(cnt == 0) return;                    // Skip if no data

    pTask->length = cnt;

    // Advance queue pointer and mark task as ready
    fifoCompleteWrite(&gUSB2SPITask, SPI_TX_BUF_SIZE);
}
```

*SPI 到 USB：*

```
/* ===== SPI RX -> USB TX ===== */

/*******************************************************
* Function name  : spi2usb_pushTask
* Description    : SPI -> USB. Read data from SPI slave and queue for USB transmission
*                  Triggered when target device signals data ready via GPIO pin.
*                  Called from GPIO interrupt handler.
* Parameter      : None (modifies gSPI2USBTask)
* Return         : None
*******************************************************/
void spi2usb_pushTask(void)
{
    // Check if SPI→USB queue has space for new task
    if (gSPI2USBTask.fifoCount >= USB_TX_BUF_SIZE) return;
    if(spiInProgress) return;

    // Get current task buffer
    TxElement_t *pTask = &gSPI2USBTask.fifoPointer[gSPI2USBTask.fifoIn];

    // Pre-set length since SPI master reads fixed 64 bytes per transaction
    pTask->length = USB_FRAME_PACKAGE;

    //Flush RXFIFO
    DL_SPI_flushRXFIFO(SPI_0_INST);

    //Flush TXFIFO
    DL_SPI_flushTXFIFO(SPI_0_INST);

    // Transmit dummy data
    DL_SPI_disableInterrupt(SPI_0_INST, DL_SPI_INTERRUPT_DMA_DONE_TX);
    DL_DMA_setSrcAddr(DMA, DMA_CH1_CHAN_ID, (uint32_t)(pTask->buffer));
    DL_DMA_setDestAddr(DMA, DMA_CH1_CHAN_ID, (uint32_t)(&SPI_0_INST->spi->TXDATA));
    DL_DMA_setTransferSize(DMA, DMA_CH1_CHAN_ID, USB_FRAME_PACKAGE);
    DL_DMA_enableChannel(DMA, DMA_CH1_CHAN_ID);

    /* DMA to receive data */
    DL_SPI_flushRXFIFO(SPI_0_INST);

    DL_DMA_setSrcAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)(&SPI_0_INST->spi->RXDATA));
    DL_DMA_setDestAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)(pTask->buffer));
    DL_DMA_setTransferSize(DMA, DMA_CH0_CHAN_ID, USB_FRAME_PACKAGE);
    DL_DMA_enableChannel(DMA, DMA_CH0_CHAN_ID);

    spiInProgress = true;
}
```

```
/*******************************************************
* Function name  : spi2usb_popTask
* Description    : SPI -> USB. Pop task from queue and transmit data through
*                  USB CDC interface. All data in one task is sent in same USB
*                  frame. Checks if USB TX buffer has space. Called from main
*                  loop when fifoCount > 0.
* Parameter      : None (modifies gSPI2USBTask)
* Return         : None
*******************************************************/
void spi2usb_popTask(void)
{
    if(gSPI2USBTask.fifoCount == 0) return;

    TxElement_t *pTask = &gSPI2USBTask.fifoPointer[gSPI2USBTask.fifoOut];

    if(tud_cdc_n_write_available(0) >= pTask->length){
        uint32_t written = tud_cdc_n_write(0, pTask->buffer, pTask->length);

        if(written == pTask->length){
            tud_cdc_n_write_flush(0);
            fifoCompleteRead(&gSPI2USBTask, USB_TX_BUF_SIZE);
        }
    }
}
```

*SPI 中断处理程序：*

```
//-----------------------------------------------------------------------+
// SPI Interrupt Handler
//-----------------------------------------------------------------------+
void SPI_0_INST_IRQHandler(void) {
    switch(DL_SPI_getPendingInterrupt(SPI_0_INST)) {
        case DL_SPI_IIDX_DMA_DONE_TX:
            // TX: DMA transfer to SPI FIFO complete
            DL_SPI_clearInterruptStatus(SPI_0_INST, DL_SPI_INTERRUPT_TX_EMPTY);
            DL_SPI_enableInterrupt(SPI_0_INST, DL_SPI_INTERRUPT_TX_EMPTY);
            break;

        case DL_SPI_IIDX_DMA_DONE_RX:
            // RX complete: DMA received 64 bytes from SPI slave
            fifoCompleteWrite(&gSPI2USBTask, USB_TX_BUF_SIZE);
            SPISlaveDataReady = false;  // Clear slave notification flag
            spiInProgress = false;
            break;

            // TX complete
        case DL_SPI_IIDX_TX_EMPTY:
            DL_SPI_disableInterrupt(SPI_0_INST, DL_SPI_INTERRUPT_TX_EMPTY);
            fifoCompleteRead(&gUSB2SPITask, SPI_TX_BUF_SIZE);
            spiInProgress = false;
            break;

        default:
            break;
    }
}
```

## 8 数据流示例

*场景 A：主机向目标写入 5 个字节*

1. 主机通过 COM 端口发送 *HELLO*
2. tud_cdc_rx_cb() 将数据加入队列
   a. usb2spi_pushTask()
   b. 队列 [H, E, L, L, O]，长度=5
3. 主循环：usb2spi_popTask ()
   a. 从队列中获取任务
   b. SPI 写入：data=[H,E,L,L,O]
   c. DMA 传输到 SPI TX FIFO
4. SPI 硬件执行
5. SPI_0_INST_IRQHandler()
   a. DL_SPI_IIDX_DMA_DONE_TX：DMA 已完成
   b. DL_SPI_IIDX_TX_EMPTY：传输完成
   c. fifoCompleteRead() 标记任务已消耗

*场景 B：主机从目标读取 64 个字节*

1. SPI 目标器件有数据，将 GPIO 拉至高电平
2. GPIO 中断触发
   a. SPISlaveDataReady = true
3. 主循环检测：spi2usb_pushTask ()
   a. 刷新 SPI RX/TX FIFO
   b. 设置 DMA 发送：发送虚拟数据（时钟生成）
   c. 设置 DMA 接收：SPI 接收数据至缓冲区
   d. 开始 SPI 传输（64 个字节）
4. SPI 硬件执行
5. SPI_0_INST_IRQHandler()
   a. DL_SPI_IIDX_DMA_DONE_RX：接收到所有数据
   b. fifoCompleteWrite (&gSPI2USBTask) 并标记为 USB 传输准备就绪
6. 主循环：spi2usb_popTask ()
   a. 向 USB 发送 64 个字节
   b. fifoCompleteRead() 标记任务已完成
7. 主机通过 COM 端口接收 64 个字节

## 9 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)，*MSPM0G5187*
- 德州仪器 (TI)，*MSPM0G5117*

## 10 商标

# 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