InvenSense	**InvenSense Inc.** 1197 Borregas Ave., Sunnyvale, CA 94089 U.S.A. Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104 Website: www.invensense.com	文件编号：AN-EMAPPS-0.0.6 修订版本：1.1 发行日期：05/05/2015

Motion Driver 6.12-用户指南

（官方手册翻译）

说明：翻译了全部内容。能力有限，个别地方采用原文没进行翻译，翻译有可能错误与不通顺，建议查看英语原文。对于翻译错误或者有好的修改意见欢迎发送电子邮件到 **2402452433@qq.com**，以便更新。

By：隐形原子

目录

1 修订历史

修订时间	修订版本	说明
06/27/2014	1.0	初始发行
07/17/2014	1.1	扩展ARM MPL库的信息
05/15/2015	1.2	更新为MD 6.12版本

2 目的

Motion Driver（运动驱动器）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）的许多功能都封装在可以使用和引用的模块化API（Application Programming Interface,应用程序编程接口）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，它包含一个用于ARM MCU和TI-MSP430的9轴解决方案。6轴解决方案应继续参考Motion Driver 5.1.2，以便于更容易了解软件。

本文档突出显示了使用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和选择。我们将深入介绍一些更详细的主题，如编程DMP，校准和自检。

3 在开始之前

请阅读 Motion Driver 6.12 Getting Started Guide 和 the Motion Driver 6.12 Features Guide。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，以便更好地了解代码和功能。了解功能之后，将会更容易将其移植到您的生态系统中。

4 Motion Driver 6.12 功能

这是对 MD6.12 功能的概述。

- DMP 功能：
  - 3/6 轴低功耗四元数
  - 点击，方向和计步器手势检测
- MPL 算法：
  - 运行时陀螺仪校准
  - 运行时陀螺仪温度补偿
  - 运行时罗盘校准
  - 运行时磁阻抑制
  - 3/6/9 轴传感器融合
- 硬件功能：
  - 出厂校准
  - 工厂自检
  - 保存和加载传感器状态
  - 低功耗加速模式
  - 低功耗运动中断模式
  - 寄存器转储

5 选择 MCU

对于每个嵌入式系统，功能和性能取决于所选择的 MCU。成本，低功耗，速度，工具链和处理都是要考虑的因素。对于 MPU 设备，如果您计划使用 InvenSense Motion Driver 6.12 软件，则需要考虑一些事项。

Flash and RAM Size：闪存和RAM大小取决于代码优化，编译器，要使用的功能以及系统中的其他组件。一般来说，MD6.12需要您可以预留以下容量的Flash和RAM。请记住，这只是运动驱动程序，而不是其他可能的功能。

- 16 位 MCU - 128K 和 12K
- 32 位 MCU - 68K 和 10K（无优化，64K 和 8K 优化）

再次，由于依赖大小非常依赖于编译器和编译器设置，客户应该花时间来确定应用程序需要多大的闪存和 RAM。

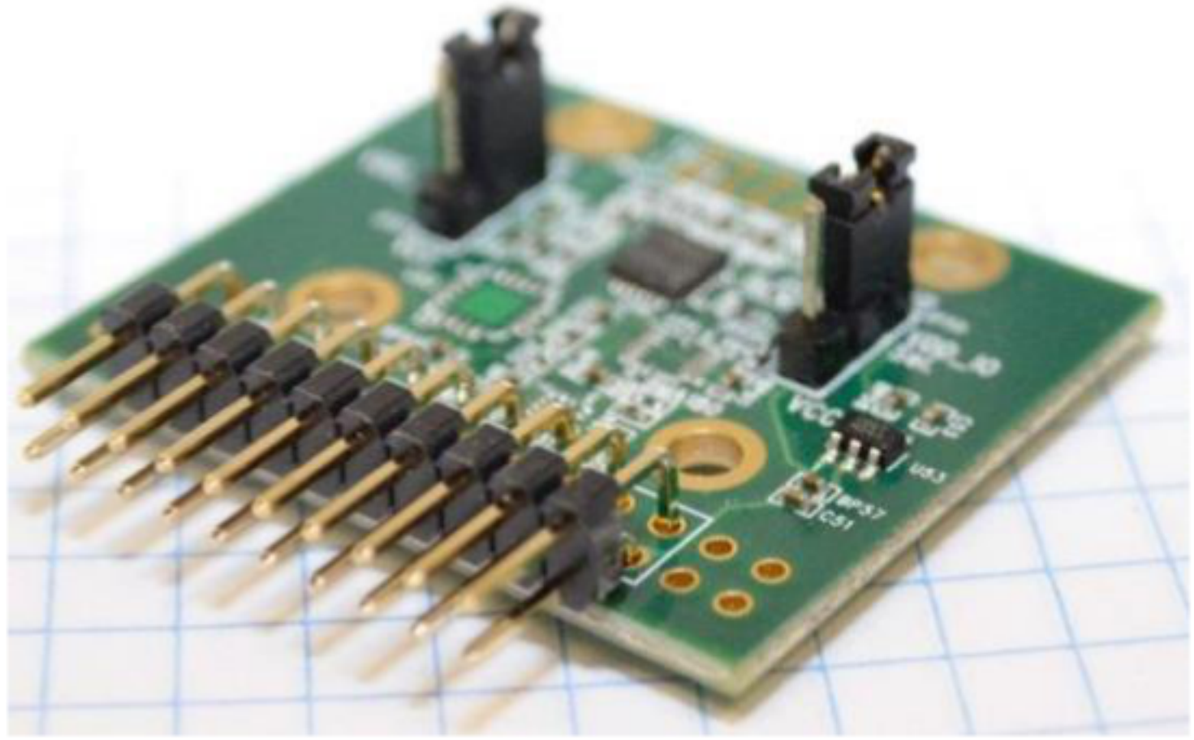
Long long math support: MPL库需要长时间（64位）的数学支持。您将需要确保您是否使用MPL库，您的工具链可以支持此功能。通常8051 MCU不能支持这种数学计算。如果工具链不支持长时间的数学运算，您仍然可以使用DMP获得6轴融合。

Interrupts: MPU设备可以为低功耗手势识别或数据就绪中断提供各种功能的中断。虽然系统不需要使用MPU中断，如果您打算使用它，则必须保留具有唤醒功能的GPIO引脚。

Sampling Rate：传感器融合需要来自MCU的大量计算能力。这样可以对每个样品进行多少处理，并限制采样率。例如，如果MCU进行完整的9轴融合，则具有运动驱动器的TI 16位MSP430应限制为100Hz采样率。任何超过100Hz采样率的MSP430启动丢失数据。如果系统中没有其他大的计算功能，高端32位MCU通常可以实现200Hz传感器融合。如果将处理卸载到DMP上，则可以增加此采样率。

6 连接硬件

选择单片机后可能你会有一个单片机评测套件或自己的 PCB 板。为了把 MPU 设备连接到 MCU 板用来评测，你可以在 InvenSense.com 获得一个 InvenSense MPU 评测板。MPU6050, MPU6500, MPU9150, and MPU9250 都可以得到。

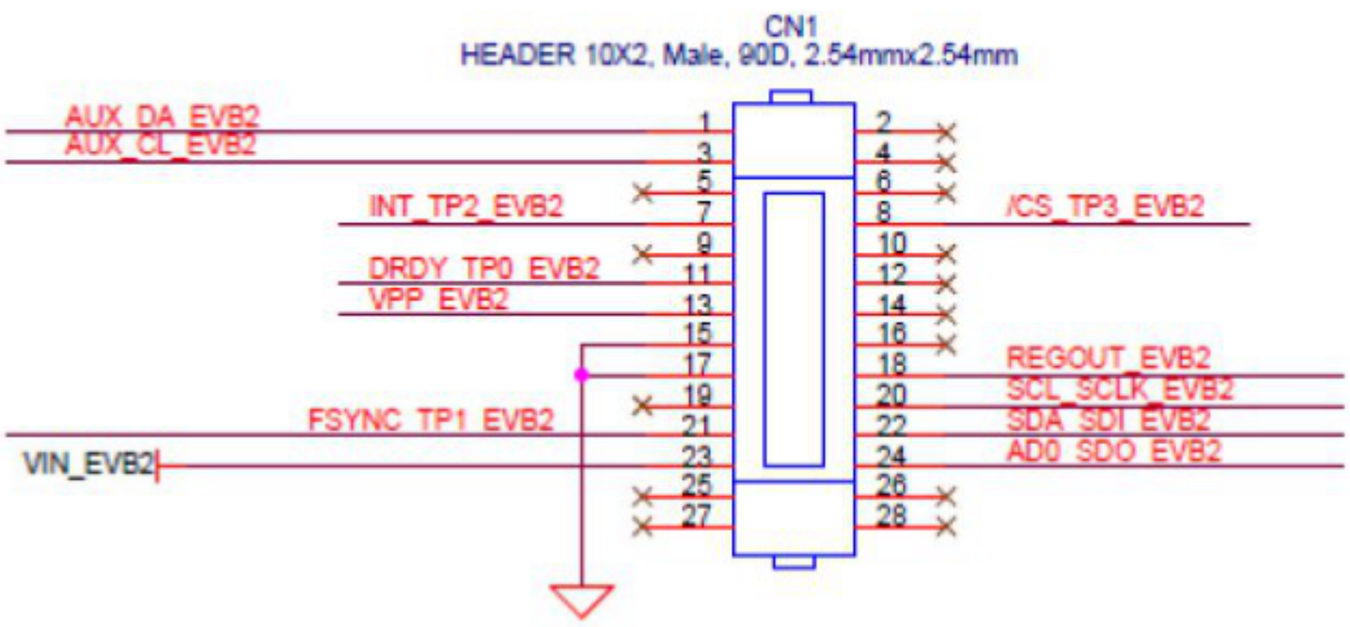


评测板

您将需要把评测板的以下引脚连接到单片机

- VDD and VDD_IO (pin 23): 根据MPU设备，这个连到 3V 或者1.8V (视MPU说明书连接)
- SDA and SCL (pins 20, 22): I2C 引脚
- GND (pins 15 or 17): 接地
- INT (pin 7): 连接GPIO用于中断 (选用，但是使用InvenSense 软件时必须连接)

InvenSense 评测板引出的引脚是相同的



确认硬件设置和基本的 I2C 功能，首先阅读 MPU 的 who am i 寄存器，并确认你得到正确的设备 ID。对于 MPU 系列，I2C 地址是 0x68，而器件 ID 对于不同的部件都是不同的，因此请阅读说明书。硬件确认后，我们可以继续进行软件集成。

7 Motion Driver 6.12 固件包

Motion Driver 6.12 发行固件包有以下文件夹：

- **core\driver**：这个文件夹有MPU设备的 InvenSense 驱动层和 MCU 具体的驱动
- **simple_apps\msp430\mllite_test.c or src\main.c**：项目的主要功能和主循环程序。客户可以使用这段代码作为将驱动功能集成到他们的项目的参考。
- **core\mllite**：这个文件夹有 MPL 数据处理功能，这个功能是存储接收到的传感器数据和处理数据。
- **core\mpl**：这个文件夹有InvenSense 专有的 MPL 库 – 一个有传感器融合和运行时校准的先进算法的库。
- **core\eMPL-hal**：这个文件夹的文件提供相同的传感器数据转换，如欧拉角。

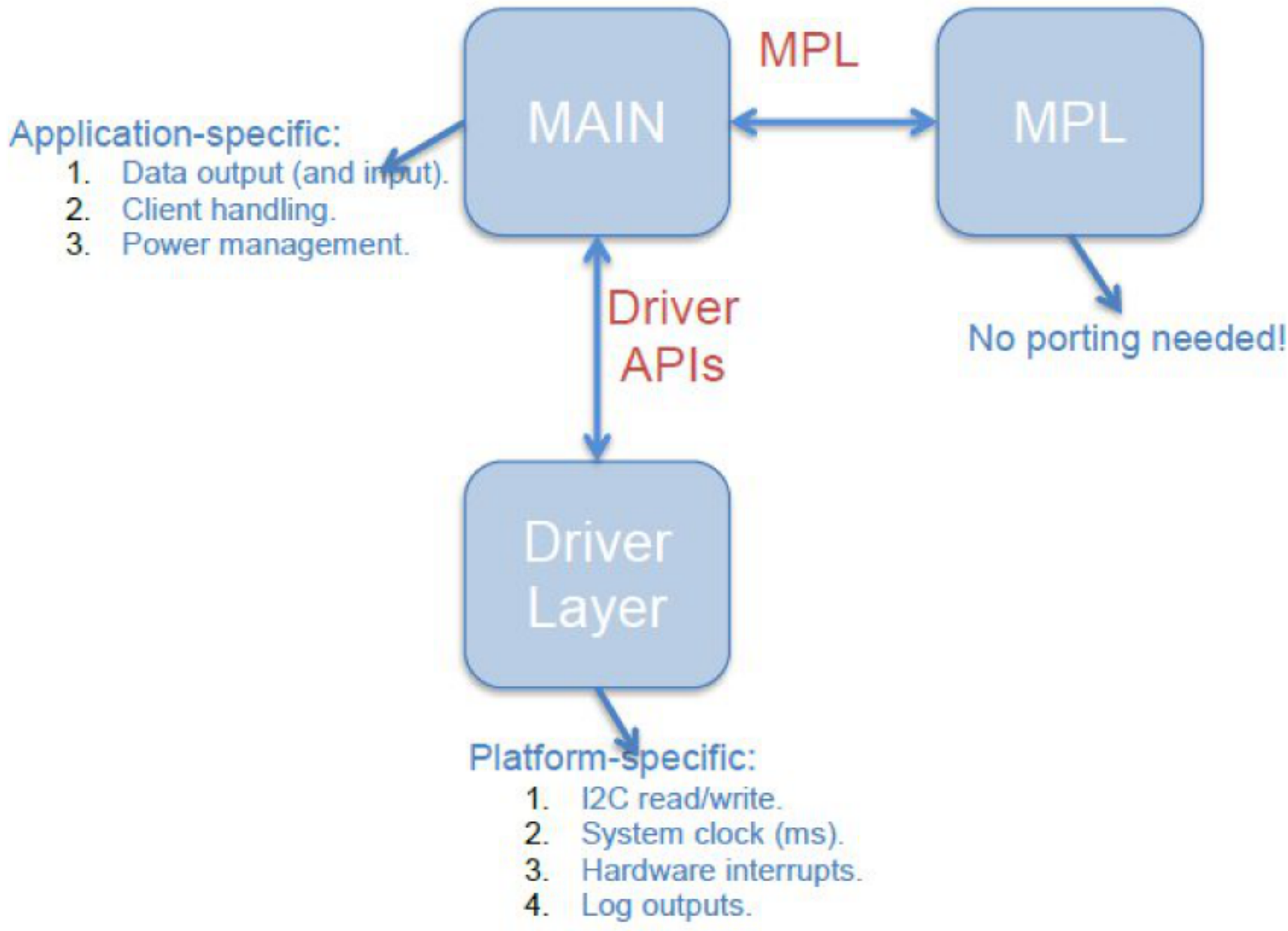
8 整合 Motion Driver 6.12

嵌入式 MD6.12 包含如下组件，这些组件需要整合到目标硬件平台上：

1) Driver
2) Motion Processing Libraries

3) Sample HAL

下图显示了 Motion Driver 6.12 的体系结构和将 MD6.12 成功移植到目标平台要做的工作。



1. Invensense MD6.12 驱动层（core\driver\eMPL）包含如下文件：

- inv_mpu.c – 此驱动可以很容易的移植到不同的嵌入式平台。
- inv_mpu.h – 包含 InvenSense 驱动的结构体和原型。
- inv_mpu_dmp_motion_driver.c – 此驱动包含 DMP 映像和要加载的 API 并配置 DMP。
- inv_mpu_dmp_motion_driver.h – 包含原型并定义了 DMP 功能
- dmpKey.h – 包含 DMP 功能的存储单元的定义。
- dmpmap.h – 包含 DMP 存储单元的定义。

用户需要提供以下 API 来支持 I2C 读/写功能，系统时钟访问，硬件中断回调和日志记录的对应 MD6.12 移植的平台。

这些功能需要在 inv_mpu.c 和 inv_mpu_dmp_motion_driver.c 中定义。下面是一个 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           MPL_LOGI
#define log_e           MPL_LOGE
```

i2c_write, **i2c_read**：这将需要连接到i2c驱动。这个函数带有4个参数，然后执行i2c事务。

- `unsigned char slave_addr`
- `unsigned char reg_addr`
- `unsigned char length`
- `unsigned char *data`

delay_ms：这个函数带有一个unsigned long参数， 作为系统的毫秒延时。

get_ms：get_ms主要用于获取当前时间戳。时间戳通常是一个无符号长量，以毫秒为单位。这个函数也将主要用于指南针调度器和传感器融合数据的附加信息。

log_i 和 **log_e**：MPL的消息传递系统可以记录信息或错误消息。当前实现了消息打包和通过USB或者UART发送，从而被python客户端接收。日志代码位于log_msp430.c或log_stm32l.c文件。客户可以按自己的喜好改变运输方式和打包。

2. MPL 库是 InvenSense Motion Apps 算法的专有核心，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来支持 Mllite 包中的 memcpy，memset 等功能的调用。MD6.12 包将包括预编译的 MPL 库，一个用于 TI MSP430 平台，另一个用于 ARM 核心平台。ARM 库有一个可以连接到任何 ARM MCU 的通用库。它还有一些预编译的 M3 和 M4 的库来进行更好的优化。

3. eMPL-HAL目录包含用于从MPL库获取各种数据的API，您可以获得的数据来自以下API：

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

4. main.c 或 mllite_test.c 文件包含一个非常特殊的应用程序

- 来自 MPU 设备的传感器数据处理的处理
- 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

上电时 ,MPU 设备将以默认状态提供传感器数据。inv_mpu.c 提供了一个如何初始化 MPU 设备的参考 API（Application Programming Interface，应用程序编程接口），它具有一些基本配置，例如为传感器供电，并设置量程和采样率

- `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 rate)`
- `int mpu_configure_fifo(unsigned char sensors)`
- `int mpu_set_sensors(unsigned char sensors)`

10 方向矩阵

应用程序还需要定义 MPU 设备的方向矩阵和平台上存在的第三方罗盘。方向矩阵将物理硬件传感器轴重新配置到设备坐标。错误的配置可能会导致传感器数据不准确的结果。有关如何定义矩阵的更多信息，请参考 Orientation Matrix Transformation chart.pdf。矩阵将被推入 MPL 库和 DMP 以进行融合计算。

```
struct platform_data_s {
  signed char orientation[9];
};

/* The sensors can be mounted onto the board in any orientation.
The mounting
* matrix seen below tells the MPL how to rotate the raw data from
the
* driver(s).
*/
static struct platform_data_s gyro_pdata ={
     .orientation = {-1, 0, 0,
                     0,-1, 0,
                     0, 0, 1}
};

static struct 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}
```

```
        #endif
    }
```

11 中断处理

MPU 设备有一个中断输出引脚，中断可以通过编程产生在以下两个地方

- FIFO 输出速率
- 生成 DMP

通常，当 FIFO 中有新的传感器数据就绪时，我们会产生中断。当检测到手势时，也可以将 DMP 编程为产生中断。

如果使用 MD6.12 参考示例，当生成传感器数据就绪中断时，中断程序将全局标志 new_gyro 设置为 1。在主循环中，将知道有一组新的传感器数据要进行处理。

以下是与中断相关的 API 列表

- `int dmp_set_interrupt_mode(unsigned char mode)`
- `static int set_int_enable(unsigned char enable)`

12 DMP-数字运动处理器（Digital Motion Processor™）

MPU-9150， MPU-6050，MPU-9250 和 MPU-6500 均采用嵌入式数字运动处理器（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 和固件位于 inv_mpu_dmp_motion_driver，dmpKey.h 和 dmpMap.h 中。

12.1 DMP 初始化

DMP 固件代码是建在下列结构体中的 3 KB 映像

- `static const unsigned char dmp_memory[DMP_CODE_SIZE]`

该映像需要下载到 DMP 存储器中。下载开始后需要提供下载起始地址，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(unsigned short rate)`
- `int mpu_set_dmp_state(unsigned char enable)`

DMP 初始化中的 MD6.12 示例可以在主循环之前的主函数中找到。

12.2 DMP 功能

DMP 具有“Features Guide.PDF”中列出的许多功能。这些功能可以动态启用和禁用。主要 API 是

- `int dmp_enable_feature(unsigned char mask);`

此功能将掩码和索引置入 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（点击）和 Orientation（方向）数据解析，MD6.12 驱动程序定义了两个回调函数，它们将处理解析并将其记录到 python 客户端。回调需要定义在 MD6.12 驱动程序中。相关的 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(unsigned char direction, unsigned char count)`
- `static void android_orient_cb(unsigned char orientation)`

Tap 还有一些可配置的设置，如阈值。这些 API 在 inv_mpu_dmp_motion_driver 中可以看到。

12.3 DMP FIFO 输出

当启用特定功能时，DMP 只会写入 FIFO，例如 Tap 或传感器数据。MD6.12 驱动程序将等待 DMP 产生中断，然后读取 FIFO 的内容。

FIFO 格式取决于哪些 DMP 功能被启用.DMP FIFO 输出格式可以在 API 函数中看到。

- `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线测试，客户可以在其生产线上用 go/no-go 测试。HWST 算法将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 数据进行比较来测试 MEMS 传感器并确认工作功能。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。硬件自检可以与 MPL 不进行任何交互，因为它完全在本地化在驱动程序中。运行完整自检的 API 是

- `static inline void run_self_test(void)`

MD6.12 将自检和出厂校准捆绑在一起，因为传感器偏移量通过正常的自检程序计算。但是，如果客户愿意，客户可以分开校准和自检。

与 MPU6500 / MPU9250 相比，MPU6050 / MPU9150 具有不同的自检算法。API 返回传感器的每个轴的状态以及用于校准的加速度和陀螺仪偏置

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

自检功能的参数如下所示

参数类型	Parameter Name参数名	
output	result	该函数返回自检结果，如下表所示。
I/O	accel	返回加速度偏差
I/O	gyro	返回陀螺仪偏差
Input	debug	额外的自我测试记录。 默认值为0

'results'的返回值定义如下，'1'

Value	Sensor Status 传感器状态
0x01	Gyro Sensor Status
0x02	Accel Sensor Status
0x04	Compass Sensor status

如果返回的值不是 0x07，则表示特定传感器失败。

以下是从 python 脚本启动的自检的传递输出的示例。

```
$ python eMPL-client.py 79
Passed!
accel:  0.0194  00121  0.0152
gyro: -3.2084  0.780 -0.4681
```

如果自检失败，则如果从 python 脚本启动自检命令，则会在 python 脚本窗口中显示失败的传感器。

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与温度相关的行为的信息。在 MPL 执行期间使用此数据来提高 MPL 返回的结果的准确性。校准数据可能会随时间，温度和环境而变化缓慢，因此 Invensense 提供了多种使用中的传感器校准算法，可在其使用寿命内不断校准传感器。功能手册中介绍了详细信息。 建议在工厂线上对 MPU 传感器加速度和陀螺仪进行校准，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出厂校准，可由 HAL 保存，用于校准传感器的性能。偏移可以被推送到硬件偏置寄存器或 MPL 库中。

默认情况下，MD6.12 将偏置推送到 MPL 库中，让融合引擎应用偏差数据。然而，客户可以通过定义来使用硬件偏移量寄存器

- USE_CAL_HW_REGISTERS

在 main.c，不同的是，如果使用 HW 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。 要了解 HW 偏置寄存器 ,请参见应用笔记“MPU HW 偏置寄存器”。重要的是，当您进行工厂线校准时，器件需要处于稳定和无振动的环境中，物理 Accel Z + 面向上或向下，这将使 MPU IC 的面向上或向下。

14.2 保存和加载校准数据

校准数据不会由 MPL 自动生成、加载或存储。计算和应用偏置后，一旦设备断电就会丢失。因此，InvenSense 提供了有关如何从存储器位置保存和加载校准数据的 API 示例。

请查看功能

- `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设备内存中的示例。在出厂校准后，断电前应保存校准。上电时，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，库需要与集成系统兼容。MD6.12 带有 2 个库。

- TI MSP430 - 使用 Code Composer 编译。应兼容所有 MSP430 产品线。
- ARM - MD6.12 带有许多用于 ARM 的预编译库。有具体的 IAR，Keil 和 GNU 4.9.3 编译库。每个编译器使用 M0，M0+，M3，M4 和 M4F 的特定设置编译库。

库链接后，代码将需要使能库和它的功能。库初始化可以在主循环之前的 main 函数中找到，功能在功能手册中有描述。以下是相关得 API

- `inv_error_t inv_init_mpl(void)`
- `inv_error_t inv_enable_quaternion(void) //enable 6-axis`
- `inv_error_t inv_enable_9x_sensor_fusion(void) //enable 9-axis fusion`
- `inv_error_t inv_enable_fast_nomot(void) //gyro in-use calibration`
- `inv_error_t inv_enable_gyro_tc(void) //gyro temperature compensation`
- `inv_error_t inv_enable_vector_compass_cal(void) //compass calibration`
- `inv_error_t inv_enable_magnetic_disturbance(void) //magnetic disturbance`
- `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断模式

LPA 模式和运动中断模式相似，可以使能设备只需要加速数据。MPU6050 / MPU9150 不支持此功能。

该功能要求禁用 DMP，FIFO 和陀螺仪。然后，它将按照用户请求的指定速率进行睡眠循环。不同的 LPA 速率为 1.25Hz 至 640hz。较低的功率将消耗的速率越低。以最低的速度，总功率将在 10uA 左右。

客户还可以将设备置于运动中断模式。在此模式下，器件将处于 LPA 模式，如果加速度数

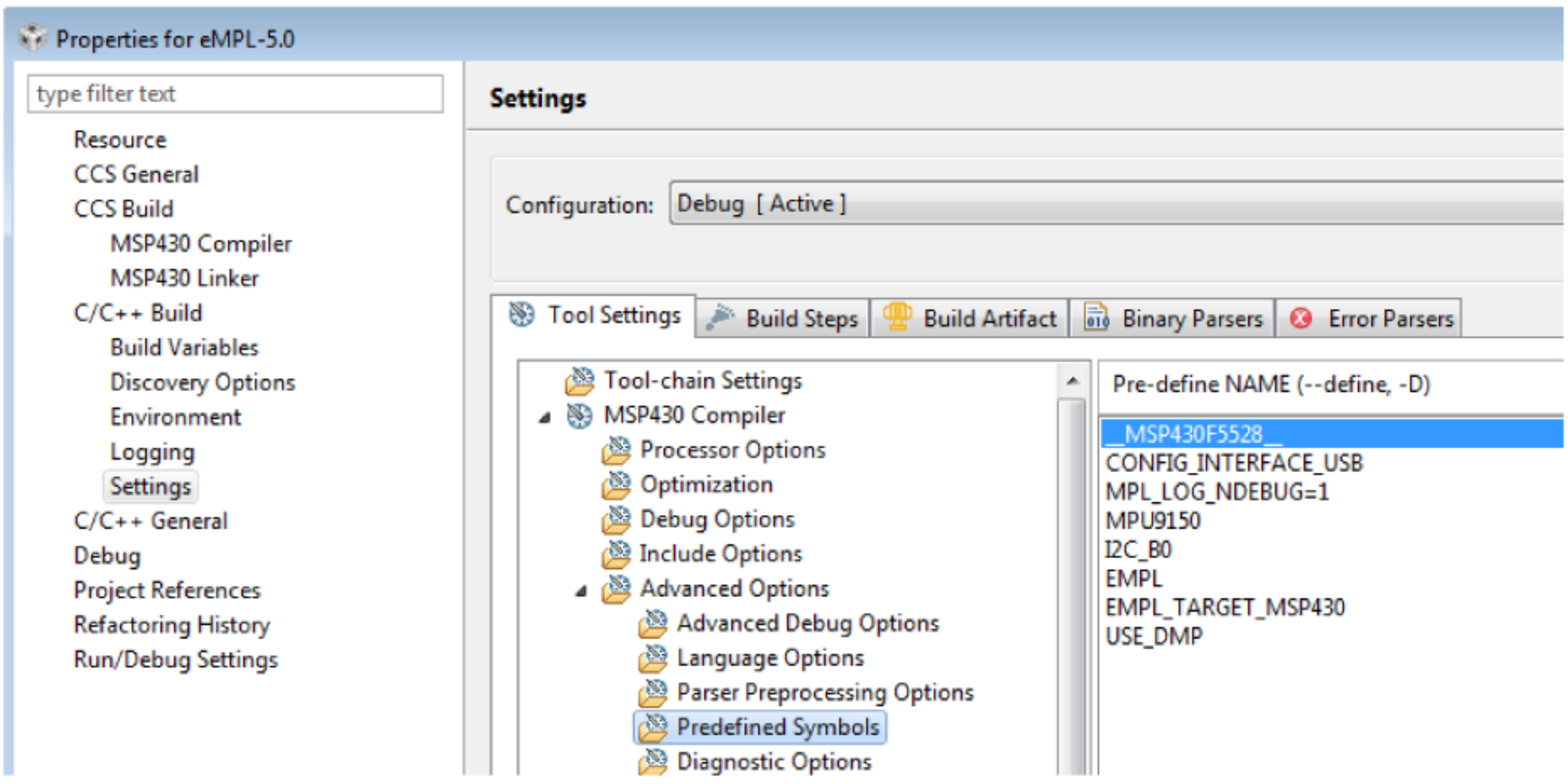
据超过某个阈值，则会产生中断。如果没有运动，并且客户想要在设备中睡觉，直到检测到运动，这是特别有用的。

相关的 API 是 -

- `int mpu_lp_accel_mode(unsigned short rate)`
- `int mpu_lp_motion_interrupt(unsigned short thresh, unsigned char time, unsigned short lpa_freq)`

17 编译器具体设置

为了编译不同的部分（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 or MPU6050 or MPU6500 or MPU9250
- EMPL
- USE_DMP
- EMPL_TARGET_MSP430 or its equivalent

一旦设置了该部分，编译器将编译该特定部分及其功能。