Subsystem Design

数据传感器聚合器子系统设计

TEXAS INSTRUMENTS

设计说明

该子系统用作 BP-BASSSENSORSMKII BoosterPack™ 插件模块的接口。该模块具有温度和湿度传感器、霍尔效应传感器、环境光传感器、惯性测量单元和磁力计。该模块用于连接 TI LaunchPad™ 开发套件。该子系统使用 I2C 接口从这些传感器收集数据，并使用 UART 接口将数据传输出去。这有助于用户快速进入原型设计阶段并使用 MSPM0 和 BASSSENSORSMKII BoosterPack 模块进行试验。

MSPM0 使用 I2C 接口连接到 BP-BASSSENSORSMKII，使用 UART 接口传输处理后的数据。

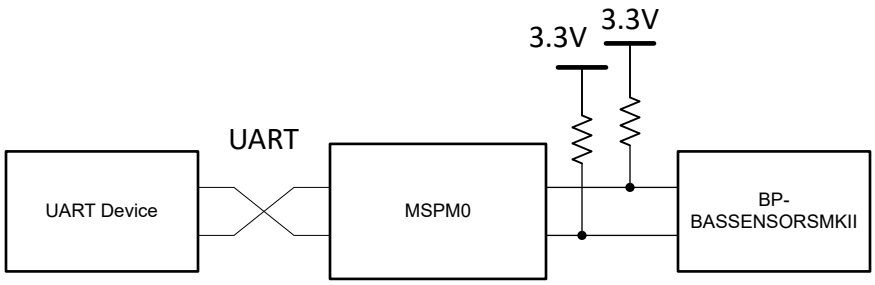

图 1-1. 系统功能方框图

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_0_INST
DMA	用于 UART TX
GPIO	这五个 GPIO 分别称为：HDC_V、DRV_V、OPT_V、INT1 和 INT2
ADC	在代码中称为 ADC12_0_INST
事件	用于将数据传输到 UART TX FIFO

兼容器件

根据所需外设中所示的要求，本示例与下表所示的器件兼容。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 GPIO 模块。添加一个名为 HDC_V 的 GPIO 作为 PB24 上的输出。添加名为 DRV_V 的第二个 GPIO 作为 PA22 上的输出。添加名为 OPT_V 的第三个 GPIO 作为 PA24 上的输出。添加名为 INT1 的第四个 GPIO 作为 PA26 上的输出。添加名为 INT2 的第五个也是最后一个 GPIO 作为 PB6 上的输出。
2. 在 SysConfig 中设置 ADC12 模块。在自动采样模式下使用单次转换模式添加实例，从地址零开始。将触发源设置为软件。打开“ADC Conversion memory configurations”选项卡，并确保内存 0 命名为 0，使用 PA25

English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

上的通道 2，以 VDDA 作为参考电压，以采样计时器 0 作为采样周期来源。在“Interrupt Configuration”选项卡中，为加载的 MEM0 结果启用中断。

3. 在 SysConfig 中设置 I2C 模块。启用控制器模式，并将总线速度设置为 100kHz。在“Interrupt Configuration”选项卡中，启用“RX Done”、“TX Done”、“RX FIFO Trigger”和“Addr/Data NACK”中断。在 PinMux 部分中，确保所选外设为 I2C1，PB3 上为 SDA，PB2 上为 SCL。
4. 在 SysConfig 中设置 UART 模块。添加 UART 实例，使用 9600Hz 波特率。在“Interrupt Configuration”选项卡中，启用“DMA Done On Transit”和“End of Transmission”中断。在“DMA Configuration”选项卡中，选择“DMA TX Trigger”作为 UART TX 中断，并启用。确保 DMA 通道 TX 设置针对固定地址模式使用块，并将源和目标长度设置为字节。将源地址方向设置为递增，将传输模式设置为单字节。源地址增量和目标地址增量均应设置为“Do not change address after each transfer”。在 PinMux 部分中，为 RX 选择 UART0 和 PA11，为 TX 选择 PA10。

设计注意事项

1. 确保您已检查并验证代码开头定义的最大数据包大小，从而实现您对子系统的正常使用。
2. 为您使用的 I2C 模块选择合适的上拉电阻值。根据经验，10kΩ 电阻适合 100kHz。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。
3. 要提高 UART 的波特率，请在 SysConfig 中打开 UART 模块，然后编辑目标波特率值。显示计算出的实际波特率和计算出的误差。
4. 为了帮助您在此处添加错误检测和处理功能以获得更强大的应用程序，许多模块都具有错误中断，用于轻松监控错误情况。
5. 请参阅“发送”功能来编辑通过 UART 发送数据的格式。

软件流程图

以下流程图简要展示了从传感器 BoosterPack 插件模块读取、收集、处理和传输数据所执行的软件步骤。

English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

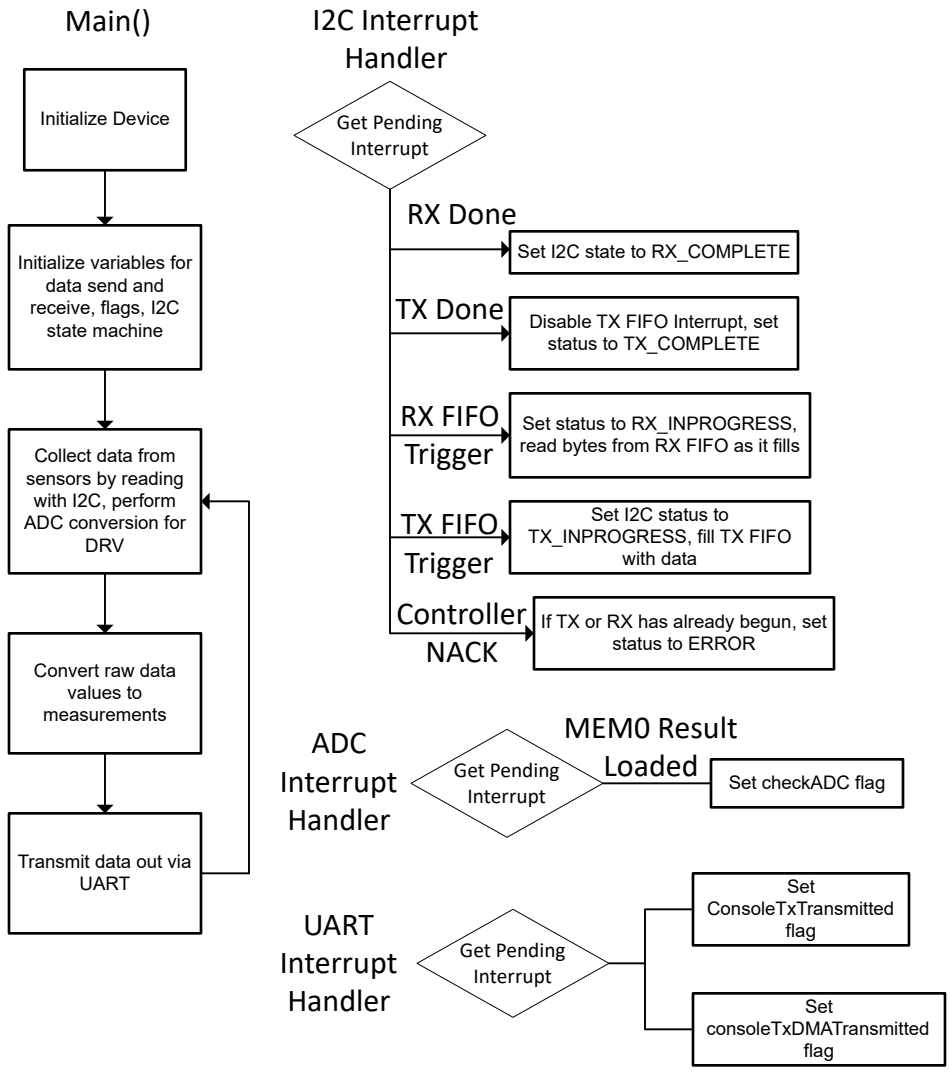

图 1-2. 应用软件流程图

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 *data_sensor_aggregator.c* 文件的 main() 的开头找到 软件流程图 中所述内容的代码。

应用代码

该应用首先设置 UART 和 I2C 传输的大小，然后分配内存来存储要传输的值。然后，它为最终的后处理测量分配内存，以便保存以通过 UART 进行传输。它还定义了一个用于记录 I2C 控制器状态的枚举。您可能需要调整某些数据包大小并更改您自己的实现中的某些数据存储。此外，建议为某些应用添加错误处理功能。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ti_msp_dl_config.h"

/* Initializing functions */

void DataCollection(void);
void TxFunction(void);
```


English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

```
void RxFunction(void);
void Transmit(void);
void UART_Console_write(const uint8_t *data, uint16_t size);

/* Earth's gravity in m/s^2 */
#define GRAVITY_EARTH (9.80665f)

/* Maximum size of TX packet */
#define I2C_TX_MAX_PACKET_SIZE (16)

/* Number of bytes to send to target device */
#define I2C_TX_PACKET_SIZE (3)

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

/* Number of bytes to received from target */
#define I2C_RX_PACKET_SIZE (16)

/*
 * Number of bytes for UART packet size
 * The packet will be transmitted by the UART.
 * This example uses FIFOs with polling, and the maximum FIFO size is 4.
 * Refer to interrupt examples to handle larger packets.
 */
#define UART_PACKET_SIZE (8)

uint8_t gSpace[] = "\r\n";
volatile bool gConsoleTxTransmitted;
volatile bool gConsoleTxDMATransmitted;
/* Data for UART to transmit */
uint8_t gTxData[UART_PACKET_SIZE];

/* Booleans for interrupts */
bool gCheckADC;
bool gDataReceived;

/* Variable to change the target address */
uint8_t gTargetAdd;

/* I2C variables for data collection */
float gHumidity, gTempHDC, gAmbient;
uint16_t gAmbientE, gAmbientR, gDRV;
uint16_t gMagX, gMagY, gMagZ, gGyrX, gGyrY, gGyrZ, gAccX, gAccY, gAccZ;

/* Data sent to the Target */
uint8_t gTxPacket[I2C_TX_MAX_PACKET_SIZE];

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

/* Data received from Target */
uint8_t gRxPacket[I2C_RX_MAX_PACKET_SIZE];

/* Counters for TX length and bytes sent */
uint32_t gRxLen, gRxCount;

/* Indicates status of I2C */
enum I2cControllerStatus {
    I2C_STATUS_IDLE = 0,
    I2C_STATUS_TX_STARTED,
    I2C_STATUS_TX_INPROGRESS,
    I2C_STATUS_TX_COMPLETE,
    I2C_STATUS_RX_STARTED,
    I2C_STATUS_RX_INPROGRESS,
    I2C_STATUS_RX_COMPLETE,
    I2C_STATUS_ERROR,
} gI2cControllerStatus;
```

此应用中的 Main() 会初始化所有外围模块，然后在主循环中设备仅收集来自传感器的所有数据，并在处理后进行传输。

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

    NVIC_EnableIRQ(I2C_INST_INT_IRQN);
```


English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

```
    NVIC_EnableIRQ(ADC12_0_INST_INT_IRQN);
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);
    DL_SYSCTL_disableSleepOnExit();    while(1) {
        DataCollection();
        Transmit();
    /* This delay is to the data is transmitted every few seconds */
        delay_cycles(100000000);
    }
}
```

下一个代码块包含所有中断服务例程。第一个是 I2C 例程，接下来是 ADC 例程，最后是 UART 例程。I2C 例程主要用于更新某些标志以及更新控制器状态变量。它还管理 TX 和 RX FIFO。ADC 中断服务例程会设置一个标志，以便主循环可以检查 ADC 值何时有效。UART 中断服务例程也只是设置标志来确认 UART 数据的有效性。

```
void I2C_INST_IRQHandler(void)
{
    switch (DL_I2C_getPendingInterrupt(I2C_INST)) {
        case DL_I2C_IIDX_CONTROLLER_RX_DONE:
            gI2cControllerStatus = I2C_STATUS_RX_COMPLETE;
            break;
        case DL_I2C_IIDX_CONTROLLER_TX_DONE:
            DL_I2C_disableInterrupt(
                I2C_INST, DL_I2C_INTERRUPT_CONTROLLER_TXFIFO_TRIGGER);
            gI2cControllerStatus = I2C_STATUS_TX_COMPLETE;
            break;
        case DL_I2C_IIDX_CONTROLLER_RXFIFO_TRIGGER:
            gI2cControllerStatus = I2C_STATUS_RX_INPROGRESS;
            /* Receive all bytes from target */
            while (DL_I2C_isControllerRXFIFOEmpty(I2C_INST) != true) {
                if (gRxCount < gRxLen) {
                    gRxPacket[gRxCount++] =
                        DL_I2C_receiveControllerData(I2C_INST);
                } else {
                    /* Ignore and remove from FIFO if the buffer is full */
                    DL_I2C_receiveControllerData(I2C_INST);
                }
            }
            break;
        case DL_I2C_IIDX_CONTROLLER_TXFIFO_TRIGGER:
            gI2cControllerStatus = I2C_STATUS_TX_INPROGRESS;
            /* Fill TX FIFO with next bytes to send */
            if (gTxCount < gTxLen) {
                gTxCount += DL_I2C_fillControllerTXFIFO(
                    I2C_INST, &gTxPacket[gTxCount], gTxLen - gTxCount);
            }
            break;
            /* Not used for this example */
        case DL_I2C_IIDX_CONTROLLER_ARBITRATION_LOST:
        case DL_I2C_IIDX_CONTROLLER_NACK:
            if ((gI2cControllerStatus == I2C_STATUS_RX_STARTED) ||
                (gI2cControllerStatus == I2C_STATUS_TX_STARTED)) {
                /* NACK interrupt if I2C Target is disconnected */
                gI2cControllerStatus = I2C_STATUS_ERROR;
            }
        case DL_I2C_IIDX_CONTROLLER_RXFIFO_FULL:
        case DL_I2C_IIDX_CONTROLLER_TXFIFO_EMPTY:
        case DL_I2C_IIDX_CONTROLLER_START:
        case DL_I2C_IIDX_CONTROLLER_STOP:
        case DL_I2C_IIDX_CONTROLLER_EVENT1_DMA_DONE:
        case DL_I2C_IIDX_CONTROLLER_EVENT2_DMA_DONE:
        default:
            break;
    }
}

void ADC12_0_INST_IRQHandler(void)
{
    switch (DL_ADC12_getPendingInterrupt(ADC12_0_INST)) {
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true;
            break;
        default:
            break;
    }
}
```


English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

```
void UART_0_INST_IRQHandler(void)
{
    switch (DL_UART_Main_getPendingInterrupt(UART_0_INST)) {
        case DL_UART_MAIN_IIDX_EOT_DONE:
            gConsoleTxTransmitted = true;
            break;
        case DL_UART_MAIN_IIDX_DMA_DONE_TX:
            gConsoleTxDMATransmitted = true;
            break;
        default:
            break;
    }
}
```

该块会格式化数据，以使用 UART 接口发送。它以易于阅读的格式传递数据，以便在 UART 终端等设备上查看。在您自己的实现中，您可能希望更改正在传输的数据的格式。

```
/* This function formats and transmits all of the collected data over UART */
void Transmit(void)
{
    int count = 1;
    char buffer[20];
    while (count < 14)
    {
    /* Formatting the name and converting int to string for transfer */
        switch(count){
            case 1:
                gTxData[0] = 84;
                gTxData[1] = 67;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%f", gTempHDC);
                break;
            case 2:
                gTxData[0] = 72;
                gTxData[1] = 37;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%f", gHumidity);
                break;
            case 3:
                gTxData[0] = 65;
                gTxData[1] = 109;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%f", gAmbient);
                break;
            case 4:
                gTxData[0] = 77;
                gTxData[1] = 120;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gMagX);
                break;
            case 5:
                gTxData[0] = 77;
                gTxData[1] = 121;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gMagY);
                break;
            case 6:
                gTxData[0] = 77;
                gTxData[1] = 122;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gMagZ);
                break;
            case 7:
                gTxData[0] = 71;
                gTxData[1] = 120;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gGyrX);
                break;
            case 8:
```


English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

```
                gTxData[0] = 71;
                gTxData[1] = 121;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gGyrY);
                break;
            case 9:
                gTxData[0] = 71;
                gTxData[1] = 122;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gGyrZ);
                break;
            case 10:
                gTxData[0] = 65;
                gTxData[1] = 120;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gAccX);
                break;
            case 11:
                gTxData[0] = 65;
                gTxData[1] = 121;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gAccY);
                break;
            case 12:
                gTxData[0] = 65;
                gTxData[1] = 122;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gAccZ);
                break;
            case 13:
                gTxData[0] = 68;
                gTxData[1] = 82;
                gTxData[2] = 86;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gDRV);
                break;
        }
        count++;
    /* Filling the UART transfer variable */
        gTxData[4] = buffer[0];
        gTxData[5] = buffer[1];
        gTxData[6] = buffer[2];
        gTxData[7] = buffer[3];

        /* Optional delay to ensure UART TX is idle before starting transmission */
        delay_cycles(160000);

        UART_Console_write(&gTxData[0], 8);
        UART_Console_write(&gSpace[0], sizeof(gSpace));
    }
    UART_Console_write(&gSpace[0], sizeof(gSpace));
}
```


English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

附加资源

1. 下载 MSPM0 SDK
2. 了解有关 SysConfig 的更多信息
3. MSPM0L LaunchPad 开发套件
4. MSPM0G LaunchPad 开发套件
5. MSPM0 I2C Academy
6. MSPM0 UART Academy
7. MSPM0 ADC Academy
8. MSPM0 DMA Academy
9. MSPM0 Events Manager Academy

English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

修订历史记录

日期	修订版本	说明
January 2024	*	初始发行版

English Document: SLAAE83
Copyright © 2024 Texas Instruments Incorporated

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
Copyright © 2024，德州仪器 (TI) 公司