Motion Driver 6.12 –用户指南

目录

1 修订历史记录

修订日期	修订	说明
06/27/2014	1.0	初始版本
07/17/2014	1.1	ARM MPL库的扩展信息
05/15/2015	1.2	针对MD 6.12版本更新

2 目的

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. 发布，它包括适用于ARM MCU和TI-MSP430. 9. 解决方案。仅6. 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和选择。我们将讨论一些更详细的主题，例如DMP编程、校准和自检。

3 在您开始之前

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功能

这是对MD6.12功能的快速概述。

- **DMP功能:**
  - 3/6轴低功率四元数
  - 点击、方向和计步器手势检测
- **MPL算法:**
  - 运行时陀螺仪校准
  - 运行时间陀螺温度补偿
  - 运行时罗盘校准
  - 运行时抗磁干扰
  - 3/6/9轴传感器融合
- **硬件功能:**
  - 工厂校准
  - 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
  - 注册转储

5 选择MCU

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、您要使用的功能以及系统中的其他组件。不过，一般来说，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。

    - 16位MCU – 128K和12K
    - 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。

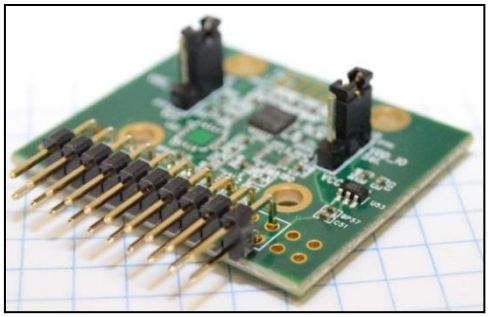
- Long Long数学支持：MPL库需要支持Long Long（64位）数学。您需要确保您使用的是MPL库，您的工具链可以支持这一点。通常8051 MCU无法支持这样的数学计算。如果工具链不支持Long long数学，您仍然可以使用DMP进行6轴融合。

- 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。这会影响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采样率的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，您可以通过InvenSense.com获取InvenSense MPU评估板。MPU6050、MPU6500、MPU9150和



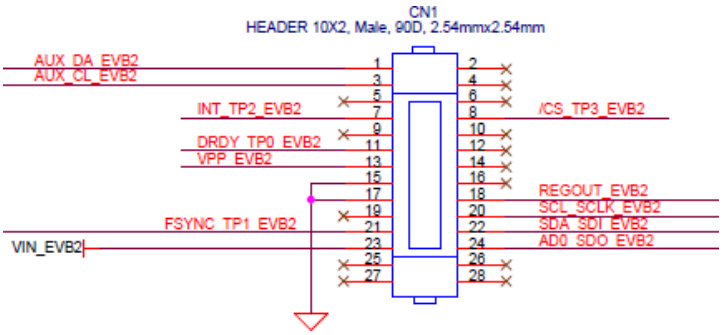
MPU9250均可用。

评价板

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（请参阅设备规范）
- 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评估板的引脚输出相同。



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 MOTION DRIVER 6.12固件包

Motion Driver 6.12版本固件包含以下文件夹:

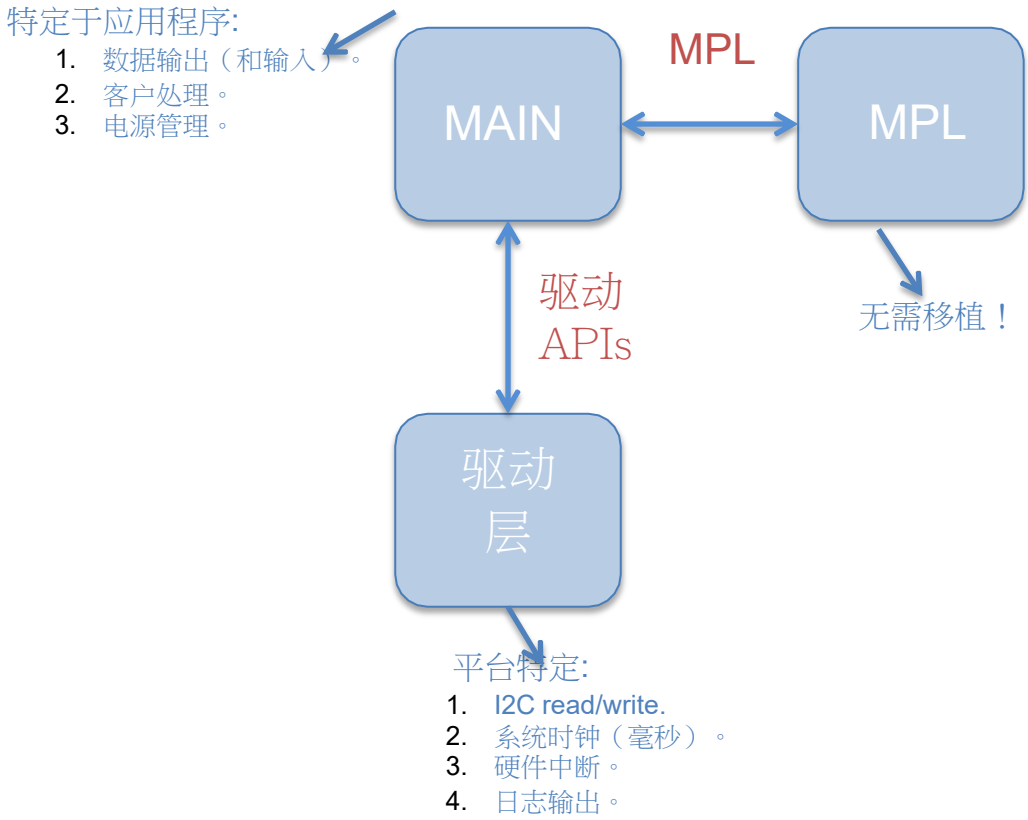
- core\driver：此文件夹包含MPU设备的InvenSense驱动程序层以及MCU特定驱动程序
- simple_apps\msp430\mllite_test.c或src\main.c：项目应用程序的主函数和主循环。客户可以将此代码作为参考，将驱动程序功能集成到他们的项目中。
- core\mllite：此文件夹包含存储接收到的传感器数据并处理数据的MPL数据处理函数。
- core\mpl：包含InvenSense专有MPL库-一个包含用于传感器融合和运行时校准的高级算法的库。
- core\eMPL-hal：此文件夹包含提供相同传感器数据转换的文件，例如欧拉角。

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) 驱动器
2) 运动处理库
3) 示例HAL

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驱动层(core\driver\eMPL)由这些文件组成

- inv_mpu.c 该驱动程序可以轻松移植到不同的嵌入式平台。
- inv_mpu.h 包含InvenSense驱动程序的结构和原型。
- inv_mpu_dmp_motion_driver.c包含dmp映像的驱动程序以及用于加载和配置DMP的API。
- inv_mpu_dmp_motion_driver.h –包含DMP功能的原型和定义
- dmpKey.h 包含DMP功能的DMP内存位置的定义
- dmpmap.h 包含DMP内存位置的定义

用户将需要提供以下API来支持I2C读/写功能、系统时钟访问、硬件中断回调和对应于平台的日志记录MD6.12将被移植。

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           MPL_LOGI
#define log_e           MPL_LOGE
```

i2c_write和i2c_read：这需要链接到i2c驱动程序。该函数将接受4个参数，然后执行i2c事务

- 无符号字符slave_addr
- 无符号字符reg_addr
- 无符号字符长度
- 无符号字符*数据

delay_ms：此函数将接受一个无符号长参数，它将作为系统的延迟（以毫秒为单位）

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整数，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库是专有InvenSense Motion Apps算法的核心，由Mllite和mpl目录组成。MPL不需要移植。您可能需要封装含系统特定的头文件以支持mllite封装中的memcpy、memset等函数调用。MD6.12封装将封装含预编译的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。您可以获取的数据来自以下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（无符号短fsr）`
- `int mpu_set_accel_fsr（无符号字符fsr）`
- `int mpu_set_lpf（无符号短lpf）`
- `int mpu_set_sample_rate（无符号短速率）`
- `int mpu_set_compass_sample_rate（无符号短速率）`
- `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3rd方罗盘（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
 * driver(s).
 */
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}
#endif
}
```

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

- 先进先出输出率
- 生成的DMP

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. 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要处理。

以下是与中断相关的API列表

- **int dmp_set_interrupt_mode（无符号字符模式）**
- **静态int set_int_enable（无符号字符启用）**

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卸载高频运动算法计算，减少中断和主机MIPS以提高整体系统性能。

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的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（无效）
- int dmp_load_motion_driver_firmware（无效）
- **int dmp_set_fifo_rate（无符号短率）**
- **int mpu_set_dmp_state（无符号字符启用）**

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功能

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（无符号字符方向，无符号字符计数）`
- `静态无效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驱动会等待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（短*陀螺，短*加速，长*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。
  ```

与MPU6500. MPU9250. 比，MPU6050. MPU9150. 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

参数类型	参数名称	
输出	结果	该函数返回自检结果，如下表所示。
I/O	加速	返回加速度偏差。
I/O	陀螺	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“1”

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  00121  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、温度和环境而缓慢变化，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长度)`

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。有特定的IAR、Keil和GNU 4.9.3编译库。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) //enable 6-axis`
- `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断模式

LPA模式和运动中断模式类似，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。

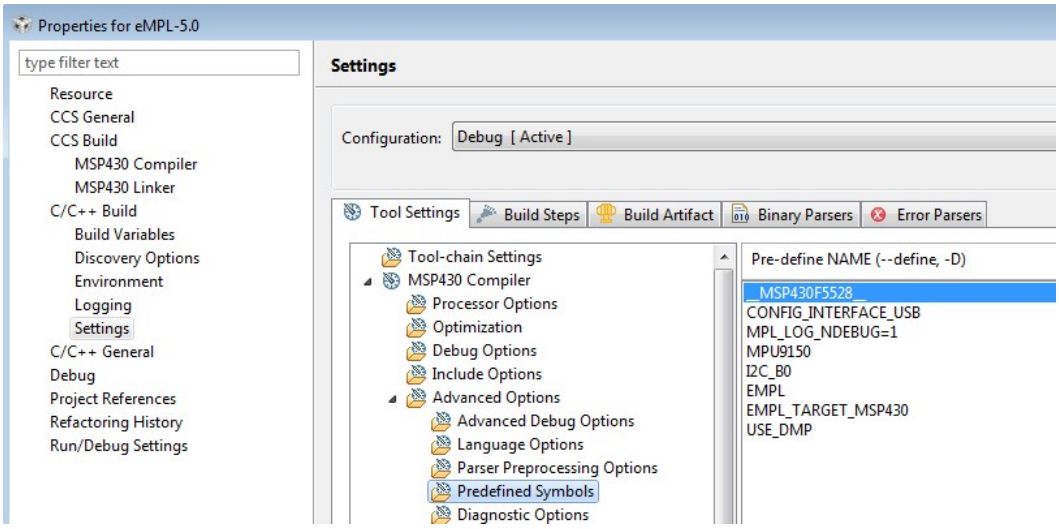
客户还可以将设备置于运动中断模式。在这种模式下，设备将处于LPA模式，如果加速度数据超过某个阈值，它将产生中断。如果没有运动并且客户想要让设备休眠直到检测到运动，这将特别有用。

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或其等价物

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。