Subsystem Design

USB 转 UART 桥接器

1 说明

此子系统演示了如何将 MSPM0 器件实现为 USB 通信器件类抽象控制模型 (USB CDC-ACM) 转通用异步收发器 (UART) 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。

该桥接器会将 USB 主机数据转发到 UART 接口，并将 UART 器件数据转发回 USB 主机。此示例使用开源 TinyUSB 库作为 USB 栈。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。

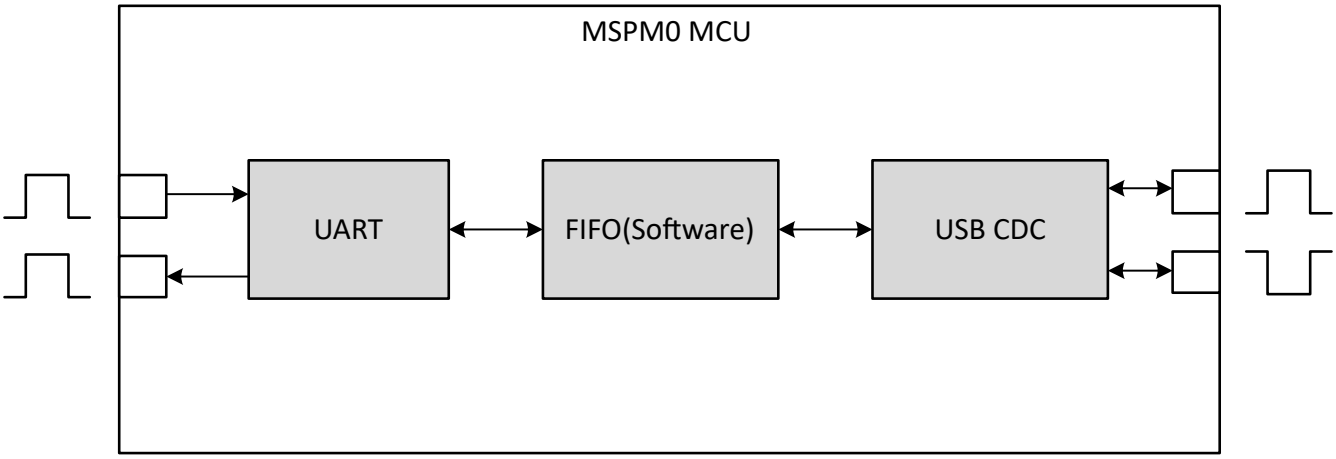

图 1-1. 系统功能方框图

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（UART_0_INST、TIMER_0_INST、DMA_CH0_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)
UART	桥接到目标 UART 器件	UC0 (UART_0_INST)	UC16 (UART_0_INST)
DMA	UART TX 加速	DMA_CH0	DMA_CH0
计时器	UART RX 帧间超时	TIMA0 (TIMER_0_INST)	TIMG0 (TIMER_0_INST)

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 UART 模块。
   a. 将目标波特率设置为 115200。
   b. 重新使能 FIFO 模块。将 RX FIFO 阈值设置为 DL_UART_RX_FIFO_LEVEL_ONE_ENTRY（每接收一个字节触发一次中断）。将 TX FIFO 阈值设置为 DL_UART_TX_FIFO_LEVEL_ONE_ENTRY。
   c. 在“Interrupt Configuration”选项卡中，使能 RX 中断和 DMA 完成 TX 中断。在“DMA Configuration”选项卡中，添加一个 TX DMA 通道 (DMA_CH0)，并配置为块地址到固定地址模式。源长度设置为字节
4. 在 SysConfig 中设置计时器模块。
   a. 添加一个计时器实例 (TIMER_0)。选择 LFCLK 作为时钟源，并使能零中断。
   b. 超时周期在软件中进行设置。

```
// ===== UART RX Timeout =====
// Timeout period: (UART_TIMEOUT+1)/32768 ≈ 200us
// Triggers when no new UART byte arrives for 200us
#define UART_TIMEOUT        6
```

5. 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 / UART_TX_BUF_SIZE)，即每个方向可缓存 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 USB→UART queue
#define UART_TX_BUF_SIZE    8               // Max tasks in UART→USB queue

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

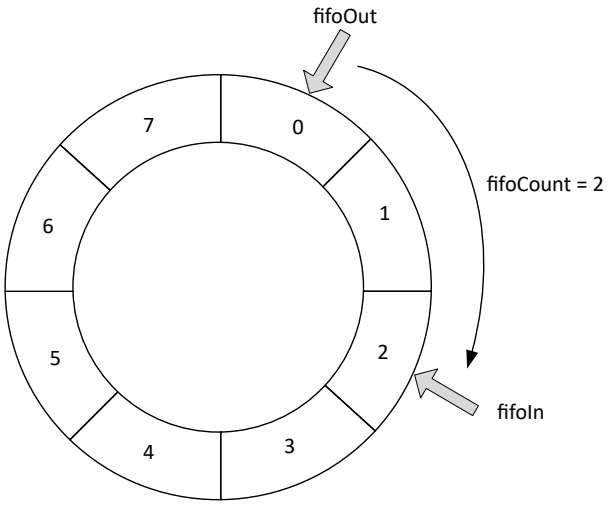

图 5-1. 缓冲区结构

```
// 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. *UART RX 超时机制：*桥接器会将 UART 接收到的字节累加到一个 64 字节任务缓冲区中。达到 64 字节时，桥接器会立即转发该任务。如果 UART 在缓冲字节数少于 64 个时进入空闲状态，则 200 µs 计时器会触发，并刷新该部分缓冲区数据，以保持较低的转发延迟。
3. *用于 UART TX 的 DMA：*在 USB 转 UART 路径上，DMA 可消除 UART 逐字节传输时产生的 ISR 开销。DMA 完成 TX 中断会在 DMA 将数据加载到 UART TX FIFO 时触发，而不是在最后一个字节从引脚发送出去时触发。每次 DMA 完成后需使能 EOT 中断，以确认数据已完全发送，然后再开始下一个任务。
4. *溢出保护：*当 FIFO 队列已满时，桥接器会丢弃接收到的数据。在 USB 转 UART 路径上，桥接器会从 TinyUSB RX 缓冲区读取并丢弃数据，以防止发生停滞。在 UART 转 USB 路径上，它会从 UART RX 寄存器读取并丢弃数据，以清除中断。对于量产设计，建议增加硬件流控制 (UART RTS/CTS) 或 USB CDC 线路状态流控制。
5. *波特率配置：*两个平台的默认波特率均为 115200。若要更改波特率，请更新“SysConfig UART”选项卡中的“Target Baud Rate”字段。SysConfig 会在目标字段下方显示可实现的波特率。请确保 UART 时钟源能够为目标波特率提供足够的频率。不同平台的时钟分频器设置可能有所不同。

6 软件流程图

图 6-1 展示了此示例的代码流程。

USB 转 UART 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 usb2uart_pushTask()，从 USB CDC 接口读取最多 64 个字节，并将其作为任务加入 gUSB2UARTTask 队列。
2. 在主循环中，当 gUSB2UARTTask.fifoCount > 0 时，usb2uart_popTask() 会使用任务缓冲区的地址和长度配置 DMA 通道，然后启动向 UART TX FIFO 的传输。
3. 当 DMA 完成对 FIFO 的加载时，会触发 DMA 完成 TX 中断。处理程序会使能 EOT 中断。当 EOT 触发时，处理程序会清除 uartTxInProgress 标志，以便后续任务继续执行。

UART 转 USB 方向：

1. 从 UART 器件接收到的每个字节都会触发 UART RX 中断，该中断会调用 uart2usb_pushTask() 来累加该字节并重新启动 200 µ s 超时计时器。
2. 当任务缓冲区达到 64 字节时，桥接器会立即调用 fifoCompleteWrite()。
3. 当超时时间已到，而缓冲的字节数少于 64 个时，计时器 ISR 将调用 fifoCompleteWrite() 以刷新该部分帧。
4. 在主循环中，当 gUART2USBTask.fifoCount > 0 时，uart2usb_popTask() 会将缓冲的数据写入 USB CDC TX FIFO 并将其刷新到主机。

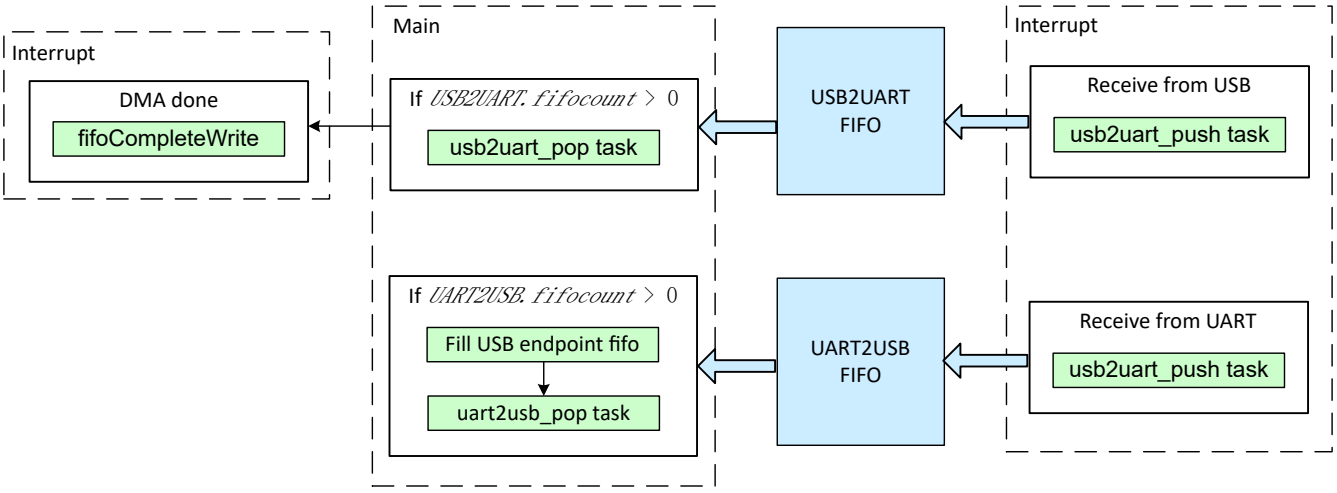

图 6-1. 应用软件流程图

7 应用代码

USB 转 UART：

```
/*******************************************************
* Function name  : usb2uart_popTask
* Description    : USB -> UART. Pop task from queue and transmit data through
*                  UART using DMA transfer. Waits if UART TX already in progress.
*                  Called from main loop when fifoCount > 0.
* Parameter      : None (modifies gUSB2UARTTask, uartTxInProgress)
* Return         : None
*******************************************************/
void usb2uart_popTask(void)
{
    if(uartTxInProgress) return;

    //Current process task
    TxElement_t *pTask = &gUSB2UARTTask.fifoPointer[gUSB2UARTTask.fifoOut];

    //Transmit data through UART DMA
    DL_DMA_setSrcAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t) pTask->buffer);
    DL_DMA_setDestAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)(&UART_0_INST->uart->TXDATA));
    DL_DMA_setTransferSize(DMA, DMA_CH0_CHAN_ID, pTask->length);
    DL_DMA_enableChannel(DMA, DMA_CH0_CHAN_ID);
    uartTxInProgress = true;

    //Update Task status
    fifoCompleteRead(&gUSB2UARTTask, UART_TX_BUF_SIZE);
}
```

```
/*******************************************************
* Function name  : usb2uart_pushTask
* Description    : USB -> UART. Read data from USB CDC interface and push
*                  into gUSB2UARTTask queue. Each task holds one USB frame
*                  (up to 64 bytes). Called from tud_cdc_rx_cb interrupt.
* Parameter      : None (modifies gUSB2UARTTask)
* Return         : None
*******************************************************/
void usb2uart_pushTask(void)
{
    // Get current task slot in queue
    TxElement_t *pTask = &gUSB2UARTTask.fifoPointer[gUSB2UARTTask.fifoIn];

    // Read up to 64 bytes from USB CDC interface (interface 0 is the first/only CDC interface)
    uint32_t cnt = tud_cdc_n_read(0,
                                  pTask->buffer,
                                  USB_FRAME_PACKAGE);

    // Skip if no data received
    if(cnt == 0) return;

    pTask->length = cnt;                    // Store actual data length

    // Advance queue pointer and mark task as ready
    fifoCompleteWrite(&gUSB2UARTTask, UART_TX_BUF_SIZE);
}
```

UART 转 USB：

```
/*********************************************************
* Function name  : uart2usb_pushTask
* Description    : UART -> USB. Read one byte from UART and accumulate into
*                  current task buffer. When buffer full (64 bytes) or when
*                  timeout triggers, move to next task. Called from UART RX
*                  interrupt handler.
* Parameter      : None (modifies gUART2USBTask, TIMER_0_INST)
* Return         : None
*********************************************************/
void uart2usb_pushTask(void)
{
    uint8_t byte = DL_UART_Main_receiveData(UART_0_INST);

    // Get current task buffer
    TxElement_t *pTask = &gUART2USBTask.fifoPointer[gUART2USBTask.fifoIn];

    // Check if current task buffer is full (64 bytes)
    if (pTask->length < USB_FRAME_PACKAGE - 1) {
        // Buffer not full: add byte and restart timeout timer
        pTask->buffer[pTask->length++] = byte;

        // Restart timeout to wait for more data
        DL_Timer_setTimerCount(TIMER_0_INST, UART_TIMEOUT);
        DL_Timer_startCounter(TIMER_0_INST);
    }
    else if(pTask->length == USB_FRAME_PACKAGE - 1){
        // Adding last byte (position 63): task becomes full
        pTask->buffer[pTask->length++] = byte;

        // Move to next task since buffer is now full (64 bytes)
        fifoCompleteWrite(&gUART2USBTask, USB_TX_BUF_SIZE);
    }
    else {
        // Error: buffer overflow (should never happen if logic is correct)
        __BKPT(0);
    }
}
```

```
/*********************************************************
* Function name  : uart2usb_popTask
* Description    : UART -> 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 gUART2USBTask)
* Return         : None
*********************************************************/
void uart2usb_popTask(void)
{
    // Get current task
    TxElement_t *pTask = &gUART2USBTask.fifoPointer[gUART2USBTask.fifoOut];

    // Wait for USB TX FIFO to be completely empty before sending next task
    // This ensures clean task boundaries and clear completion points
    if(tud_cdc_n_write_available(0) >= pTask->length){
        // FIFO has enough space for entire task
        uint32_t written = tud_cdc_n_write(0, pTask->buffer, pTask->length);

        // Only advance queue if entire task was written in one go
        if(written == pTask->length){
            tud_cdc_n_write_flush(0);
            fifoCompleteRead(&gUART2USBTask, USB_TX_BUF_SIZE);
        }
    }
}
```

8 数据流示例

场景 A：主机向器件写入 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
3. 主循环：usb2uart_popTask() 通过 UART DMA 进行传输
4. UART 硬件发送具有起始位/停止位的字节
5. 器件成功接收 *HELLO*

场景 B：器件发送可变长度数据

1. UART 器件发送 *DATA*（4 字节）
2. 每个字节都会触发 UART RX 中断
3. 接收到第 4 个字节后，触发 200us 超时
4. uart2usb_pushTask() 将 4 字节数据帧加入队列
5. 主循环：uart2usb_popTask() 将数据发送至 USB
6. 主机通过 COM 端口接收 4 个字节

场景 C：快速突发数据（64 字节）

1. UART 器件发送快速数据流
2. 字节在 FIFO 中累加
3. 达到 64 字节时，缓冲区满触发加入队列操作
4. uart2usb_popTask() 将完整的 64 字节数据帧发送至 USB
5. 主机接收到完整数据块

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

LaunchPad™ is a trademark of Texas Instruments.
Windows® is a registered trademark of Microsoft Corporation.
macOS® is a registered trademark of Apple Inc..
Linux® is a registered trademark of Linus Torvalds.
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、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