Subsystem Design

I2C 转 UART 子系统设计

设计说明

该子系统用作 I2C 转 UART 桥接器。在该子系统中，MSPM0 器件是 I2C 目标器件。当 I2C 控制器向 I2C 目标器件发送数据时，目标器件会收集接收到的所有数据。一旦目标器件检测到停止条件，目标器件就会使用 UART 接口将数据发送出去。当 I2C 控制器尝试从电桥读取时，电桥传输从 UART 器件接收到的最后一个字节。当 I2C 控制器读取两个字节时，电桥会传输从 UART 器件接收到的最后一个字节和电桥生成的最新错误代码。

MSPM0 通过 I2C SCL 和 SDA 线连接到 I2C 控制器。MSPM0 还使用 UART TX 和 RX 线路连接到 UART 器件。

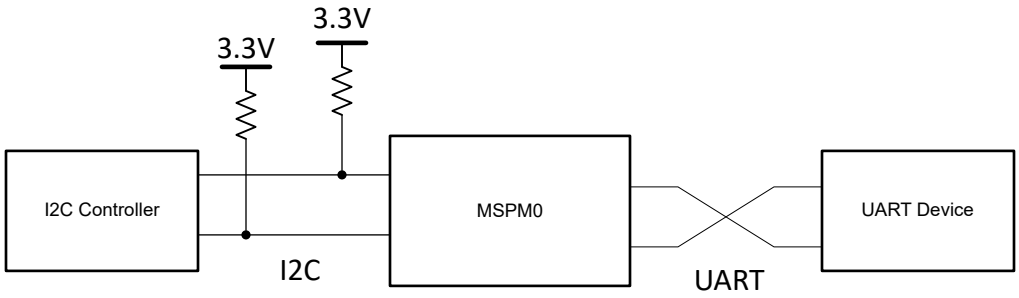

图 1-1. 系统功能方框图

所需外设

使用的外设	说明
I2C	在代码中称为 I2C_INST
UART	在代码中称为 UART_INST

兼容器件

根据所需外设 中所示的要求，该示例与兼容器件 中所示的器件兼容。相应的 EVM 可用于原型设计。

兼容器件	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

设计步骤

1. 在 SysConfig 中设置 I2C 模块。将器件设置为目标模式，并启用 RX FIFO 触发、开始检测、停止检测、目标仲裁丢失、TX FIFO 下溢、RX FIFO 溢出和中断溢出中断。
2. 在 SysConfig 中设置 UART 模块。为器件选择所需的波特率。使能接收、发送、溢出错误、中断错误、帧错误、奇偶校验错误、噪声错误和 RX 超时。

设计注意事项

1. 在应用程序代码中，确保 I2C_MAX_PACKET_SIZE 足够大，可包含要传输的数据包。
2. 确保为所使用的 I2C 模块选择适当的上拉电阻值。一般而言，10kΩ 适用于 100kHz 频率。较高的 I2C 总线速率需要值较低的上拉电阻。对于 400kHz 通信，请使用更接近 4.7kΩ 的电阻器。
3. 要提高 UART 波特率，请调整标记为 *Target Baud Rate* 的 SysConfig UART 选项卡中的值。在此下方，观察计算得出的波特率变化以反映目标波特率。这可以使用可用的时钟和分频器进行计算。
4. 检查错误标志并进行适当处理。UART 和 I2C 外设都能够引发信息性错误中断。为了方便调试，该子系统在引发错误代码时使用枚举和全局变量来保存错误代码。在实际应用中，应在代码中处理错误，这样错误就不会使工程崩溃。

软件流程图

图 1-2 展示了此示例的代码流程图，并说明了器件如何使用接收到的 I2C 数据填充数据缓冲区，然后通过 UART 传输数据。

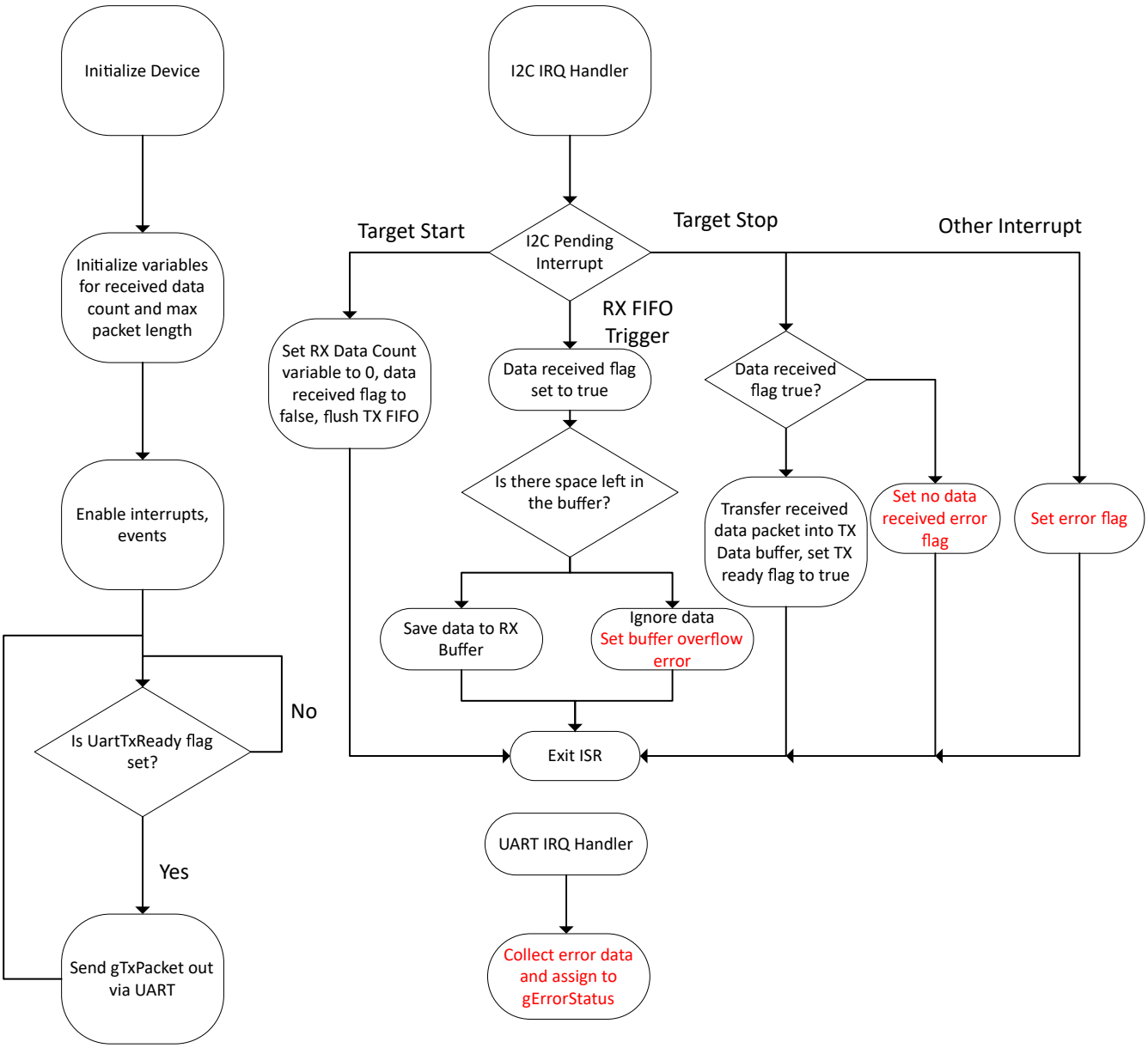

图 1-2. 应用软件流程图

器件配置

该应用利用 TI 系统配置工具 (SysConfig) 图形界面来生成器件外设的配置代码。使用图形界面配置器件外设可简化应用原型设计过程。

可以在 *i2c_to_uart_bridge.c* 文件的 main() 开头找到图 1-2 中所述内容的代码。

应用代码

该应用程序必须为接收到的数据和要发送的数据分配内存。该应用程序还需要统计接收和传输的数据量。需要一个标志来确定正在接收的数据何时完成并准备好通过 UART 发送出去。还有一个错误代码枚举，以及一个用于保存它们的变量。缓冲区、计数器、枚举和标志的初始化如下所示：

```
#include "ti_msp_dl_config.h"

/* Maximum size of TX packet */
#define I2C_TX_MAX_PACKET_SIZE (1)

/* Maximum size of RX packet */
#define I2C_RX_MAX_PACKET_SIZE (16)

/* Data sent to Controller in response to Read transfer */
uint8_t gTxPacket[I2C_TX_MAX_PACKET_SIZE] = {0x00};

/* Counters for TX length and bytes sent */
uint32_t gTxLen, gTxCount;

/* Data received from Controller during a Write transfer */
uint8_t gRxPacket[I2C_RX_MAX_PACKET_SIZE];
/* Counters for TX length and bytes sent */
uint32_t gRxLen, gRxCount;

enum error_codes{
    NO_ERROR,
    DATA_BUFFER_OVERFLOW,
    RX_FIFO_FULL,
    NO_DATA_RECEIVED,
    I2C_TARGET_TXFIFO_UNDERFLOW,
    I2C_TARGET_RXFIFO_OVERFLOW,
    I2C_TARGET_ARBITRATION_LOST,
    I2C_INTERRUPT_OVERFLOW,
    UART_OVERRUN_ERROR,
    UART_BREAK_ERROR,
    UART_PARITY_ERROR,
    UART_FRAMING_ERROR,
    UART_RX_TIMEOUT_ERROR
};

uint8_t gErrorStatus = NO_ERROR;

/* Buffer to hold data received from UART device */
uint8_t gUARTRxData = 0;
/* Flags */
bool gUartTxReady = false;  /* Flag to start UART transfer */
bool gUartRxDone = false;   /* Flag to indicate UART data has been received */
```

应用程序代码的主体相对较短。首先，器件和外设被初始化。然后启用中断和事件。计数器值也会被初始化。最后，到达主循环，其中轮询标志检测接收到的数据何时准备好通过 UART 传回：

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

    gTxCount = 0;
    gTxLen   = I2C_TX_MAX_PACKET_SIZE;
    DL_I2C_enableInterrupt(I2C_INST, DL_I2C_INTERRUPT_TARGET_TXFIFO_TRIGGER);

    /* Initialize variables to receive data inside RX ISR */
    gRxCount = 0;
    gRxLen   = I2C_RX_MAX_PACKET_SIZE;

    NVIC_EnableIRQ(I2C_INST_INT_IRQN);
    NVIC_EnableIRQ(UART_INST_INT_IRQN);

    while (1) {
        if(gUartTxReady){
            gUartTxReady = false;
            for(int i = 0; i < gRxCount; i++){
                /* Transmit data out via UART and wait until transfer is complete */
                DL_UART_Main_transmitDataBlocking(UART_INST, gTxPacket[i]);
            }
        }
```

```
    }
}
```

此代码的下一段是 I2C IRQ 处理程序。此代码用于启动然后停止数据收集。接下来，此代码在接收数据时保存该数据。当挂起中断是检测到的 I2C 启动条件时，器件会初始化计数器变量。当挂起中断表示 RX FIFO 有数据可用时，器件会检查数据缓冲区中是否存在剩余空间。如果有空间，则保存接收到的值。如果没有更多的空间，接收到的值会被忽略。当挂起中断是 TX FIFO 触发信号时，器件会检查已发送了多少个字节。如果器件已经发送了一个字节，FIFO 中将填充最近报告的错误代码。当挂起中断是一个 I2C 停止条件时，器件会检查是否接收到数据。如果接收到数据，接收到的数据缓冲区就会复制到发送数据缓冲区，UART TX 就绪标志设置为 true。如果未收到任何数据，器件不会发送任何内容。该 ISR 还通过以下方式处理 I2C 错误中断：向 gErrorStatus 变量分配适当的错误代码。

```
void I2C_INST_IRQHandler(void)
{
    static bool dataRx = false;

    switch (DL_I2C_getPendingInterrupt(I2C_INST)) {
        case DL_I2C_IIDX_TARGET_START:
            /* Initialize RX or TX after Start condition is received */
            gTxCount = 0;
            gRxCount = 0;
            dataRx   = false;
            /* Flush TX FIFO to refill it */
            DL_I2C_flushTargetTXFIFO(I2C_INST);
            break;
        case DL_I2C_IIDX_TARGET_RXFIFO_TRIGGER:
            /* Store received data in buffer */
            dataRx = true;
            while (DL_I2C_isTargetRXFIFOEmpty(I2C_INST) != true) {
                if (gRxCount < gRxLen) {
                    gRxPacket[gRxCount++] = DL_I2C_receiveTargetData(I2C_INST);
                } else {
                    /* Prevent overflow and just ignore data */
                    DL_I2C_receiveTargetData(I2C_INST);
                }
            }
            break;
        case DL_I2C_IIDX_TARGET_TXFIFO_TRIGGER:
            /* Fill TX FIFO if there are more bytes to send */
            if (gTxCount < gTxLen) {
                gTxCount += DL_I2C_fillTargetTXFIFO(
                    I2C_INST, &gUARTRxData, (gTxLen - gTxCount));
            } else {
                /*
                 * Fill FIFO with error status after sending latest received
                 * byte
                 */
                while (DL_I2C_transmitTargetDataCheck(I2C_INST, gErrorStatus) != false)
                    ;
            }
            break;
        case DL_I2C_IIDX_TARGET_STOP:
            /* If data was received, echo to TX buffer */
            if (dataRx == true) {
                for (uint16_t i = 0;
                     (i < gRxCount) && (i < I2C_TX_MAX_PACKET_SIZE); i++) {
                    gTxPacket[i] = gRxPacket[i];
                    DL_I2C_flushTargetTXFIFO(I2C_INST);
                }
                dataRx = false;
            }
            /* Set flag to indicate data ready for UART TX */
            gUartTxReady = true;
            break;
        case DL_I2C_IIDX_TARGET_RX_DONE:
            /* Not used for this example */
        case DL_I2C_IIDX_TARGET_RXFIFO_FULL:
            /* Not used for this example */
        case DL_I2C_IIDX_TARGET_GENERAL_CALL:
            /* Not used for this example */
        case DL_I2C_IIDX_TARGET_EVENT1_DMA_DONE:
            /* Not used for this example */
        case DL_I2C_IIDX_TARGET_EVENT2_DMA_DONE:
```

```
            /* Not used for this example */
        case DL_I2C_IIDX_TARGET_TXFIFO_UNDERFLOW:
            gErrorStatus = I2C_TARGET_TXFIFO_UNDERFLOW;
            break;
        case DL_I2C_IIDX_TARGET_RXFIFO_OVERFLOW:
            gErrorStatus = I2C_TARGET_RXFIFO_OVERFLOW;
            break;
        case DL_I2C_IIDX_TARGET_ARBITRATION_LOST:
            gErrorStatus = I2C_TARGET_ARBITRATION_LOST;
            break;
        case DL_I2C_IIDX_INTERRUPT_OVERFLOW:
            gErrorStatus = I2C_INTERRUPT_OVERFLOW;
            break;
        default:
            break;
    }
}
```

本示例中的最后一段代码是 UART IRQ 处理程序。UART IRQ 处理程序仅用于保存接收到的数据，并检查是否存在错误。当 UART RX 中断挂起时，器件将接收到的数据保存到缓冲区 gUARTRxData，然后设置一个标志以指示保存了新的 RX 数据。当 UART 错误确实发生时，该 ISR 会执行以将正确的错误代码分配给 gErrorStatus。

```
void UART_INST_IRQHandler(void)
{
    switch (DL_UART_Main_getPendingInterrupt(UART_INST)) {
        case DL_UART_MAIN_IIDX_RX:
            DL_UART_Main_receiveDataCheck(UART_INST, &gUARTRxData);
            gUartRxDone = true;
            break;
        case DL_UART_INTERRUPT_OVERRUN_ERROR:
            gErrorStatus = UART_OVERRUN_ERROR;
            break;
        case DL_UART_INTERRUPT_BREAK_ERROR:
            gErrorStatus = UART_BREAK_ERROR;
            break;
        case DL_UART_INTERRUPT_PARITY_ERROR:
            gErrorStatus = UART_PARITY_ERROR;
            break;
        case DL_UART_INTERRUPT_FRAMING_ERROR:
            gErrorStatus = UART_FRAMING_ERROR;
            break;
        case DL_UART_INTERRUPT_RX_TIMEOUT_ERROR:
            gErrorStatus = UART_RX_TIMEOUT_ERROR;
            break;
        default:
            break;
    }
}
```

其他资源

1. 德州仪器 (TI)，下载 MSPM0 SDK
2. 德州仪器 (TI)，详细了解 SysConfig
3. 德州仪器 (TI)，MSPM0L LaunchPad™
4. 德州仪器 (TI)，MSPM0G LaunchPad™
5. 德州仪器 (TI)，MSPM0 I2C Academy
6. 德州仪器 (TI)，MSPM0 UART Academy

重要声明和免责声明

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 产品的应用。严禁对这些资源进行其他复制或展示。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