




InvenSense Inc.  
1197 Borregas Ave., Sunnyvale, CA 94089 USA  
电话: +1 (408) 988-7339 传真: +1 (408) 988-8104 网站:  
www.invensense.com

文件编号: AN-EMAPS-0.0.6  
修订: 1.2  
发布日期: 05/05/2015

## Motion Driver 6.12 – 用户指南

	Motion Driver 6.12 – 用户指南	文件编号 :AN-EMAPS-0.0.6 修订 :1.2 发布日期 :05/05/2015
---	---------------------------	---

目录

1 修订历史 .....3

2 目的..... 4

3 开始之前 .....4

4 运动驱动器 6.12 特性.....4

5 选择 MCU .....5

6 连接硬件 .....5

7 运动驱动器 6.12 固件包 .....6

8 集成运动驱动器 6.12 .....6

9 初始化 API.....9

10 方向矩阵 .....9

11 中断处理 .....10

12 DMP - 数字运动处理器™ ..... 10

    12.1 DMP初始化..... 10

    12.2 DMP功能.....10

    12.3 DMP FIFO输出.....11

13 INVENSENSE 硬件自检.....11

14 校准数据和存储.....12

    14.1工厂生产线校准.....12

    14.2保存和加载校准数据.....13

15 集成 MPL 库.....13


16 MPU6500/MPU9250 的低功耗加速模式和运动中断模式 .....14

17 编译器特定设置.....14



## 1 修订历史

修订日期	修订说明	
2014 年 6 月 27 日	1.0	初始发行
2014 年 7 月 17 日	1.1	ARM MPL 库的扩展信息
2015 年 5 月 15 日	1.2	针对 MD 6.12 版本更新

	Motion Driver 6.12 – 用户指南	文件编号: AN-EMAPS-0.0.6 修订: 1.2 发布日期: 05/05/2015
---	---------------------------	---

## 2 用途Motion

Driver 是传感器驱动层的嵌入式软件堆栈,可轻松配置和利用

InvenSense 运动跟踪解决方案的许多功能。支持的运动设备有 MPU6050/MPU6500/MPU9150/MPU9250。硬件和板载数字运动处理器 (DMP) 的许多功能都被封装到可以使用和引用的模块化 API 中。

Motion Driver 被设计为可以轻松移植到大多数 MCU 的解决方案。随着 Motion Driver 6.12 的发布,它包括适用于 ARM MCU 和 TI-MSP430 的 9 轴解决方案。仅 6 轴解决方案应继续参考 Motion Driver 5.1.2,以便更容易理解该软件。

本文档重点介绍了使用 Motion Driver 6.12 作为参考开始开发嵌入式项目时将遇到的基本过程 and 选择。我们将讨论一些更详细的主题,例如 DMP 编程、校准和自检。

## 3 开始之前

请阅读 Motion Driver 6.12 入门指南和 Motion Driver 6.12 功能指南。建议客户在其中一个移植平台 (TI-MSP430 或 IAR ARM)上启动 Motion Driver 6.12,以便他们更好地理解代码和功能。在了解了这些功能之后,它将更容易将其移植到您的生态系统中。

## 4 运动驱动器 6.12 特性

这是对 MD6.12 功能的快速概述。

- DMP 功能：
  - o 3/6 轴低功率四元数
  - o 点击、方向和计步器手势检测
- MPL 算法：
  - o 运行时陀螺仪校准
  - o 运行时间陀螺温度补偿
  - o 运行时间罗盘校准
  - o 运行时抗磁干扰
  - o 3/6/9 轴传感器融合
- 硬件特性：
  - o 工厂校准
  - o 工厂自检
  - o 保存和加载传感器状态
  - o 低功耗加速模式
  - o 低功耗运动中断模式
  - o 注册转储



## 5 选择单片机

对于每个嵌入式系统,功能和性能都取决于所选的 MCU。成本、低功耗、速度、工具链和加工都是需要考虑的因素。对于 MPU 设备,如果您计划使用 InvenSense Motion Driver 6.12 软件,这里有一些需要考虑的事项。

· 闪存和 RAM 大小: 闪存和 RAM 大小取决于代码优化、编译器、什么

您要使用的功能,以及系统中的其他组件。但总的来说,MD6.12

要求您可以需要保留以下数量的 Flash 和 RAM。请记住,这仅适用于运动驱动程序,不适用于其他可能的功能。

- o 16 位 MCU – 128K 和 12K

- o 32 位 MCU – 68K 和 10K (无优化,64K 和 8K 有优化)

同样,由于取决于大小非常依赖于编译器和编译器设置,客户应该花时间确定他们的应用程序需要多大的闪存和 RAM。

· Long long 数学支持: MPL 库需要支持 long long (64 位)数学。您需要确保您使用的是 MPL 库,您的工具链可以支持这一点。通常 8051 MCU 无法支持这样的数学计算。如果工具链不支持 long long 数学,您仍然可以使用 DMP 进行 6 轴融合。

· 中断: MPU 设备可以为来自低功耗手势识别或数据就绪中断的各种功能提供中断。虽然系统不需要使用 MPU 中断,但如果您打算使用它,则必须保留一个具有唤醒功能的 GPIO 引脚。

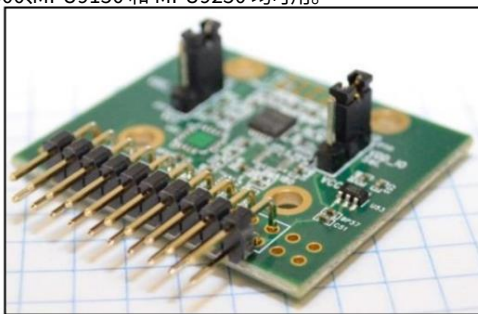
· 采样率: 传感器融合需要来自 MCU 的大量计算能力。这起到了

每个样本可以进行多少处理并限制您的采样率。例如,如果 MCU 正在执行完整的 9 轴融合,则带有运动驱动器的 TI 16 位 MSP430 应限制为 100Hz 采样率。任何超过 100Hz 采样率 MSP430 都会开始丢失数据。如果系统中不需要其他大型计算功能,高端 32 位 MCU 通常可以实现 200Hz 传感器融合。如果您将处理工作卸载到 DMP 上,则可以提高该采样率。

## 6 连接硬件

选择 MCU 后,您很可能会拥有一个 MCU 评估套件或您自己的 PCB 板。要将 MPU 设备连接到 MCU 板进行评估,您可以获得 InvenSense MPU 评估板

通过 InvenSense.com。MPU6050、MPU6500、MPU9150 和 MPU9250 均可用。



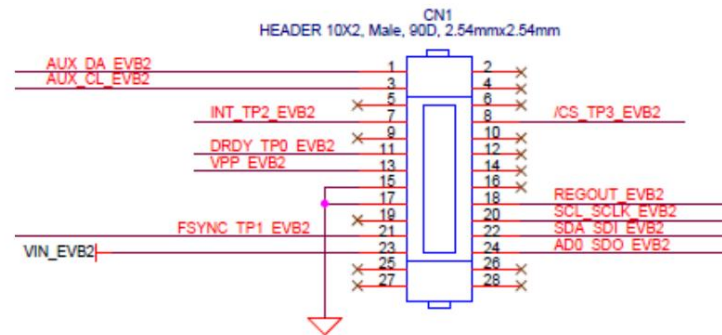
评估板



您需要将以下引脚从评估板连接到 MCU 板

- VDD 和 VDD\_IO (引脚 23) : 取决于 MPU 设备, 这将是 3V 或 1.8V (请参阅设备规格)
- SDA 和 SCL (引脚 20, 22) : I2C 引脚
- GND (引脚 15 或 17) : 接地
- INT (引脚 7) : 连接到 GPIO 以进行中断 (可选, 但如果使用 InvenSense 软件则需要)

InvenSense 评估板的引脚是相同的。



要确认您的硬件设置以及基本的 I2C 功能, 请先读取 MPU 设备的 whoami 寄存器并确认您获得了正确的设备 ID。对于 MPU 系列, I2C 地址为 0x68, 而不同部件的设备 ID 都不同, 因此请查看规格。硬件确认后, 我们可以继续进行软件集成。

## 7 运动驱动 6.12 固件包


Motion Driver 6.12 版本固件包含以下文件夹:

- core\driver : 此文件夹包含 MPU 设备的 InvenSense 驱动程序层以及 MCU 特定驱动程序
- simple\_apps\msp430\mllite\_test.c 或 src\main.c : 项目的主函数和主循环应用。客户可以将此代码作为参考, 将驱动程序功能集成到他们的项目中。
- core\mllite : 此文件夹包含 MPL 数据处理函数, 用于存储接收到的传感器数据并处理数据。
- core\mpl : 包含 InvenSense 专有 MPL 库 - 包含高级的库用于传感器融合和运行时校准的算法。
- core\eMPL-hal : 此文件夹包含提供相同传感器数据转换的文件, 例如欧拉角。

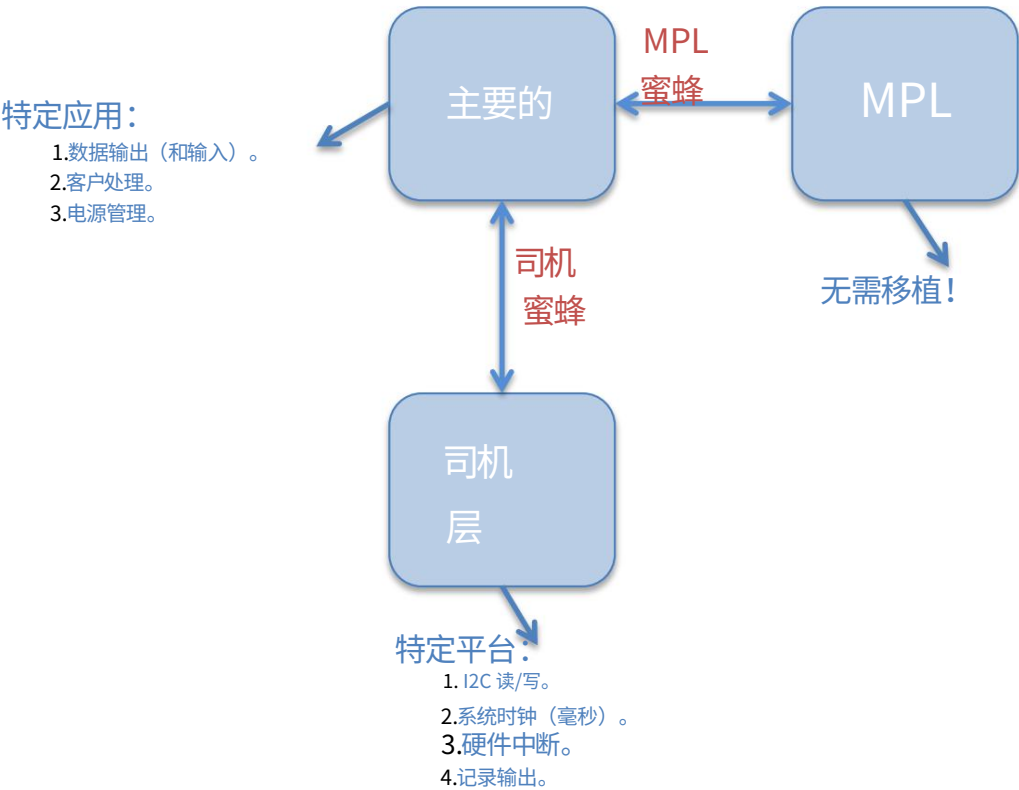
## 8 集成运动驱动器 6.12

Embedded MD6.12 由以下组件组成, 需要在目标硬件平台上集成:

- 1) 驱动程序
- 2) 运动处理库
- 3) 样本 HAL

	Motion Driver 6.12 – 用户指南	文件编号: AN-EMAPS-0.0.6 修订: 1.2 发布日期: 05/05/2015
---	---------------------------	---

下图显示了 Motion Driver 6.12 的架构以及将 MD6.12 成功移植到目标平台需要执行的任务。



1. Invensense MD6.12 Driver Layer (core\driver\eMPL) 由这些文件组成

- inv\_mpu.c · - 可轻松移植到不同嵌入式平台的驱动程序。
- inv\_mpu.h · - 包含 InvenSense 驱动程序的结构和原型。
- inv\_mpu\_dmp\_motion\_driver.c - 用于包含 dmp 图像和要加载的 API 的驱动程序并配置 DMP。
- inv\_mpu\_dmp\_motion\_driver.h - 包含 DMP 功能的原型和定义
- 位置的定义 - 包含 DMP 功能的 DMP 内存位置的定义 · dmpKey.h · dmpmap.h - 包含 DMP 内存

用户需要提供以下 API 来支持 I2C 读/写功能、系统时钟访问、硬件中断回调和与移植 MD6.12 的平台相对应的日志记录。

这些函数需要在 inv\_mpu.c 和 inv\_mpu\_dmp\_motion\_driver.c 中定义。下面是 MSP430 平台的示例。

```
#define i2c_write      msp430_i2c_write
#define i2c_read      msp430_i2c_read
#define delay_ms      msp430_delay_ms
#define get_ms        msp430_get_clock_ms
#define log_i #define MPL_LOGI
log_e                MPL_LOGE
```



i2c\_write和i2c\_read :这需要链接到 i2c 驱动程序。此功能将占用 4 参数然后执行 i2c 事务

`unsigned char slave_addr`

`unsigned char reg_addr`

无符号字符长度

无符号字符\*数据

delay\_ms :这个函数将接受一个无符号长参数,它将作为毫秒的延迟为系统

get\_ms :get\_ms 主要用于获取当前时间戳。时间戳通常是无符号长并以毫秒为单位。此功能将主要用于指南针调度程序以及传感器融合数据的附加信息。

log\_i和log\_e :MPL 消息传递系统,它可以在其中记录信息或错误消息。当前的实现将消息打包并通过 USB 或 UART 发送出去,供 python 客户端接收。日志记录代码位于文件 log\_msp430.c 或 log\_stm32l.c 中。客户可以根据自己的喜好更改传输方式和数据包。

2. MPL 库是 InvenSense Motion Apps 专有算法的核心,由 Mllite 和 mpl 目录组成。MPL不需要移植。您可能需要包含系统特定的头文件以支持 mllite 包中的 memcpy、memset 等函数调用。MD6.12 包将包含预编译的 MPL 库,一个用于 TI MSP430 平台,另一个用于 ARM 内核平台。ARM 库有一个通用库,可以链接到任何 ARM MCU。它还为 M3 和 M4 进行了一些预编译,以进行更好的优化。

3. eMPL-HAL 目录包含从 MPL 库中获取各种数据的 API。你能得到的数据获取来自以下 API

- `int inv_get_sensor_type_accel(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_gyro(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_compass(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_quat(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_euler(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_rot_mat(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_heading(long *data, int8_t *accuracy, inv_time_t *时间戳);`
- `int inv_get_sensor_type_linear_acceleration(float *values, int8_t *accuracy, inv_time_t *时间戳);`

4. main.c 或 mllite\_test.c 包含一个非常具体的应用程序,

- 来自 MPU 设备的传感器数据处理句柄
- 处理来自客户端的输入请求
- 电源管理
- 初始化 MPL 库、DMP 和硬件
- 处理中断





## 9 初始化 API

在 MPU 设备上电时,它将在默认状态下提供传感器数据。inv\_mpu.c 提供了一个参考 API,说明如何使用一些基本配置初始化 MPU 设备,例如打开传感器电源和设置比例范围和采样率

```
· int mpu_init(struct int_param_s *int_param) · int
mpu_set_gyro_fsr(unsigned short fsr) · int mpu_set_accel_fsr(unsigned
char fsr) · int mpu_set_lpf(unsigned short lpf) · int
mpu_set_sample_rate(unsigned short rate) · int
mpu_set_compass_sample_rate(unsigned short fsr) int
mpu_configure_fifo (无符号字符传感器) · int mpu_set_sensors (无符号字符传感器)
```

## 10 方向矩阵

应用程序还需要为 MPU 设备和第3方指南针 (如果存在于平台上)定义方向矩阵。方向矩阵会将物理硬件传感器轴重新配置为设备坐标。错误的配置将使您从传感器数据中获得不准确的结果。有关如何定义矩阵的更多信息,请参考Orientation Matrix Transformation chart.pdf。

矩阵将被推送到 MPL 库和 DMP 中以进行融合计算。

```
结构平台_数据_s {
    有符号字符方向[9]; };

/* 传感器可以以任何方向安装到板上。安装
   下面看到的矩阵告诉 MPL 如何从
   * 驱动程序。
   */ 静态结构
platform_data_s gyro_pdata = {
    .orientation = {-1, 0, 0,
                   0,-1, 0,
                   0, 0, 1}
};

静态结构 platform_data_s compass_pdata = { #ifdef
MPU9150_IS_ACTUALLY_AN_MPU6050_WITH_AK8975_ON_SECONDARY
    .orientation = {-1, 0, 0,
                   0, 1, 0,
                   0, 0,-1}

#else .orientation = { 0,
                      1, 0, 1,
                      0, 0, 0, 0,-1}

#万一 }
```



## 11 中断处理

MPU 设备有一个中断输出引脚。中断可以被编程为产生在要么

- FIFO 输出率
- DMP 生成

通常,当 FIFO 中有新的传感器数据可用时,我们会生成一个中断。DMP 也可以编程为在检测到手势时产生中断。

如果您使用 MD6.12 参考示例,当产生传感器数据就绪中断时,中断例程将全局标志new\_gyro设置为 1。在主循环中它将知道有一组新的传感器数据进行处理。

以下是与中断相关的 API 列表

- `int dmp_set_interrupt_mode` (无符号字符模式)
- `static int set_int_enable(unsigned char enable)`

## 12 DMP -数字运动处理器™

MPU-9150、MPU-6050、MPU-9250 和 MPU-6500 均具有嵌入式 Digital Motion Processor™ (DMP) 硬件加速器引擎。DMP 与嵌入式 FIFO 一起,从主机应用处理器中卸载高频运动算法计算,减少中断和主机 MIPS,从而提高整体系统性能。

所有相关的 DMP API 和固件都可以在 `inv_mpu_dmp_motion_driver.dmpKey.h` 和 `dmpMap.h` 中找到。

### 12.1 DMP 初始化

DMP 固件代码是在结构中找到 3kB 图像

静态常量无符号字符 `dmp_memory[DMP_CODE_SIZE]`

该图像需要下载到 DMP 存储库中。下载后需要提供起始地址,然后需要打开 DMP 状态。与 DMP 初始化相关的 API 如下

- `int dmp_load_motion_driver_firmware(void)`
- `int dmp_load_motion_driver_firmware(void)`
- `int dmp_set_fifo_rate` (无符号短率)
- `int mpu_set_dmp_state(unsigned char enable)`

可以在主循环之前的主函数中找到有关 DMP 初始化的 MD6.12 示例。

### 12.2 DMP 特性

DMP 具有许多功能,详见功能指南。这些功能可以动态启用和禁用。主要 API 是

- `int dmp_enable_feature` (无符号字符掩码) ;

此函数将掩码和索引放入 DMP 固件中的正确内存地址,以启用和禁用该功能。特点是



```

· #define DMP_FEATURE_TAP (0x001)
· #define DMP_FEATURE_ANDROID_ORIENT (0x002)
· #define DMP_FEATURE_LP_QUAT · (0x004)
#define DMP_FEATURE_PEDOMETER (0x008)
· #define DMP_FEATURE_6X_LP_QUAT · (0x010)
#define DMP_FEATURE_GYRO_CAL (0x020)
· #define DMP_FEATURE_SEND_RAW_ACCEL (0x040)
· #define DMP_FEATURE_SEND_RAW_GYRO (0x080)
· #define DMP_FEATURE_SEND_CAL_GYRO (0x100)

```

对于 Tap 和 Orientation 数据解析,MD6.12 驱动程序定义了 2 个回调函数,它们将处理解析并将其记录到 python 客户端。回调将需要定义 MD6.12 驱动程序。相关 API 是

```

· int dmp_register_tap_cb(void (*func)(unsigned char, unsigned char))
· int dmp_register_android_orient_cb(void (*func)(unsigned char))
· static int decode_gesture(unsigned char *gesture)
· static void tap_cb (无符号字符方向,无符号字符计数)
· static void android_orient_cb (无符号字符方向)

```

Tap 还有一些可配置的设置,例如阈值。这些 API 在 inv\_mpu\_dmp\_motion\_driver 中可用。

### 12.3 DMP FIFO 输出

DMP 仅在启用特定功能 (例如抽头或传感器数据)时写入 FIFO。 MD6.12 驱动程序将等待 DMP 产生中断,然后读取 FIFO 的内容。

FIFO 格式取决于启用了哪些 DMP 功能。在 API 函数中可以看到 DMP FIFO 输出格式。

```

· int dmp_read_fifo(short *gyro, short *accel, long *quat,
    unsigned long *timestamp, short *sensors, unsigned char *more);

```

## 13 InvenSense 硬件自检

硬件自检是一种可选的工厂生产线测试,客户可以将其用作生产线上的通过/不通过测试。 HWST 算法将测试 MEMS 传感器并通过内部移动和测量移动并将输出与保存在其寄存器中的 InvenSense 数据进行比较来确认工作功能。有关更多详细信息,请查看产品规格。


MD6.12 代码提供了有关如何运行 HWST 及其输出的示例代码。硬件自检可以在不与 MPL 进行任何交互的情况下运行,因为它完全本地化在驱动程序中。运行完整自检的 API 是

```

·静态内联 void run_self_test(void)

```

MD6.12 将自检和工厂校准捆绑在一起,因为传感器偏移是通过正常的自检例程计算的。但是,如果客户愿意,可以将校准和自检分开。

	Motion Driver 6.12 – 用户指南	文件编号 :AN-EMAPS-0.0.6 修订 :1.2 发布日期 :05/05/2015
---	---------------------------	---

与 MPU6500/MPU9250 相比,MPU6050/MPU9150 具有不同的自检算法。API 返回传感器各轴的状态以及加速度和陀螺仪偏差进行校准

- `int mpu_run_6500_self_test(long *gyro, long *accel, unsigned char debug)`
- `int mpu_run_self_test(long *gyro, long *accel)`

自检功能参数如下所示

范围 类型	参数名称	
输出	result	该函数返回自检结果,如下表所示。
输入/输出	accel	返回加速度偏差。
输入/输出	陀螺仪	返回陀螺仪偏差。
输入	调试自检的	额外日志。默认为 0

results 的返回值定义如下,带有 1

价值	传感器状态
0x01	陀螺仪传感器状态
0x02	加速度传感器状态
0x04	罗盘传感器状态

如果返回的值不是 0x07,则表示特定传感器发生故障。

以下是从 python 脚本启动的自检通过输出的示例。

```
$ python eMPL-client.py 79
Passed!
accel: 0.0194 0.0121 0.0152
gyro: -3.2084 0.780 -0.4681
```

如果自检失败,如果自检命令是从 python 脚本启动的,那么失败的传感器将显示在 python 脚本窗口中。

14 校准数据和存储

校准数据包含描述 MPU 陀螺仪、加速度计和罗盘的固有偏差和温度相关行为的信息。在 MPL 执行期间使用此数据来提高 MPL 返回的结果的准确性。校准数据可能会随着时间、温度和环境缓慢变化,因此 InvenSense 提供了几种使用中的传感器校准算法,这些算法将在传感器的整个生命周期内不断校准传感器。详细信息在功能指南中进行了描述。建议在出厂时校准 MPU 传感器加速度和陀螺仪,如果使用 MPL 库打开使用中的算法。

14.1 工厂线校准

自检功能返回的加速度和陀螺仪偏差可用于工厂校准,并可被 HAL 保存并用于校准传感器的性能。偏置可以被推送到硬件偏移寄存器或 MPL 库中。



默认情况下,MD6.12 将偏差推送到 MPL 库中,并让融合引擎应用偏差数据。

然而,客户可以通过定义使用硬件偏移寄存器

- USE\_CAL\_HW\_REGISTERS

在 main.c,不同之处在于,如果使用硬件偏移寄存器,MEMS 数据将在被推入传感器数据寄存器之前自动调整。要更好地了解硬件偏移寄存器,请参阅应用说明“MPU 硬件偏移寄存器”。

重要的是,当您进行工厂生产线校准时,设备需要处于稳定且无振动的环境中,物理 Accel Z+ 朝上或朝下,这将是 MPU IC 的面需要朝上或朝下。

## 14.2 保存和加载校准数据

MPL 不会自动生成、加载或存储校准数据。在计算并应用偏差后,一旦设备断电,它就会丢失。因此,InvenSense 提供了有关如何从内存位置保存和加载校准数据的 API 示例。请看功能

- `inv_error_t inv_save_mpl_states(unsigned char *data, size_t sz)`
- `inv_error_t inv_load_mpl_states(const unsigned char *data, size_t length)`

客户可以使用这些功能作为如何保存到其设备内存中的示例。出厂标定后,断电前应保存标定。开机后需要重新加载。

## 15 集成 MPL 库

MPL 库是一个包含传感器融合引擎的预编译库。移植 MD6.12 时,库需要与集成器系统兼容。MD6.12 带有 2 个库。

- TI MSP430 - 使用 Code Composer 编译。应与所有 MSP430 产品线兼容
- ARM – MD6.12 带有许多用于 ARM 的预编译库。具体有 IAR、Keil 和 GNU 4.9.3 编译的库。每个编译器都使用 M0、M0+、M3、M4 和 M4F 的特定设置来编译库。

链接库后,代码将需要启用库及其功能。库初始化可以在主循环之前的主函数中找到。功能指南中描述了这些功能。

以下是相关的 API

- `inv_error_t inv_init_mpl(void)`
- `inv_error_t inv_enable_quaternion(void)` // 启用 6 轴
- `inv_error_t inv_enable_9x_sensor_fusion(void)` // 启用 9 轴融合
- `inv_error_t inv_enable_fast_nomot(void)` // 陀螺仪在用校准
- `inv_error_t inv_enable_gyro_tc(void)` // 陀螺温度补偿
- `inv_error_t inv_enable_vector_compass_cal(void)` // 罗盘校准
- `inv_error_t inv_enable_magnetic_disturbance(void)` // 磁干扰
- `inv_error_t inv_enable_eMPL_outputs(void)`
- `inv_error_t inv_start_mpl(void)`



## 16 MPU6500/MPU9250 的低功耗加速模式和运动中中断模式

LPA 模式和运动中中断模式类似,可以通过设备启用,只需要加速数据。

MPU6050/MPU9150 不支持此功能。

此功能要求禁用 DMP、FIFO 和陀螺仪。然后它将睡眠循环加速,仅以用户请求的指定速率唤醒它。不同的 LPA 速率从 1.25Hz 到 640Hz。速率越低,消耗的功率就越低。以最低速率,总功率约为 10uA。

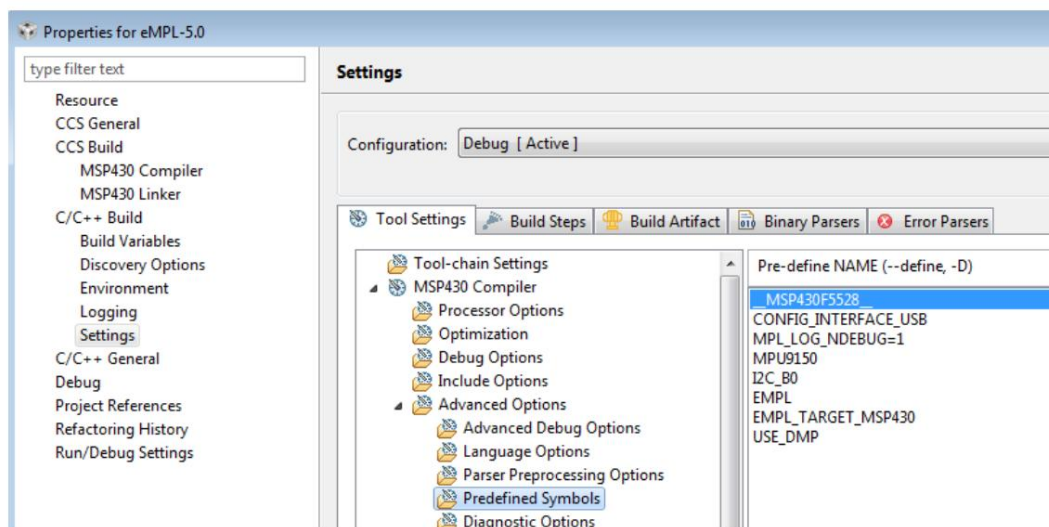
客户还可以将设备置于运动中中断模式。在这种模式下,设备将处于 LPA 模式,如果加速度数据超过某个阈值,它将产生中断。如果没有运动并且客户想要让设备休眠直到检测到运动,这将特别有用。

相关的 API 是 –

- `int mpu_lp_accel_mode` (无符号短速率)
- `int mpu_lp_motion_interrupt` (无符号短阈值,无符号字符时间,无符号短 lpa\_freq)

## 17 编译器特定设置

要编译不同的部分 (MPU6050、MPU9150、MPU6500 和 MPU9250),您需要设置编译器标志



所需的默认符号是

- `MPL_LOG_NDEBUG=1`
- `MPU9150` 或 `MPU6050` 或 `MPU6500` 或 `MPU9250`
- `EMPL`
- `USE_DMP`
- `EMPL_TARGET_MSP430` 或其等效项

一旦设置了部件,编译器将针对该特定部件及其功能进行编译。