**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

Doc : SW-EMD-REL-5.1.1
Doc Rev : 1.0
Date : 12/14/2012

# Embedded Motion Driver v5.1.1 APIs Specification

A printed copy of this document is
**NOT UNDER REVISION CONTROL**
unless it is dated and stamped in red ink as,
"REVISION CONTROLLED COPY."

# Chapter 1

# Purpose and Scope

This document is a guide to all of the functions available in the InvenSense Embedded Motion Driver (eMD), and corresponds with Embedded Motion Driver Release v5.1.1.

The eMD contains the code for configuring the InvenSense devices and using the DMP hardware features. All of the source code is in ANSI C and can be compiled in C or C++ environments.

All functions available in the eMD are described in this document, including all parameters involved in the function calls.

For more information on how to use these functions in a specific application, refer to InvenSense Application Notes.

# Chapter 2

# About this document

This document is automatically generated from the source files using Doxygen's output format in the LaTeX. Heading, footer, and general document format are customized from the standard header template provided by Doxygen. The document is subdivided in the various sections, each describing the main source Modules composing the eMD and implementing specific features.

Every section starts with a brief description and an overview of the functions composing the module. Each of those functions is also fully documented in the analogous "Function Documentation" section. Clicking on the function prototype will lead to the portion of text full documentating it.

This **Embedded Motion Driver Functional Specification** is best viewed in a PDF viewer, as it provides text hyperlinks and bookmarks on the left-hand side for ease of browsing. There is an Alphabetical Index of the modules and their functions available at the bottom of this document.

# Chapter 3

# Module Index

## 3.1 Modules

Here is a list of all modules:

## 2 Module Index

# Chapter 4

# Module Documentation

## 4.1 Sensor Driver Layer

Hardware drivers to communicate with sensors via I2C.

### Files

- file inv_mpu.c
  *An I2C-based driver for Invensense gyroscopes.*
- file inv_mpu_dmp_android.c
  *DMP image and interface functions.*

### Functions

- int dmp_enable_6x_lp_quat (unsigned char enable)
  *Generate 6-axis quaternions from the DMP.*
- int dmp_enable_feature (unsigned short mask)
  *Enable DMP features.*
- int dmp_enable_gyro_cal (unsigned char enable)
  *Calibrate the gyro data in the DMP.*
- int dmp_enable_lp_quat (unsigned char enable)
  *Generate 3-axis quaternions from the DMP.*

- int dmp_enable_no_motion_detection (unsigned char enable)

  *Detect accel-based no motion events.*

- int dmp_get_fifo_rate (unsigned short *rate)

  *Get DMP output rate.*

- int dmp_get_pedometer_step_count (unsigned long *count)

  *Get current step count.*

- int dmp_get_pedometer_walk_time (unsigned long *time)

  *Get duration of walking time.*

- int dmp_load_android_firmware (void)

  *Load the DMP with this image.*

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

  *Get one packet from the FIFO.*

- int dmp_register_android_orient_cb (void(*func)(unsigned char))

  *Register a function to be executed on a android orientation event.*

- int dmp_register_no_motion_cb (void(*func)(void))

  *Register a function to be executed on a no motion event.*

- int dmp_register_tap_cb (void(*func)(unsigned char, unsigned char))

  *Register a function to be executed on a tap event.*

- int dmp_set_accel_bias (long *bias)

  *Push accel biases to the DMP.*

- int dmp_set_fifo_rate (unsigned short rate)

  *Set DMP output rate.*

- int dmp_set_gyro_bias (long *bias)

  *Push gyro biases to the DMP.*

- int dmp_set_interrupt_mode (unsigned char mode)

  *Specify when a DMP interrupt should occur.*

- int dmp_set_no_motion_thresh (unsigned long thresh_mg)

*Set no motion threshold.*

- int dmp_set_no_motion_time (unsigned short time_ms)
  *Set no motion delay.*
- int dmp_set_orientation (unsigned short orient)
  *Push gyro and accel orientation to the DMP.*
- int dmp_set_pedometer_step_count (unsigned long count)
  *Overwrite current step count.*
- int dmp_set_pedometer_walk_time (unsigned long time)
  *Overwrite current walk time.*
- int dmp_set_shake_reject_thresh (long sf, unsigned short thresh)
  *Set shake rejection threshold.*
- int dmp_set_shake_reject_time (unsigned short time)
  *Set shake rejection time.*
- int dmp_set_shake_reject_timeout (unsigned short time)
  *Set shake rejection timeout.*
- int dmp_set_tap_axes (unsigned char axis)
  *Set which axes will register a tap.*
- int dmp_set_tap_count (unsigned char min_taps)
  *Set minimum number of taps needed for an interrupt.*
- int dmp_set_tap_thresh (unsigned char axis, unsigned short thresh)
  *Set tap threshold for a specific axis.*
- int dmp_set_tap_time (unsigned short time)
  *Set length between valid taps.*
- int dmp_set_tap_time_multi (unsigned short time)
  *Set max time between taps to register as a multi-tap.*
- int mpu_configure_fifo (unsigned char sensors)
  *Select which sensors are pushed to FIFO.*
- int mpu_get_accel_fsr (unsigned char *fsr)

*Get the accel full-scale range.*

- int mpu_get_accel_reg (short *data, unsigned long *timestamp)
  *Read raw accel data directly from the registers.*
- int mpu_get_accel_sens (unsigned short *sens)
  *Get accel sensitivity scale factor.*
- int mpu_get_compass_fsr (unsigned short *fsr)
  *Get the compass full-scale range.*
- int mpu_get_compass_reg (short *data, unsigned long *timestamp)
  *Read raw compass data.*
- int mpu_get_compass_sample_rate (unsigned short *rate)
  *Get compass sampling rate.*
- int mpu_get_dmp_state (unsigned char *enabled)
  *Get DMP state.*
- int mpu_get_fifo_config (unsigned char *sensors)
  *Get current FIFO configuration.*
- int mpu_get_gyro_fsr (unsigned short *fsr)
  *Get the gyro full-scale range.*
- int mpu_get_gyro_reg (short *data, unsigned long *timestamp)
  *Read raw gyro data directly from the registers.*
- int mpu_get_gyro_sens (float *sens)
  *Get gyro sensitivity scale factor.*
- int mpu_get_int_status (short *status)
  *Read the MPU interrupt status registers.*
- int mpu_get_lpf (unsigned short *lpf)
  *Get the current DLPF setting.*
- int mpu_get_power_state (unsigned char *power_on)
  *Get current power state.*
- int mpu_get_sample_rate (unsigned short *rate)

*Get sampling rate.*

- int mpu_get_temperature (long *data, unsigned long *timestamp)

  *Read temperature data directly from the registers.*

- int mpu_init (struct int_param_s *int_param)

  *Initialize hardware.*

- int mpu_load_firmware (unsigned short length, const unsigned char *firmware, unsigned short start_addr, unsigned short sample_rate)

  *Load and verify DMP image.*

- int mpu_lp_accel_mode (unsigned char rate)

  *Enter low-power accel-only mode.*

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

  *Enters LP accel motion interrupt mode.*

- int mpu_read_fifo (short *gyro, short *accel, unsigned long *timestamp, unsigned char *sensors, unsigned char *more)

  *Get one packet from the FIFO.*

- int mpu_read_fifo_stream (unsigned short length, unsigned char *data, unsigned char *more)

  *Get one unparsed packet from the FIFO.*

- int mpu_read_mem (unsigned short mem_addr, unsigned short length, unsigned char *data)

  *Read from the DMP memory.*

- int mpu_read_reg (unsigned char reg, unsigned char *data)

  *Read from a single register.*

- int mpu_reg_dump (void)

  *Register dump for testing.*

- int mpu_reset_fifo (void)

  *Reset FIFO read/write pointers.*

- int mpu_run_self_test (long *gyro, long *accel)

  *Trigger gyro/accel/compass self-test.*

- int mpu_set_accel_bias (const long *accel_bias)

  *Push biases to the accel bias registers.*

- int mpu_set_accel_fsr (unsigned char fsr)

  *Set the accel full-scale range.*

- int mpu_set_bypass (unsigned char bypass_on)

  *Set device to bypass mode.*

- int mpu_set_compass_sample_rate (unsigned short rate)

  *Set compass sampling rate.*

- int mpu_set_dmp_state (unsigned char enable)

  *Enable/disable DMP support.*

- int mpu_set_gyro_fsr (unsigned short fsr)

  *Set the gyro full-scale range.*

- int mpu_set_int_latched (unsigned char enable)

  *Enable latched interrupts.*

- int mpu_set_int_level (unsigned char active_low)

  *Set interrupt level.*

- int mpu_set_lpf (unsigned short lpf)

  *Set digital low pass filter.*

- int mpu_set_sample_rate (unsigned short rate)

  *Set sampling rate.*

- int mpu_set_sensors (unsigned char sensors)

  *Turn specific sensors on/off.*

- int mpu_write_mem (unsigned short mem_addr, unsigned short length, unsigned char *data)

  *Write to the DMP memory.*

### 4.1.1 Detailed Description

Hardware drivers to communicate with sensors via I2C.

### 4.1.2 Function Documentation

#### 4.1.2.1 int dmp_enable_6x_lp_quat (unsigned char *enable*)

Generate 6-axis quaternions from the DMP.

In this driver, the 3-axis and 6-axis DMP quaternion features are mutually exclusive.

**Parameters:**

***enable*** 1 to enable 6-axis quaternion.

**Returns:**

0 if successful.

#### 4.1.2.2 int dmp_enable_feature (unsigned short *mask*)

Enable DMP features.

The following #define's are used in the input mask:

DMP_FEATURE_TAP

DMP_FEATURE_ANDROID_ORIENT

DMP_FEATURE_LP_QUAT

DMP_FEATURE_6X_LP_QUAT

DMP_FEATURE_GYRO_CAL

DMP_FEATURE_SEND_RAW_ACCEL

DMP_FEATURE_SEND_RAW_GYRO

NOTE: DMP_FEATURE_LP_QUAT and DMP_FEATURE_6X_LP_QUAT are mutually exclusive.

NOTE: DMP_FEATURE_SEND_RAW_GYRO and DMP_FEATURE_SEND_CAL_GYRO are also mutually exclusive.

**Parameters:**

***mask*** Mask of features to enable.

**Returns:**

0 if successful.

#### 4.1.2.3 int dmp_enable_gyro_cal (unsigned char *enable*)

Calibrate the gyro data in the DMP.

After eight seconds of no motion, the DMP will compute gyro biases and subtract them from the quaternion output. If *dmp_enable_feature* is called with *DMP_FEATURE_SEND_CAL_GYRO*, the biases will also be subtracted from the gyro output.

**Parameters:**

***enable*** 1 to enable gyro calibration.

**Returns:**

0 if successful.

#### 4.1.2.4 int dmp_enable_lp_quat (unsigned char *enable*)

Generate 3-axis quaternions from the DMP.

In this driver, the 3-axis and 6-axis DMP quaternion features are mutually exclusive.

**Parameters:**

***enable*** 1 to enable 3-axis quaternion.

**Returns:**

0 if successful.

#### 4.1.2.5 int dmp_enable_no_motion_detection (unsigned char *enable*)

Detect accel-based no motion events.

**Parameters:**

***enable*** 1 to enable accel-based no motion detection.

**Returns:**

0 if successful.

#### 4.1.2.6 int dmp_get_fifo_rate (unsigned short * *rate*)

Get DMP output rate.

**Parameters:**

***rate*** Current fifo rate (Hz).

**Returns:**

0 if successful.

#### 4.1.2.7 int dmp_get_pedometer_step_count (unsigned long * *count*)

Get current step count.

**Parameters:**

***count*** Number of steps detected.

**Returns:**

0 if successful.

#### 4.1.2.8 int dmp_get_pedometer_walk_time (unsigned long * *time*)

Get duration of walking time.

**Parameters:**

***time*** Walk time in milliseconds.

**Returns:**

0 if successful.

#### 4.1.2.9 int dmp_load_android_firmware (void)

Load the DMP with this image.

**Returns:**

0 if successful.

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

Get one packet from the FIFO.

If *sensors* does not contain a particular sensor, disregard the data returned to that pointer.

*sensors* can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

INV_WXYZ_QUAT

If the FIFO has no new data, *sensors* will be zero.

If the FIFO is disabled, *sensors* will be zero and this function will return a non-zero error code.

**Parameters:**

***gyro*** Gyro data in hardware units.

***accel*** Accel data in hardware units.

***quat*** 3-axis quaternion data in hardware units.

***timestamp*** Timestamp in milliseconds.

***sensors*** Mask of sensors read from FIFO.

***more*** Number of remaining packets.

**Returns:**

0 if successful.

#### 4.1.2.11 int dmp_register_android_orient_cb (void(*)(unsigned char) *func*)

Register a function to be executed on a android orientation event.

**Parameters:**

***func*** Callback function.

**Returns:**

0 if successful.

#### 4.1.2.12 int dmp_register_no_motion_cb (void(∗)(void) *func*)

Register a function to be executed on a no motion event.

**Parameters:**

***func*** Callback function.

**Returns:**

0 if successful.

#### 4.1.2.13 int dmp_register_tap_cb (void(∗)(unsigned char, unsigned char) *func*)

Register a function to be executed on a tap event.

The tap direction is represented by one of the following:

TAP_X_UP

TAP_X_DOWN

TAP_Y_UP

TAP_Y_DOWN

TAP_Z_UP

TAP_Z_DOWN

**Parameters:**

***func*** Callback function.

**Returns:**

0 if successful.

#### 4.1.2.14 int dmp_set_accel_bias (long ∗ *bias*)

Push accel biases to the DMP.

These biases will be removed from the DMP 6-axis quaternion.

**Parameters:**

***bias*** Accel biases in q16.

**Returns:**

0 if successful.

#### 4.1.2.15 int dmp_set_fifo_rate (unsigned short *rate*)

Set DMP output rate.

Only used when DMP is on.

**Parameters:**

***rate*** Desired fifo rate (Hz).

**Returns:**

0 if successful.

#### 4.1.2.16 int dmp_set_gyro_bias (long * *bias*)

Push gyro biases to the DMP.

Because the gyro integration is handled in the DMP, any gyro biases calculated by the MPL should be pushed down to DMP memory to remove 3-axis quaternion drift.

NOTE: If the DMP-based gyro calibration is enabled, the DMP will overwrite the biases written to this location once a new one is computed.

**Parameters:**

***bias*** Gyro biases in q16.

**Returns:**

0 if successful.

#### 4.1.2.17 int dmp_set_interrupt_mode (unsigned char *mode*)

Specify when a DMP interrupt should occur.

A DMP interrupt can be configured to trigger on either of the two conditions below:

a. One FIFO period has elapsed (set by *mpu_set_sample_rate*).

b. A tap event has been detected.

**Parameters:**

***mode*** DMP_INT_GESTURE or DMP_INT_CONTINUOUS.

**Returns:**

0 if successful.

#### 4.1.2.18 int dmp_set_no_motion_thresh (unsigned long *thresh_mg*)

Set no motion threshold.

The DMP detects no motion when linear acceleration in each accel axis is below this threshold.

**Parameters:**

***thresh_mg*** Threshold in milli-gs, q16.

**Returns:**

0 if successful.

#### 4.1.2.19 int dmp_set_no_motion_time (unsigned short *time_ms*)

Set no motion delay.

This function sets how long the device must be in no motion before a no motion event is reported.

**Parameters:**

***time_ms*** Delay in milliseconds.

**Returns:**

0 if successful.

#### 4.1.2.20 int dmp_set_orientation (unsigned short *orient*)

Push gyro and accel orientation to the DMP.

The orientation is represented here as the output of *inv_orientation_matrix_to_scalar*.

**Parameters:**

***orient*** Gyro and accel orientation in body frame.

**Returns:**

0 if successful.

#### 4.1.2.21 int dmp_set_pedometer_step_count (unsigned long *count*)

Overwrite current step count.

WARNING: This function writes to DMP memory and could potentially encounter a race condition if called while the pedometer is enabled.

**Parameters:**

***count*** New step count.

**Returns:**

0 if successful.

#### 4.1.2.22 int dmp_set_pedometer_walk_time (unsigned long *time*)

Overwrite current walk time.

WARNING: This function writes to DMP memory and could potentially encounter a race condition if called while the pedometer is enabled.

**Parameters:**

***time*** New walk time in milliseconds.

#### 4.1.2.23 int dmp_set_shake_reject_thresh (long *sf*, unsigned short *thresh*)

Set shake rejection threshold.

If the DMP detects a gyro sample larger than *thresh*, taps are rejected.

**Parameters:**

***sf*** Gyro scale factor.

***thresh*** Gyro threshold in dps.

**Returns:**

0 if successful.

#### 4.1.2.24 int dmp_set_shake_reject_time (unsigned short *time*)

Set shake rejection time.

Sets the length of time that the gyro must be outside of the threshold set by *gyro_set_shake_reject_thresh* before taps are rejected. A mandatory 60 ms is added to this parameter.

**Parameters:**

***time*** Time in milliseconds.

**Returns:**

0 if successful.

#### 4.1.2.25 int dmp_set_shake_reject_timeout (unsigned short *time*)

Set shake rejection timeout.

Sets the length of time after a shake rejection that the gyro must stay inside of the threshold before taps can be detected again. A mandatory 60 ms is added to this parameter.

**Parameters:**

***time*** Time in milliseconds.

**Returns:**

0 if successful.

#### 4.1.2.26 int dmp_set_tap_axes (unsigned char *axis*)

Set which axes will register a tap.

**Parameters:**

***axis*** 1, 2, and 4 for XYZ, respectively.

**Returns:**

0 if successful.

#### 4.1.2.27 int dmp_set_tap_count (unsigned char *min_taps*)

Set minimum number of taps needed for an interrupt.

**Parameters:**

***min_taps*** Minimum consecutive taps (1-4).

**Returns:**

0 if successful.

#### 4.1.2.28 int dmp_set_tap_thresh (unsigned char *axis*, unsigned short *thresh*)

Set tap threshold for a specific axis.

**Parameters:**

***axis*** 1, 2, and 4 for XYZ accel, respectively.

***thresh*** Tap threshold, in mg/ms.

**Returns:**

0 if successful.

#### 4.1.2.29 int dmp_set_tap_time (unsigned short *time*)

Set length between valid taps.

**Parameters:**

***time*** Milliseconds between taps.

**Returns:**

0 if successful.

#### 4.1.2.30 int dmp_set_tap_time_multi (unsigned short *time*)

Set max time between taps to register as a multi-tap.

**Parameters:**

***time*** Max milliseconds between taps.

**Returns:**

0 if successful.

#### 4.1.2.31 int mpu_configure_fifo (unsigned char *sensors*)

Select which sensors are pushed to FIFO.

*sensors* can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

**Parameters:**

***sensors*** Mask of sensors to push to FIFO.

**Returns:**

0 if successful.

#### 4.1.2.32 int mpu_get_accel_fsr (unsigned char * *fsr*)

Get the accel full-scale range.

**Parameters:**

***fsr*** Current full-scale range.

**Returns:**

0 if successful.

#### 4.1.2.33 int mpu_get_accel_reg (short * *data*, unsigned long * *timestamp*)

Read raw accel data directly from the registers.

**Parameters:**

***data*** Raw data in hardware units.

***timestamp*** Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

#### 4.1.2.34 int mpu_get_accel_sens (unsigned short * *sens*)

Get accel sensitivity scale factor.

**Parameters:**

***sens*** Conversion from hardware units to g's.

**Returns:**

0 if successful.

#### 4.1.2.35 int mpu_get_compass_fsr (unsigned short * *fsr*)

Get the compass full-scale range.

**Parameters:**

***fsr*** Current full-scale range.

**Returns:**

0 if successful.

#### 4.1.2.36 int mpu_get_compass_reg (short * *data*, unsigned long * *timestamp*)

Read raw compass data.

**Parameters:**

***data*** Raw data in hardware units.
***timestamp*** Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

#### 4.1.2.37 int mpu_get_compass_sample_rate (unsigned short * *rate*)

Get compass sampling rate.

**Parameters:**

***rate*** Current compass sampling rate (Hz).

**Returns:**

0 if successful.

#### 4.1.2.38 int mpu_get_dmp_state (unsigned char * *enabled*)

Get DMP state.

**Parameters:**

***enabled*** 1 if enabled.

**Returns:**

0 if successful.

#### 4.1.2.39 int mpu_get_fifo_config (unsigned char * *sensors*)

Get current FIFO configuration.

*sensors* can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

**Parameters:**

***sensors*** Mask of sensors in FIFO.

**Returns:**

0 if successful.

#### 4.1.2.40 int mpu_get_gyro_fsr (unsigned short * *fsr*)

Get the gyro full-scale range.

**Parameters:**

***fsr*** Current full-scale range.

**Returns:**

0 if successful.

#### 4.1.2.41 int mpu_get_gyro_reg (short * *data*, unsigned long * *timestamp*)

Read raw gyro data directly from the registers.

**Parameters:**

***data*** Raw data in hardware units.

***timestamp*** Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

#### 4.1.2.42 int mpu_get_gyro_sens (float * *sens*)

Get gyro sensitivity scale factor.

**Parameters:**

***sens*** Conversion from hardware units to dps.

**Returns:**

0 if successful.

#### 4.1.2.43 int mpu_get_int_status (short * *status*)

Read the MPU interrupt status registers.

**Parameters:**

***status*** Mask of interrupt bits.

**Returns:**

0 if successful.

#### 4.1.2.44 int mpu_get_lpf (unsigned short * *lpf*)

Get the current DLPF setting.

**Parameters:**

***lpf*** Current LPF setting. 0 if successful.

#### 4.1.2.45 int mpu_get_power_state (unsigned char * *power_on*)

Get current power state.

**Parameters:**

***power_on*** 1 if turned on, 0 if suspended.

**Returns:**

0 if successful.

#### 4.1.2.46 int mpu_get_sample_rate (unsigned short * *rate*)

Get sampling rate.

**Parameters:**

***rate*** Current sampling rate (Hz).

**Returns:**

0 if successful.

#### 4.1.2.47 int mpu_get_temperature (long * *data*, unsigned long * *timestamp*)

Read temperature data directly from the registers.

**Parameters:**

***data*** Data in q16 format.
***timestamp*** Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

#### 4.1.2.48 int mpu_init (struct int_param_s * *int_param*)

Initialize hardware.

Initial configuration:

Gyro FSR: +/- 2000DPS

Accel FSR +/- 2G

DLPF: 42Hz

FIFO rate: 50Hz

Clock source: Gyro PLL

FIFO: Disabled.

Data ready interrupt: Disabled, active low, unlatched.

**Parameters:**

***int_param*** Platform-specific parameters to interrupt API.

**Returns:**

0 if successful.

#### 4.1.2.49 int mpu_load_firmware (unsigned short *length*, const unsigned char * *firmware*, unsigned short *start_addr*, unsigned short *sample_rate*)

Load and verify DMP image.

**Parameters:**

***length*** Length of DMP image.

***firmware*** DMP code.

***start_addr*** Starting address of DMP code memory.

***sample_rate*** Fixed sampling rate used when DMP is enabled.

**Returns:**

0 if successful.

#### 4.1.2.50 int mpu_lp_accel_mode (unsigned char *rate*)

Enter low-power accel-only mode.

In low-power accel mode, the chip goes to sleep and only wakes up to sample the accelerometer at one of the following frequencies:

MPU6050: 1.25Hz, 5Hz, 20Hz, 40Hz

MPU6500: 1.25Hz, 2.5Hz, 5Hz, 10Hz, 20Hz, 40Hz, 80Hz, 160Hz, 320Hz, 640Hz

If the requested rate is not one listed above, the device will be set to the next highest rate. Requesting a rate above the maximum supported frequency will result in an error.

To select a fractional wake-up frequency, round down the value passed to *rate*.

**Parameters:**

***rate*** Minimum sampling rate, or zero to disable LP accel mode.

**Returns:**

0 if successful.

#### 4.1.2.51 int mpu_lp_motion_interrupt (unsigned short *thresh*, unsigned char *time*, unsigned char *lpa_freq*)

Enters LP accel motion interrupt mode.

The behavior of this feature is very different between the MPU6050 and the MPU6500. Each chip's version of this feature is explained below.

MPU6050:

When this mode is first enabled, the hardware captures a single accel sample, and subsequent samples are compared with this one to determine if the device is in motion. Therefore, whenever this "locked" sample needs to be changed, this function must be called again.

The hardware motion threshold can be between 32mg and 8160mg in 32mg increments.

Low-power accel mode supports the following frequencies:

1.25Hz, 5Hz, 20Hz, 40Hz

MPU6500:

Unlike the MPU6050 version, the hardware does not "lock in" a reference sample. The hardware monitors the accel data and detects any large change over a short period of time.

The hardware motion threshold can be between 4mg and 1020mg in 4mg increments.

MPU6500 Low-power accel mode supports the following frequencies:

1.25Hz, 2.5Hz, 5Hz, 10Hz, 20Hz, 40Hz, 80Hz, 160Hz, 320Hz, 640Hz

NOTES:

The driver will round down *thresh* to the nearest supported value if an unsupported threshold is selected.

To select a fractional wake-up frequency, round down the value passed to *lpa_freq*.

The MPU6500 does not support a delay parameter. If this function is used for the MPU6500, the value passed to *time* will be ignored.

To disable this mode, set *lpa_freq* to zero. The driver will restore the previous configuration.

**Parameters:**

***thresh*** Motion threshold in mg.

***time*** Duration in milliseconds that the accel data must exceed *thresh* before motion is reported.

***lpa_freq*** Minimum sampling rate, or zero to disable.

**Returns:**

0 if successful.

#### 4.1.2.52 int mpu_read_fifo (short * *gyro*, short * *accel*, unsigned long * *timestamp*, unsigned char * *sensors*, unsigned char * *more*)

Get one packet from the FIFO.

If *sensors* does not contain a particular sensor, disregard the data returned to that pointer.

*sensors* can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

If the FIFO has no new data, *sensors* will be zero.

If the FIFO is disabled, *sensors* will be zero and this function will return a non-zero error code.

**Parameters:**

***gyro*** Gyro data in hardware units.

***accel*** Accel data in hardware units.

***timestamp*** Timestamp in milliseconds.

***sensors*** Mask of sensors read from FIFO.

***more*** Number of remaining packets.

**Returns:**

0 if successful.

#### 4.1.2.53 int mpu_read_fifo_stream (unsigned short *length*, unsigned char * *data*, unsigned char * *more*)

Get one unparsed packet from the FIFO.

This function should be used if the packet is to be parsed elsewhere.

**Parameters:**

***length*** Length of one FIFO packet.

***data*** FIFO packet.

***more*** Number of remaining packets.

#### 4.1.2.54 int mpu_read_mem (unsigned short *mem_addr*, unsigned short *length*, unsigned char * *data*)

Read from the DMP memory.

This function prevents I2C reads past the bank boundaries. The DMP memory is only accessible when the chip is awake.

**Parameters:**

***mem_addr*** Memory location (bank << 8 | start address)

***length*** Number of bytes to read.

***data*** Bytes read from memory.

**Returns:**

0 if successful.

#### 4.1.2.55 int mpu_read_reg (unsigned char *reg*, unsigned char * *data*)

Read from a single register.

NOTE: The memory and FIFO read/write registers cannot be accessed.

**Parameters:**

***reg*** Register address.

***data*** Register data.

**Returns:**

0 if successful.

#### 4.1.2.56 int mpu_reg_dump (void)

Register dump for testing.

**Returns:**

0 if successful.

#### 4.1.2.57 int mpu_reset_fifo (void)

Reset FIFO read/write pointers.

**Returns:**

0 if successful.

#### 4.1.2.58 int mpu_run_self_test (long * *gyro*, long * *accel*)

Trigger gyro/accel/compass self-test.

On success/error, the self-test returns a mask representing the sensor(s) that failed. For each bit, a one (1) represents a "pass" case; conversely, a zero (0) indicates a failure.

The mask is defined as follows:

Bit 0: Gyro.

Bit 1: Accel.

Bit 2: Compass.

Currently, the hardware self-test is unsupported for MPU6500. However, this function can still be used to obtain the accel and gyro biases.

This function must be called with the device either face-up or face-down (z-axis is parallel to gravity).

**Parameters:**

***gyro*** Gyro biases in q16 format.

***accel*** Accel biases (if applicable) in q16 format.

**Returns:**

Result mask (see above).

#### 4.1.2.59 int mpu_set_accel_bias (const long * *accel_bias*)

Push biases to the accel bias registers.

This function expects biases relative to the current sensor output, and these biases will be added to the factory-supplied values.

**Parameters:**

***accel_bias*** New biases.

**Returns:**

0 if successful.

#### 4.1.2.60 int mpu_set_accel_fsr (unsigned char *fsr*)

Set the accel full-scale range.

**Parameters:**

***fsr*** Desired full-scale range.

**Returns:**

0 if successful.

#### 4.1.2.61 int mpu_set_bypass (unsigned char *bypass_on*)

Set device to bypass mode.

**Parameters:**

***bypass_on*** 1 to enable bypass mode.

**Returns:**

0 if successful.

#### 4.1.2.62 int mpu_set_compass_sample_rate (unsigned short *rate*)

Set compass sampling rate.

The compass on the auxiliary I2C bus is read by the MPU hardware at a maximum of 100Hz. The actual rate can be set to a fraction of the gyro sampling rate.

WARNING: The new rate may be different than what was requested. Call mpu_get_-compass_sample_rate to check the actual setting.

**Parameters:**

***rate*** Desired compass sampling rate (Hz).

**Returns:**

0 if successful.

#### 4.1.2.63 int mpu_set_dmp_state (unsigned char *enable*)

Enable/disable DMP support.

**Parameters:**

***enable*** 1 to turn on the DMP.

**Returns:**

0 if successful.

#### 4.1.2.64 int mpu_set_gyro_fsr (unsigned short *fsr*)

Set the gyro full-scale range.

**Parameters:**

***fsr*** Desired full-scale range.

**Returns:**

0 if successful.

#### 4.1.2.65 int mpu_set_int_latched (unsigned char *enable*)

Enable latched interrupts.

Any MPU register will clear the interrupt.

**Parameters:**

***enable*** 1 to enable, 0 to disable.

**Returns:**

0 if successful.

#### 4.1.2.66 int mpu_set_int_level (unsigned char *active_low*)

Set interrupt level.

**Parameters:**

***active_low*** 1 for active low, 0 for active high.

**Returns:**

0 if successful.

#### 4.1.2.67 int mpu_set_lpf (unsigned short *lpf*)

Set digital low pass filter.

The following LPF settings are supported: 188, 98, 42, 20, 10, 5.

**Parameters:**

***lpf*** Desired LPF setting.

**Returns:**

0 if successful.

#### 4.1.2.68 int mpu_set_sample_rate (unsigned short *rate*)

Set sampling rate.

Sampling rate must be between 4Hz and 1kHz.

**Parameters:**

***rate*** Desired sampling rate (Hz).

**Returns:**

0 if successful.

#### 4.1.2.69 int mpu_set_sensors (unsigned char *sensors*)

Turn specific sensors on/off.

*sensors* can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

INV_XYZ_COMPASS

**Parameters:**

***sensors*** Mask of sensors to wake.

**Returns:**

0 if successful.

#### 4.1.2.70 int mpu_write_mem (unsigned short *mem_addr*, unsigned short *length*, unsigned char * *data*)

Write to the DMP memory.

This function prevents I2C writes past the bank boundaries. The DMP memory is only accessible when the chip is awake.

**Parameters:**

***mem_addr*** Memory location (bank << 8 | start address)

***length*** Number of bytes to write.

***data*** Bytes to write to memory.

**Returns:**

0 if successful.