Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:51:42

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 ArduinoGyroscope Class Reference

```
#include <ArduinoGyroscope.h>
```

Private Member Functions

- virtual int getRotationX ()=0
- virtual int getRotationY ()=0
- virtual int getRotationZ ()=0

3.1.1 Detailed Description

Arduino - Serial Gyroscope Driver.

SerialGyroscope.h

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file ArduinoGyroscope.h.

3.1.2 Member Function Documentation

3.1.2.1 virtual int ArduinoGyroscope::getRotationX ( ) [private], [pure virtual]

3.1.2.2 virtual int ArduinoGyroscope::getRotationY ( ) [private], [pure virtual]

3.1.2.3 virtual int ArduinoGyroscope::getRotationZ ( ) [private], [pure virtual]

The documentation for this class was generated from the following file:

- ArduinoGyroscope.h

3.2 MPU6050 Class Reference

```
#include <ArduinoGyroscopeMPU6050.h>
```

Public Member Functions

- MPU6050 ()
- MPU6050 (uint8_t address)
- void initialize ()
- bool testConnection ()
- uint8_t getAuxVDDIOLevel ()
- void setAuxVDDIOLevel (uint8_t level)
- uint8_t getRate ()
- void setRate (uint8_t rate)
- uint8_t getExternalFrameSync ()
- void setExternalFrameSync (uint8_t sync)
- uint8_t getDLPFMode ()
- void setDLPFMode (uint8_t bandwidth)
- uint8_t getFullScaleGyroRange ()
- void setFullScaleGyroRange (uint8_t range)
- bool getAccelXSelfTest ()
- void setAccelXSelfTest (bool enabled)
- bool getAccelYSelfTest ()
- void setAccelYSelfTest (bool enabled)
- bool getAccelZSelfTest ()
- void setAccelZSelfTest (bool enabled)
- uint8_t getFullScaleAccelRange ()
- void setFullScaleAccelRange (uint8_t range)
- uint8_t getDHPFMode ()
- void setDHPFMode (uint8_t mode)
- uint8_t getFreefallDetectionThreshold ()
- void setFreefallDetectionThreshold (uint8_t threshold)
- uint8_t getFreefallDetectionDuration ()
- void setFreefallDetectionDuration (uint8_t duration)
- uint8_t getMotionDetectionThreshold ()
- void setMotionDetectionThreshold (uint8_t threshold)
- uint8_t getMotionDetectionDuration ()
- void setMotionDetectionDuration (uint8_t duration)
- uint8_t getZeroMotionDetectionThreshold ()
- void setZeroMotionDetectionThreshold (uint8_t threshold)
- uint8_t getZeroMotionDetectionDuration ()
- void setZeroMotionDetectionDuration (uint8_t duration)
- bool getTempFIFOEnabled ()
- void setTempFIFOEnabled (bool enabled)
- bool getXGyroFIFOEnabled ()
- void setXGyroFIFOEnabled (bool enabled)
- bool getYGyroFIFOEnabled ()
- void setYGyroFIFOEnabled (bool enabled)
- bool getZGyroFIFOEnabled ()
- void setZGyroFIFOEnabled (bool enabled)
- bool getAccelFIFOEnabled ()
- void setAccelFIFOEnabled (bool enabled)
- bool getSlave2FIFOEnabled ()
- void setSlave2FIFOEnabled (bool enabled)
- bool getSlave1FIFOEnabled ()
- void setSlave1FIFOEnabled (bool enabled)
- bool getSlave0FIFOEnabled ()
- void setSlave0FIFOEnabled (bool enabled)
- bool getMultiMasterEnabled ()

- bool getFSyncInterruptEnabled ()
- void setFSyncInterruptEnabled (bool enabled)
- bool getI2CBypassEnabled ()
- void setI2CBypassEnabled (bool enabled)
- bool getClockOutputEnabled ()
- void setClockOutputEnabled (bool enabled)
- uint8_t getIntEnabled ()
- void setIntEnabled (uint8_t enabled)
- bool getIntFreefallEnabled ()
- void setIntFreefallEnabled (bool enabled)
- bool getIntMotionEnabled ()
- void setIntMotionEnabled (bool enabled)
- bool getIntZeroMotionEnabled ()
- void setIntZeroMotionEnabled (bool enabled)
- bool getIntFIFOBufferOverflowEnabled ()
- void setIntFIFOBufferOverflowEnabled (bool enabled)
- bool getIntI2CMasterEnabled ()
- void setIntI2CMasterEnabled (bool enabled)
- bool getIntDataReadyEnabled ()
- void setIntDataReadyEnabled (bool enabled)
- uint8_t getIntStatus ()
- bool getIntFreefallStatus ()
- bool getIntMotionStatus ()
- bool getIntZeroMotionStatus ()
- bool getIntFIFOBufferOverflowStatus ()
- bool getIntI2CMasterStatus ()
- bool getIntDataReadyStatus ()
- void getMotion9 (int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx, int16_t *gy, int16_t *gz, int16_t *mx, int16_t *my, int16_t *mz)
- void getMotion6 (int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx, int16_t *gy, int16_t *gz)
- void getAcceleration (int16_t *x, int16_t *y, int16_t *z)
- int16_t getAccelerationX ()
- int16_t getAccelerationY ()
- int16_t getAccelerationZ ()
- int16_t getTemperature ()
- void getRotation (int16_t *x, int16_t *y, int16_t *z)
- void getRotationXY (int16_t *x, int16_t *y)
- int16_t getRotationX ()
- int16_t getRotationY ()
- int16_t getRotationZ ()
- uint8_t getExternalSensorByte (int position)
- uint16_t getExternalSensorWord (int position)
- uint32_t getExternalSensorDWord (int position)
- bool getXNegMotionDetected ()
- bool getXPosMotionDetected ()
- bool getYNegMotionDetected ()
- bool getYPosMotionDetected ()
- bool getZNegMotionDetected ()
- bool getZPosMotionDetected ()
- bool getZeroMotionDetected ()
- void setSlaveOutputByte (uint8_t num, uint8_t data)
- bool getExternalShadowDelayEnabled ()
- void setExternalShadowDelayEnabled (bool enabled)
- bool getSlaveDelayEnabled (uint8_t num)
- void setSlaveDelayEnabled (uint8_t num, bool enabled)

- void setXFineGain (int8_t gain)
- int8_t getYFineGain ()
- void setYFineGain (int8_t gain)
- int8_t getZFineGain ()
- void setZFineGain (int8_t gain)
- int16_t getXAccelOffset ()
- void setXAccelOffset (int16_t offset)
- int16_t getYAccelOffset ()
- void setYAccelOffset (int16_t offset)
- int16_t getZAccelOffset ()
- void setZAccelOffset (int16_t offset)
- int16_t getXGyroOffsetUser ()
- void setXGyroOffsetUser (int16_t offset)
- int16_t getYGyroOffsetUser ()
- void setYGyroOffsetUser (int16_t offset)
- int16_t getZGyroOffsetUser ()
- void setZGyroOffsetUser (int16_t offset)
- bool getIntPLLReadyEnabled ()
- void setIntPLLReadyEnabled (bool enabled)
- bool getIntDMPEnabled ()
- void setIntDMPEnabled (bool enabled)
- bool getDMPInt5Status ()
- bool getDMPInt4Status ()
- bool getDMPInt3Status ()
- bool getDMPInt2Status ()
- bool getDMPInt1Status ()
- bool getDMPInt0Status ()
- bool getIntPLLReadyStatus ()
- bool getIntDMPStatus ()
- bool getDMPEnabled ()
- void setDMPEnabled (bool enabled)
- void resetDMP ()
- void setMemoryBank (uint8_t bank, bool prefetchEnabled=false, bool userBank=false)
- void setMemoryStartAddress (uint8_t address)
- uint8_t readMemoryByte ()
- void writeMemoryByte (uint8_t data)
- void readMemoryBlock (uint8_t *data, uint16_t dataSize, uint8_t bank=0, uint8_t address=0)
- bool writeMemoryBlock (const uint8_t *data, uint16_t dataSize, uint8_t bank=0, uint8_t address=0, bool verify=true, bool useProgMem=false)
- bool writeProgMemoryBlock (const uint8_t *data, uint16_t dataSize, uint8_t bank=0, uint8_t address=0, bool verify=true)
- bool writeDMPConfigurationSet (const uint8_t *data, uint16_t dataSize, bool useProgMem=false)
- bool writeProgDMPConfigurationSet (const uint8_t *data, uint16_t dataSize)
- uint8_t getDMPConfig1 ()
- void setDMPConfig1 (uint8_t config)
- uint8_t getDMPConfig2 ()
- void setDMPConfig2 (uint8_t config)

Private Attributes

- uint8_t devAddr
- uint8_t buffer [14]

3.2.1 Detailed Description

Definition at line 402 of file ArduinoGyroscopeMPU6050.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 MPU6050::MPU6050 ( )

Default constructor, uses default I2C address.

See also

MPU6050_DEFAULT_ADDRESS

Definition at line 42 of file ArduinoGyroscopeMPU6050.cpp.

3.2.2.2 MPU6050::MPU6050 (uint8_t *address*)

Specific address constructor.

Parameters

address	I2C address

See also

MPU6050_DEFAULT_ADDRESS
MPU6050_ADDRESS_AD0_LOW
MPU6050_ADDRESS_AD0_HIGH

Definition at line 52 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3 Member Function Documentation

3.2.3.1 void MPU6050::getAcceleration (int16_t * *x*, int16_t * *y*, int16_t * *z*)

Get 3-axis accelerometer readings.

These registers store the most recent accelerometer measurements. Accelerometer measurements are written to these registers at the Sample Rate as defined in Register 25.

The accelerometer measurement registers, along with the temperature measurement registers, gyroscope measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the accelerometer sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit accelerometer measurement has a full scale defined in ACCEL_FS (Register 28). For each full scale setting, the accelerometers' sensitivity per LSB in ACCEL_xOUT is shown in the table below:

```
AFS_SEL | Full Scale Range | LSB Sensitivity
--------+------------------+----------------
0       | +/- 2g           | 8192 LSB/mg
1       | +/- 4g           | 4096 LSB/mg
2       | +/- 8g           | 2048 LSB/mg
3       | +/- 16g          | 1024 LSB/mg
```

Parameters

x	16-bit signed integer container for X-axis acceleration
y	16-bit signed integer container for Y-axis acceleration
z	16-bit signed integer container for Z-axis acceleration

See also

MPU6050_RA_GYRO_XOUT_H

Definition at line 1779 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.2 int16_t MPU6050::getAccelerationX ( )

Get X-axis accelerometer reading.

Returns

X-axis acceleration measurement in 16-bit 2's complement format

See also

getMotion6()
MPU6050_RA_ACCEL_XOUT_H

Definition at line 1790 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.3 int16_t MPU6050::getAccelerationY ( )

Get Y-axis accelerometer reading.

Returns

Y-axis acceleration measurement in 16-bit 2's complement format

See also

getMotion6()
MPU6050_RA_ACCEL_YOUT_H

Definition at line 1799 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.4 int16_t MPU6050::getAccelerationZ ( )

Get Z-axis accelerometer reading.

Returns

Z-axis acceleration measurement in 16-bit 2's complement format

See also

getMotion6()
MPU6050_RA_ACCEL_ZOUT_H

Definition at line 1808 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.5 uint8_t MPU6050::getAccelerometerPowerOnDelay ( )

Get accelerometer power-on delay.

The accelerometer data path provides samples to the sensor registers, Motion detection, Zero Motion detection, and Free Fall detection modules. The signal path contains filters which must be flushed on wake-up with new samples before the detection modules begin operations. The default wake-up delay, of 4ms can be lengthened by up to 3ms. This additional delay is specified in ACCEL_ON_DELAY in units of 1 LSB = 1 ms. The user may select any value above zero unless instructed otherwise by InvenSense. Please refer to Section 8 of the MPU-6000/MPU-6050 Product Specification document for further information regarding the detection modules.

Returns

Current accelerometer power-on delay

See also

MPU6050_RA_MOT_DETECT_CTRL
MPU6050_DETECT_ACCEL_ON_DELAY_BIT

Definition at line 2182 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.6 bool MPU6050::getAccelFIFOEnabled ( )

Get accelerometer FIFO enabled value.

When set to 1, this bit enables ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACC↩EL_ZOUT_H, and ACCEL_ZOUT_L (Registers 59 to 64) to be written into the FIFO buffer.

Returns

Current accelerometer FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 658 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.7 bool MPU6050::getAccelXSelfTest ( )

Get self-test enabled setting for accelerometer X axis.

Returns

Self-test enabled value

See also

MPU6050_RA_ACCEL_CONFIG

Definition at line 262 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.8 bool MPU6050::getAccelYSelfTest ( )

Get self-test enabled value for accelerometer Y axis.

Returns

Self-test enabled value

See also

MPU6050_RA_ACCEL_CONFIG

Definition at line 277 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.9 bool MPU6050::getAccelZSelfTest ( )

Get self-test enabled value for accelerometer Z axis.

Returns

Self-test enabled value

See also

MPU6050_RA_ACCEL_CONFIG

Definition at line 292 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.10 uint8_t MPU6050::getAuxVDDIOLevel ( )

Get the auxiliary I2C supply voltage level.

When set to 1, the auxiliary I2C bus high logic level is VDD. When cleared to 0, the auxiliary I2C bus high logic level is VLOGIC. This does not apply to the MPU-6000, which does not have a VLOGIC pin.

Returns

I2C supply voltage level (0=VLOGIC, 1=VDD)

Definition at line 86 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.11 bool MPU6050::getClockOutputEnabled ( )

Get reference clock output enabled status.

When this bit is equal to 1, a reference clock output is provided at the CLKOUT pin. When this bit is equal to 0, the clock output is disabled. For further information regarding CLKOUT, please refer to the MPU-60X0 Product Specification document.

Returns

Current reference clock output enabled status

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_CLKOUT_EN_BIT

Definition at line 1461 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.12 uint8_t MPU6050::getClockSource ( )

Get clock source setting.

Returns

Current clock source setting

See also

MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_CLKSEL_BIT
MPU6050_PWR1_CLKSEL_LENGTH

Definition at line 2449 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.13 uint8_t MPU6050::getDeviceID ( )

3.2.3.14 uint8_t MPU6050::getDHPFMode ( )

Get the high-pass filter configuration.

The DHPF is a filter module in the path leading to motion detectors (Free Fall, Motion threshold, and Zero Motion). The high pass filter output is not available to the data registers (see Figure in Section 8 of the MPU-6000/ MPU-6050 Product Specification document).

The high pass filter has three modes:

```
   Reset: The filter output settles to zero within one sample. This
          effectively disables the high pass filter. This mode may be toggled
          to quickly settle the filter.

   On:    The high pass filter will pass signals above the cut off frequency.

   Hold:  When triggered, the filter holds the present sample. The filter
          output will be the difference between the input sample and the held
          sample.
```

ACCEL_HPF	Filter Mode	Cut-off Frequency
0	Reset	None
1	On	5Hz
2	On	2.5Hz
3	On	1.25Hz
4	On	0.63Hz
7	Hold	None

Returns

Current high-pass filter configuration

See also

MPU6050_DHPF_RESET
MPU6050_RA_ACCEL_CONFIG

Definition at line 366 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.15 uint8_t MPU6050::getDLPFMode ( )

Get digital low-pass filter configuration.

The DLPF_CFG parameter sets the digital low pass filter configuration. It also determines the internal sampling rate used by the device as shown in the table below.

Note: The accelerometer output rate is 1kHz. This means that for a Sample Rate greater than 1kHz, the same accelerometer sample may be output to the FIFO, DMP, and sensor registers more than once.

	ACCELEROMETER		GYROSCOPE		
DLPF_CFG	Bandwidth	Delay	Bandwidth	Delay	Sample Rate
0	260Hz	0ms	256Hz	0.98ms	8kHz
1	184Hz	2.0ms	188Hz	1.9ms	1kHz
2	94Hz	3.0ms	98Hz	2.8ms	1kHz
3	44Hz	4.9ms	42Hz	4.8ms	1kHz
4	21Hz	8.5ms	20Hz	8.3ms	1kHz
5	10Hz	13.8ms	10Hz	13.4ms	1kHz
6	5Hz	19.0ms	5Hz	18.6ms	1kHz
7	-- Reserved --		-- Reserved --		Reserved

Returns

DLFP configuration

See also

MPU6050_RA_CONFIG
MPU6050_CFG_DLPF_CFG_BIT
MPU6050_CFG_DLPF_CFG_LENGTH

Definition at line 205 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.16 uint8_t MPU6050::getDMPConfig1 ( )

3.2.3.17 uint8_t MPU6050::getDMPConfig2 ( )

3.2.3.18 bool MPU6050::getDMPEnabled ( )

3.2.3.19 bool MPU6050::getDMPInt0Status ( )

3.2.3.20 bool MPU6050::getDMPInt1Status ( )

3.2.3.21 bool MPU6050::getDMPInt2Status ( )

3.2.3.22 bool MPU6050::getDMPInt3Status ( )

3.2.3.23 bool MPU6050::getDMPInt4Status ( )

3.2.3.24 bool MPU6050::getDMPInt5Status ( )

3.2.3.25 uint8_t MPU6050::getExternalFrameSync ( )

Get external FSYNC configuration.

Configures the external Frame Synchronization (FSYNC) pin sampling. An external signal connected to the FSYNC pin can be sampled by configuring EXT_SYNC_SET. Signal changes to the FSYNC pin are latched so that short strobes may be captured. The latched FSYNC signal will be sampled at the Sampling Rate, as defined in register 25. After sampling, the latch will reset to the current FSYNC signal state.

The sampled value will be reported in place of the least significant bit in a sensor data register determined by the value of EXT_SYNC_SET according to the following table.

```
EXT_SYNC_SET | FSYNC Bit Location
-------------+-------------------
0            | Input disabled
1            | TEMP_OUT_L[0]
2            | GYRO_XOUT_L[0]
3            | GYRO_YOUT_L[0]
4            | GYRO_ZOUT_L[0]
5            | ACCEL_XOUT_L[0]
6            | ACCEL_YOUT_L[0]
7            | ACCEL_ZOUT_L[0]
```

Returns

FSYNC configuration value

Definition at line 165 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.26 uint8_t MPU6050::getExternalSensorByte (int *position*)

Read single byte from external sensor data register.

These registers store data read from external sensors by the Slave 0, 1, 2, and 3 on the auxiliary I2C interface. Data read by Slave 4 is stored in I2C_SLV4_DI (Register 53).

External sensor data is written to these registers at the Sample Rate as defined in Register 25. This access rate can be reduced by using the Slave Delay Enable registers (Register 103).

External sensor data registers, along with the gyroscope measurement registers, accelerometer measurement registers, and temperature measurement registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the external sensors' internal register set is always updated at the Sample Rate (or the reduced access rate) whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Data is placed in these external sensor data registers according to I2C_SLV0_CTRL, I2C_SLV1_CTRL, I2C_SL↩ V2_CTRL, and I2C_SLV3_CTRL (Registers 39, 42, 45, and 48). When more than zero bytes are read (I2C_SLVx↩ _LEN $>$ 0) from an enabled slave (I2C_SLVx_EN = 1), the slave is read at the Sample Rate (as defined in Register 25) or delayed rate (if specified in Register 52 and 103). During each Sample cycle, slave reads are performed in order of Slave number. If all slaves are enabled with more than zero bytes to be read, the order will be Slave 0, followed by Slave 1, Slave 2, and Slave 3.

Each enabled slave will have EXT_SENS_DATA registers associated with it by number of bytes read (I2C_SLVx↩ _LEN) in order of slave number, starting from EXT_SENS_DATA_00. Note that this means enabling or disabling a slave may change the higher numbered slaves' associated registers. Furthermore, if fewer total bytes are being read from the external sensors as a result of such a change, then the data remaining in the registers which no longer have an associated slave device (i.e. high numbered registers) will remain in these previously allocated registers unless reset.

If the sum of the read lengths of all SLVx transactions exceed the number of available EXT_SENS_DATA registers, the excess bytes will be dropped. There are 24 EXT_SENS_DATA registers and hence the total read lengths between all the slaves cannot be greater than 24 or some bytes will be lost.

Note: Slave 4's behavior is distinct from that of Slaves 0-3. For further information regarding the characteristics of Slave 4, please refer to Registers 49 to 53.

EXAMPLE: Suppose that Slave 0 is enabled with 4 bytes to be read (I2C_SLV0_EN = 1 and I2C_SLV0_LEN = 4) while Slave 1 is enabled with 2 bytes to be read so that I2C_SLV1_EN = 1 and I2C_SLV1_LEN = 2. In such a situation, EXT_SENS_DATA _00 through _03 will be associated with Slave 0, while EXT_SENS_DATA _04 and 05 will be associated with Slave 1. If Slave 2 is enabled as well, registers starting from EXT_SENS_DATA_06 will be allocated to Slave 2.

If Slave 2 is disabled while Slave 3 is enabled in this same situation, then registers starting from EXT_SENS_DA↩ TA_06 will be allocated to Slave 3 instead.

REGISTER ALLOCATION FOR DYNAMIC DISABLE VS. NORMAL DISABLE: If a slave is disabled at any time, the space initially allocated to the slave in the EXT_SENS_DATA register, will remain associated with that slave. This is to avoid dynamic adjustment of the register allocation.

The allocation of the EXT_SENS_DATA registers is recomputed only when (1) all slaves are disabled, or (2) the I2C_MST_RST bit is set (Register 106).

This above is also true if one of the slaves gets NACKed and stops functioning.

Parameters

position	Starting position (0-23)

Returns

Byte read from register

Definition at line 1975 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.27 uint32_t MPU6050::getExternalSensorDWord (int *position*)

Read double word (4 bytes) from external sensor data registers.

Parameters

position	Starting position (0-20)

Returns

Double word read from registers

See also

getExternalSensorByte()

Definition at line 1993 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.28 uint16_t MPU6050::getExternalSensorWord (int *position*)

Read word (2 bytes) from external sensor data registers.

Parameters

position	Starting position (0-21)

Returns

Word read from register

See also

getExternalSensorByte()

Definition at line 1984 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.29 bool MPU6050::getExternalShadowDelayEnabled ( )

Get external data shadow delay enabled status.

This register is used to specify the timing of external sensor data shadowing. When DELAY_ES_SHADOW is set to 1, shadowing of external sensor data is delayed until all data has been received.

Returns

Current external data shadow delay enabled status.

See also

MPU6050_RA_I2C_MST_DELAY_CTRL
MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT

Definition at line 2089 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.30 uint8_t MPU6050::getFIFOByte ( )

3.2.3.31 void MPU6050::getFIFOBytes (uint8_t * *data*, uint8_t *length*)

3.2.3.32 uint16_t MPU6050::getFIFOCount ( )

3.2.3.33 bool MPU6050::getFIFOEnabled ( )

Get FIFO enabled status.

When this bit is set to 0, the FIFO buffer is disabled. The FIFO buffer cannot be written to or read from while disabled. The FIFO buffer's state does not change unless the MPU-60X0 is power cycled.

Returns

Current FIFO enabled status

See also

MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_FIFO_EN_BIT

Definition at line 2281 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.34 uint8_t MPU6050::getFreefallDetectionCounterDecrement ( )

Get Free Fall detection counter decrement configuration.

Detection is registered by the Free Fall detection module after accelerometer measurements meet their respective threshold conditions over a specified number of samples. When the threshold conditions are met, the corresponding detection counter increments by 1. The user may control the rate at which the detection counter decrements when the threshold condition is not met by configuring FF_COUNT. The decrement rate can be set according to the following table:

```
FF_COUNT | Counter Decrement
---------+------------------
0        | Reset
1        | 1
2        | 2
3        | 4
```

When FF_COUNT is configured to 0 (reset), any non-qualifying sample will reset the counter to 0. For further information on Free Fall detection, please refer to Registers 29 to 32.

Returns

Current decrement configuration

See also

MPU6050_RA_MOT_DETECT_CTRL
MPU6050_DETECT_FF_COUNT_BIT

Definition at line 2221 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.35 uint8_t MPU6050::getFreefallDetectionDuration ( )

Get free-fall event duration threshold.

This register configures the duration counter threshold for Free Fall event detection. The duration counter ticks at 1kHz, therefore FF_DUR has a unit of 1 LSB = 1 ms.

The Free Fall duration counter increments while the absolute value of the accelerometer measurements are each less than the detection threshold (Register 29). The Free Fall interrupt is triggered when the Free Fall duration counter reaches the time specified in this register.

For more details on the Free Fall detection interrupt, see Section 8.2 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Returns

Current free-fall duration threshold value (LSB = 1ms)

See also

MPU6050_RA_FF_DUR

Definition at line 429 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.36 uint8_t MPU6050::getFreefallDetectionThreshold ( )

Get free-fall event acceleration threshold.

This register configures the detection threshold for Free Fall event detection. The unit of FF_THR is 1LSB = 2mg. Free Fall is detected when the absolute value of the accelerometer measurements for the three axes are each less than the detection threshold. This condition increments the Free Fall duration counter (Register 30). The Free Fall interrupt is triggered when the Free Fall duration counter reaches the time specified in FF_DUR.

For more details on the Free Fall detection interrupt, see Section 8.2 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Returns

Current free-fall acceleration threshold value (LSB = 2mg)

See also

MPU6050_RA_FF_THR

Definition at line 397 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.37 bool MPU6050::getFSyncInterruptEnabled ( )

Get FSYNC pin interrupt enabled setting.

Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled setting

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_FSYNC_INT_EN_BIT

Definition at line 1410 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.38 bool MPU6050::getFSyncInterruptLevel ( )

Get FSYNC interrupt logic level mode.

Returns

Current FSYNC interrupt mode (0=active-high, 1=active-low)

See also

getFSyncInterruptMode()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT

Definition at line 1391 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.39 uint8_t MPU6050::getFullScaleAccelRange ( )

Get full-scale accelerometer range.

The FS_SEL parameter allows setting the full-scale range of the accelerometer sensors, as described in the table below.

```
0 = +/- 2g
1 = +/- 4g
2 = +/- 8g
3 = +/- 16g
```

Returns

Current full-scale accelerometer range setting

See also

MPU6050_ACCEL_FS_2
MPU6050_RA_ACCEL_CONFIG
MPU6050_ACONFIG_AFS_SEL_BIT
MPU6050_ACONFIG_AFS_SEL_LENGTH

Definition at line 320 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.40 uint8_t MPU6050::getFullScaleGyroRange ( )

Get full-scale gyroscope range.

The FS_SEL parameter allows setting the full-scale range of the gyro sensors, as described in the table below.

```
0 = +/- 250 degrees/sec
1 = +/- 500 degrees/sec
2 = +/- 1000 degrees/sec
3 = +/- 2000 degrees/sec
```

Returns

Current full-scale gyroscope range setting

See also

MPU6050_GYRO_FS_250
MPU6050_RA_GYRO_CONFIG
MPU6050_GCONFIG_FS_SEL_BIT
MPU6050_GCONFIG_FS_SEL_LENGTH

Definition at line 240 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.41 bool MPU6050::getI2CBypassEnabled ( )

Get I2C bypass enabled status.

When this bit is equal to 1 and I2C_MST_EN (Register 106 bit[5]) is equal to 0, the host application processor will be able to directly access the auxiliary I2C bus of the MPU-60X0. When this bit is equal to 0, the host application processor will not be able to directly access the auxiliary I2C bus of the MPU-60X0 regardless of the state of I2C_MST_EN (Register 106 bit[5]).

Returns

Current I2C bypass enabled status

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_I2C_BYPASS_EN_BIT

Definition at line 1434 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.42 bool MPU6050::getI2CMasterModeEnabled ( )

Get I2C Master Mode enabled status.

When this mode is enabled, the MPU-60X0 acts as the I2C Master to the external sensor slave devices on the auxiliary I2C bus. When this bit is cleared to 0, the auxiliary I2C bus lines (AUX_DA and AUX_CL) are logically driven by the primary I2C bus (SDA and SCL). This is a precondition to enabling Bypass Mode. For further information regarding Bypass Mode, please refer to Register 55.

Returns

Current I2C Master Mode enabled status

See also

MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_I2C_MST_EN_BIT

Definition at line 2305 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.43 bool MPU6050::getIntDataReadyEnabled ( )

Get Data Ready interrupt enabled setting.

This event occurs each time a write operation to all of the sensor registers has been completed. Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_DATA_RDY_BIT

Definition at line 1605 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.44 bool MPU6050::getIntDataReadyStatus ( )

Get Data Ready interrupt status.

This bit automatically sets to 1 when a Data Ready interrupt has been generated. The bit clears to 0 after the register has been read.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS
MPU6050_INTERRUPT_DATA_RDY_BIT

Definition at line 1695 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.45 bool MPU6050::getIntDMPEnabled ( )

3.2.3.46 bool MPU6050::getIntDMPStatus ( )

3.2.3.47 uint8_t MPU6050::getIntEnabled ( )

Get full interrupt enabled status.

Full register byte for all interrupts, for quick reading. Each bit will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_FF_BIT

Definition at line 1487 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.48 bool MPU6050::getInterruptDrive ( )

Get interrupt drive mode.

Will be set 0 for push-pull, 1 for open-drain.

Returns

Current interrupt drive mode (0=push-pull, 1=open-drain)

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_INT_OPEN_BIT

Definition at line 1334 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.49 bool MPU6050::getInterruptLatch ( )

Get interrupt latch mode.

Will be set 0 for 50us-pulse, 1 for latch-until-int-cleared.

Returns

Current latch mode (0=50us-pulse, 1=latch-until-int-cleared)

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_LATCH_INT_EN_BIT

Definition at line 1353 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.50 bool MPU6050::getInterruptLatchClear ( )

Get interrupt latch clear mode.

Will be set 0 for status-read-only, 1 for any-register-read.

Returns

Current latch clear mode (0=status-read-only, 1=any-register-read)

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_INT_RD_CLEAR_BIT

Definition at line 1372 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.51 bool MPU6050::getInterruptMode ( )

Get interrupt logic level mode.

Will be set 0 for active-high, 1 for active-low.

Returns

Current interrupt mode (0=active-high, 1=active-low)

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_INT_LEVEL_BIT

Definition at line 1315 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.52 bool MPU6050::getIntFIFOBufferOverflowEnabled ( )

Get FIFO Buffer Overflow interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_FIFO_OFLOW_BIT

Definition at line 1565 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.53 bool MPU6050::getIntFIFOBufferOverflowStatus ( )

Get FIFO Buffer Overflow interrupt status.

This bit automatically sets to 1 when a Free Fall interrupt has been generated. The bit clears to 0 after the register has been read.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS
MPU6050_INTERRUPT_FIFO_OFLOW_BIT

Definition at line 1672 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.54 bool MPU6050::getIntFreefallEnabled ( )

Get Free Fall interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_FF_BIT

Definition at line 1508 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.55 bool MPU6050::getIntFreefallStatus ( )

Get Free Fall interrupt status.

This bit automatically sets to 1 when a Free Fall interrupt has been generated. The bit clears to 0 after the register has been read.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS
MPU6050_INTERRUPT_FF_BIT

Definition at line 1639 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.56 bool MPU6050::getIntI2CMasterEnabled ( )

Get I2C Master interrupt enabled status.

This enables any of the I2C Master interrupt sources to generate an interrupt. Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_I2C_MST_INT_BIT

Definition at line 1585 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.57 bool MPU6050::getIntI2CMasterStatus ( )

Get I2C Master interrupt status.

This bit automatically sets to 1 when an I2C Master interrupt has been generated. For a list of I2C Master interrupts, please refer to Register 54. The bit clears to 0 after the register has been read.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS
MPU6050_INTERRUPT_I2C_MST_INT_BIT

Definition at line 1684 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.58 bool MPU6050::getIntMotionEnabled ( )

Get Motion Detection interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_MOT_BIT

Definition at line 1527 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.59 bool MPU6050::getIntMotionStatus ( )

Get Motion Detection interrupt status.

This bit automatically sets to 1 when a Motion Detection interrupt has been generated. The bit clears to 0 after the register has been read.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS
MPU6050_INTERRUPT_MOT_BIT

Definition at line 1650 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.60 bool MPU6050::getIntPLLReadyEnabled ( )

3.2.3.61 bool MPU6050::getIntPLLReadyStatus ( )

3.2.3.62 uint8_t MPU6050::getIntStatus ( )

Get full set of interrupt status bits.

These bits clear to 0 after the register has been read. Very useful for getting multiple INT statuses, since each single bit read clears all of them because it has to read the whole byte.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS

Definition at line 1628 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.63 bool MPU6050::getIntZeroMotionEnabled ( )

Get Zero Motion Detection interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Returns

Current interrupt enabled status

See also

MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_ZMOT_BIT

Definition at line 1546 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.64 bool MPU6050::getIntZeroMotionStatus ( )

Get Zero Motion Detection interrupt status.

This bit automatically sets to 1 when a Zero Motion Detection interrupt has been generated. The bit clears to 0 after the register has been read.

Returns

Current interrupt status

See also

MPU6050_RA_INT_STATUS
MPU6050_INTERRUPT_ZMOT_BIT

Definition at line 1661 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.65 bool MPU6050::getLostArbitration ( )

Get master arbitration lost status.

This bit automatically sets to 1 when the I2C Master has lost arbitration of the auxiliary I2C bus (an error condition). This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Returns

Master arbitration lost status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1247 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.66 uint8_t MPU6050::getMasterClockSpeed ( )

Get I2C master clock speed.

I2C_MST_CLK is a 4 bit unsigned value which configures a divider on the MPU-60X0 internal 8MHz clock. It sets the I2C master clock speed according to the following table:

```
I2C_MST_CLK | I2C Master Clock Speed | 8MHz Clock Divider
------------+------------------------+-------------------
0           | 348kHz                 | 23
1           | 333kHz                 | 24
2           | 320kHz                 | 25
3           | 308kHz                 | 26
4           | 296kHz                 | 27
5           | 286kHz                 | 28
```

```
6           | 276kHz                | 29
7           | 267kHz                | 30
8           | 258kHz                | 31
9           | 500kHz                | 16
10          | 471kHz                | 17
11          | 444kHz                | 18
12          | 421kHz                | 19
13          | 400kHz                | 20
14          | 381kHz                | 21
15          | 364kHz                | 22
```

Returns

Current I2C master clock speed

See also

MPU6050_RA_I2C_MST_CTRL

Definition at line 846 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.67 void MPU6050::getMotion6 (int16_t * *ax*, int16_t * *ay*, int16_t * *az*, int16_t * *gx*, int16_t * *gy*, int16_t * *gz*)

Get raw 6-axis motion sensor readings (accel/gyro).

Retrieves all currently available motion sensor values.

Parameters

ax	16-bit signed integer container for accelerometer X-axis value
ay	16-bit signed integer container for accelerometer Y-axis value
az	16-bit signed integer container for accelerometer Z-axis value
gx	16-bit signed integer container for gyroscope X-axis value
gy	16-bit signed integer container for gyroscope Y-axis value
gz	16-bit signed integer container for gyroscope Z-axis value

See also

getAcceleration()
getRotation()
MPU6050_RA_ACCEL_XOUT_H

Definition at line 1734 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.68 void MPU6050::getMotion9 (int16_t * *ax*, int16_t * *ay*, int16_t * *az*, int16_t * *gx*, int16_t * *gy*, int16_t * *gz*, int16_t * *mx*, int16_t * *my*, int16_t * *mz*)

Get raw 9-axis motion sensor readings (accel/gyro/compass).

FUNCTION NOT FULLY IMPLEMENTED YET.

Parameters

ax	16-bit signed integer container for accelerometer X-axis value
ay	16-bit signed integer container for accelerometer Y-axis value
az	16-bit signed integer container for accelerometer Z-axis value

gx	16-bit signed integer container for gyroscope X-axis value
gy	16-bit signed integer container for gyroscope Y-axis value
gz	16-bit signed integer container for gyroscope Z-axis value
mx	16-bit signed integer container for magnetometer X-axis value
my	16-bit signed integer container for magnetometer Y-axis value
mz	16-bit signed integer container for magnetometer Z-axis value

See also

getMotion6()
getAcceleration()
getRotation()
MPU6050_RA_ACCEL_XOUT_H

Definition at line 1718 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.69 uint8_t MPU6050::getMotionDetectionCounterDecrement ( )

Get Motion detection counter decrement configuration.

Detection is registered by the Motion detection module after accelerometer measurements meet their respective threshold conditions over a specified number of samples. When the threshold conditions are met, the corresponding detection counter increments by 1. The user may control the rate at which the detection counter decrements when the threshold condition is not met by configuring MOT_COUNT. The decrement rate can be set according to the following table:

```
MOT_COUNT | Counter Decrement
----------+------------------
0         | Reset
1         | 1
2         | 2
3         | 4
```

When MOT_COUNT is configured to 0 (reset), any non-qualifying sample will reset the counter to 0. For further information on Motion detection, please refer to Registers 29 to 32.

Definition at line 2257 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.70 uint8_t MPU6050::getMotionDetectionDuration ( )

Get motion detection event duration threshold.

This register configures the duration counter threshold for Motion interrupt generation. The duration counter ticks at 1 kHz, therefore MOT_DUR has a unit of 1LSB = 1ms. The Motion detection duration counter increments when the absolute value of any of the accelerometer measurements exceeds the Motion detection threshold (Register 31). The Motion detection interrupt is triggered when the Motion detection counter reaches the time count specified in this register.

For more details on the Motion detection interrupt, see Section 8.3 of the MPU-6000/MPU-6050 Product Specification document.

Returns

Current motion detection duration threshold value (LSB = 1ms)

See also

MPU6050_RA_MOT_DUR

Definition at line 493 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.71 uint8_t MPU6050::getMotionDetectionThreshold ( )

Get motion detection event acceleration threshold.

This register configures the detection threshold for Motion interrupt generation. The unit of MOT_THR is 1LSB = 2mg. Motion is detected when the absolute value of any of the accelerometer measurements exceeds this Motion detection threshold. This condition increments the Motion detection duration counter (Register 32). The Motion detection interrupt is triggered when the Motion Detection counter reaches the time count specified in MOT_DUR (Register 32).

The Motion interrupt will indicate the axis and polarity of detected motion in MOT_DETECT_STATUS (Register 97).

For more details on the Motion detection interrupt, see Section 8.3 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Returns

Current motion detection acceleration threshold value (LSB = 2mg)

See also

MPU6050_RA_MOT_THR

Definition at line 463 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.72 bool MPU6050::getMultiMasterEnabled ( )

Get multi-master enabled value.

Multi-master capability allows multiple I2C masters to operate on the same bus. In circuits where multi-master capability is required, set MULT_MST_EN to 1. This will increase current drawn by approximately 30uA.

In circuits where multi-master capability is required, the state of the I2C bus must always be monitored by each separate I2C Master. Before an I2C Master can assume arbitration of the bus, it must first confirm that no other I2C Master has arbitration of the bus. When MULT_MST_EN is set to 1, the MPU-60X0's bus arbitration detection logic is turned on, enabling it to detect when the bus is available.

Returns

Current multi-master enabled value

See also

MPU6050_RA_I2C_MST_CTRL

Definition at line 742 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.73 uint8_t MPU6050::getOTPBankValid ( )

3.2.3.74 bool MPU6050::getPassthroughStatus ( )

Get FSYNC interrupt status.

This bit reflects the status of the FSYNC interrupt from an external device into the MPU-60X0. This is used as a way to pass an external interrupt through the MPU-60X0 to the host application processor. When set to 1, this bit will cause an interrupt if FSYNC_INT_EN is asserted in INT_PIN_CFG (Register 55).

Returns

FSYNC interrupt status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1224 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.75 uint8_t MPU6050::getRate ( )

Get gyroscope output rate divider.

The sensor register output, FIFO output, DMP sampling, Motion detection, Zero Motion detection, and Free Fall detection are all based on the Sample Rate. The Sample Rate is generated by dividing the gyroscope output rate by SMPLRT_DIV:

Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)

where Gyroscope Output Rate = 8kHz when the DLPF is disabled (DLPF_CFG = 0 or 7), and 1kHz when the DLPF is enabled (see Register 26).

Note: The accelerometer output rate is 1kHz. This means that for a Sample Rate greater than 1kHz, the same accelerometer sample may be output to the FIFO, DMP, and sensor registers more than once.

For a diagram of the gyroscope and accelerometer signal paths, see Section 8 of the MPU-6000/MPU-6050 Product Specification document.

Returns

Current sample rate

See also

MPU6050_RA_SMPLRT_DIV

Definition at line 123 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.76 void MPU6050::getRotation (int16_t * x, int16_t * y, int16_t * z)

Get 3-axis gyroscope readings.

These gyroscope measurement registers, along with the accelerometer measurement registers, temperature measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set. The data within the gyroscope sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit gyroscope measurement has a full scale defined in FS_SEL (Register 27). For each full scale setting, the gyroscopes' sensitivity per LSB in GYRO_xOUT is shown in the table below:

```
FS_SEL | Full Scale Range   | LSB Sensitivity
-------+--------------------+----------------
0      | +/- 250 degrees/s  | 131 LSB/deg/s
1      | +/- 500 degrees/s  | 65.5 LSB/deg/s
2      | +/- 1000 degrees/s | 32.8 LSB/deg/s
3      | +/- 2000 degrees/s | 16.4 LSB/deg/s
```

Parameters

x	16-bit signed integer container for X-axis rotation
y	16-bit signed integer container for Y-axis rotation
z	16-bit signed integer container for Z-axis rotation

See also

getMotion6()
MPU6050_RA_GYRO_XOUT_H

Definition at line 1858 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.77 int16_t MPU6050::getRotationX ( )

Get X-axis gyroscope reading.

Returns

X-axis rotation measurement in 16-bit 2's complement format

See also

getMotion6()
MPU6050_RA_GYRO_XOUT_H

Definition at line 1876 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.78 void MPU6050::getRotationXY (int16_t * x, int16_t * y)

Definition at line 1865 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.79 int16_t MPU6050::getRotationY ( )

Get Y-axis gyroscope reading.

Returns

Y-axis rotation measurement in 16-bit 2's complement format

See also

getMotion6()
MPU6050_RA_GYRO_YOUT_H

Definition at line 1885 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.80 int16_t MPU6050::getRotationZ ( )

Get Z-axis gyroscope reading.

Returns

Z-axis rotation measurement in 16-bit 2's complement format

See also

getMotion6()
MPU6050_RA_GYRO_ZOUT_H

Definition at line 1894 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.81 uint8_t MPU6050::getSlate4InputByte ( )

Get last available byte read from Slave 4.

This register stores the data read from Slave 4. This field is populated after a read transaction.

Returns

Last available byte read from to Slave 4

See also

MPU6050_RA_I2C_SLV4_DI

Definition at line 1208 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.82 bool MPU6050::getSlave0FIFOEnabled ( )

Get Slave 0 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 0 to be written into the FIFO buffer.

Returns

Current Slave 0 FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 712 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.83 bool MPU6050::getSlave0Nack ( )

Get Slave 0 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 0. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Returns

Slave 0 NACK interrupt status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1302 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.84 bool MPU6050::getSlave1FIFOEnabled ( )

Get Slave 1 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 1 to be written into the FIFO buffer.

Returns

Current Slave 1 FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 694 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.85 bool MPU6050::getSlave1Nack ( )

Get Slave 1 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 1. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Returns

Slave 1 NACK interrupt status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1291 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.86 bool MPU6050::getSlave2FIFOEnabled ( )

Get Slave 2 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 2 to be written into the FIFO buffer.

Returns

Current Slave 2 FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 676 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.87 bool MPU6050::getSlave2Nack ( )

Get Slave 2 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 2. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Returns

Slave 2 NACK interrupt status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1280 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.88 bool MPU6050::getSlave3FIFOEnabled ( )

Get Slave 3 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 3 to be written into the FIFO buffer.

Returns

Current Slave 3 FIFO enabled value

See also

MPU6050_RA_MST_CTRL

Definition at line 783 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.89 bool MPU6050::getSlave3Nack ( )

Get Slave 3 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 3. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Returns

Slave 3 NACK interrupt status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1269 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.90 uint8_t MPU6050::getSlave4Address ( )

Get the I2C address of Slave 4.

Note that Bit 7 (MSB) controls read/write mode. If Bit 7 is set, it's a read operation, and if it is cleared, then it's a write operation. The remaining bits (6-0) are the 7-bit device address of the slave device.

Returns

Current address for Slave 4

See also

getSlaveAddress()
MPU6050_RA_I2C_SLV4_ADDR

Definition at line 1075 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.91 bool MPU6050::getSlave4Enabled ( )

Get the enabled value for the Slave 4.

When set to 1, this bit enables Slave 4 for data transfer operations. When cleared to 0, this bit disables Slave 4 from data transfer operations.

Returns

Current enabled value for Slave 4

See also

MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1121 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.92 bool MPU6050::getSlave4InterruptEnabled ( )

Get the enabled value for Slave 4 transaction interrupts.

When set to 1, this bit enables the generation of an interrupt signal upon completion of a Slave 4 transaction. When cleared to 0, this bit disables the generation of an interrupt signal upon completion of a Slave 4 transaction. The interrupt status can be observed in Register 54.

Returns

Current enabled value for Slave 4 transaction interrupts.

See also

MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1142 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.93 bool MPU6050::getSlave4IsDone ( )

Get Slave 4 transaction done status.

Automatically sets to 1 when a Slave 4 transaction has completed. This triggers an interrupt if the I2C_MST_IN↩T_EN bit in the INT_ENABLE register (Register 56) is asserted and if the SLV_4_DONE_INT bit is asserted in the I2C_SLV4_CTRL register (Register 52).

Returns

Slave 4 transaction done status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1236 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.94 uint8_t MPU6050::getSlave4MasterDelay ( )

Get Slave 4 master delay value.

This configures the reduced access rate of I2C slaves relative to the Sample Rate. When a slave's access rate is decreased relative to the Sample Rate, the slave is accessed every:

```
1 / (1 + I2C_MST_DLY) samples
```

This base Sample Rate in turn is determined by SMPLRT_DIV (register 25) and DLPF_CFG (register 26). Whether a slave's access rate is reduced relative to the Sample Rate is determined by I2C_MST_DELAY_CTRL (register 103). For further information regarding the Sample Rate, please refer to register 25.

Returns

Current Slave 4 master delay value

See also

MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1190 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.95 bool MPU6050::getSlave4Nack ( )

Get Slave 4 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 4. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Returns

Slave 4 NACK interrupt status

See also

MPU6050_RA_I2C_MST_STATUS

Definition at line 1258 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.96 uint8_t MPU6050::getSlave4Register ( )

Get the active internal register for the Slave 4.

Read/write operations for this slave will be done to whatever internal register address is stored in this MPU register.

Returns

Current active register for Slave 4

See also

MPU6050_RA_I2C_SLV4_REG

Definition at line 1094 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.97 bool MPU6050::getSlave4WriteMode ( )

Get write mode for Slave 4.

When set to 1, the transaction will read or write data only. When cleared to 0, the transaction will write a register address prior to reading or writing data. This should equal 0 when specifying the register address within the Slave device to/from which the ensuing data transaction will take place.

Returns

Current write mode for Slave 4 (0 = register address + data, 1 = data only)

See also

MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1163 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.98 uint8_t MPU6050::getSlaveAddress (uint8_t *num*)

Get the I2C address of the specified slave (0-3).

Note that Bit 7 (MSB) controls read/write mode. If Bit 7 is set, it's a read operation, and if it is cleared, then it's a write operation. The remaining bits (6-0) are the 7-bit device address of the slave device.

In read mode, the result of the read is placed in the lowest available EXT_SENS_DATA register. For further information regarding the allocation of read results, please refer to the EXT_SENS_DATA register description (Registers 73 - 96).

The MPU-6050 supports a total of five slaves, but Slave 4 has unique characteristics, and so it has its own functions (getSlave4∗ and setSlave4∗).

I2C data transactions are performed at the Sample Rate, as defined in Register 25. The user is responsible for ensuring that I2C data transactions to and from each enabled Slave can be completed within a single period of the Sample Rate.

The I2C slave access rate can be reduced relative to the Sample Rate. This reduced access rate is determined by I2C_MST_DLY (Register 52). Whether a slave's access rate is reduced relative to the Sample Rate is determined by I2C_MST_DELAY_CTRL (Register 103).

The processing order for the slaves is fixed. The sequence followed for processing the slaves is Slave 0, Slave 1, Slave 2, Slave 3 and Slave 4. If a particular Slave is disabled it will be skipped.

Each slave can either be accessed at the sample rate or at a reduced sample rate. In a case where some slaves are accessed at the Sample Rate and some slaves are accessed at the reduced rate, the sequence of accessing the slaves (Slave 0 to Slave 4) is still followed. However, the reduced rate slaves will be skipped if their access rate dictates that they should not be accessed during that particular cycle. For further information regarding the reduced access rate, please refer to Register 52. Whether a slave is accessed at the Sample Rate or at the reduced rate is determined by the Delay Enable bits in Register 103.

Parameters

num	Slave number (0-3)

Returns

Current address for specified slave

See also

MPU6050_RA_I2C_SLV0_ADDR

Definition at line 901 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.99 uint8_t MPU6050::getSlaveDataLength (uint8_t *num*)

Get number of bytes to read for the specified slave (0-3).

Specifies the number of bytes transferred to and from Slave 0. Clearing this bit to 0 is equivalent to disabling the register by writing 0 to I2C_SLV0_EN.

Parameters

num	Slave number (0-3)

Returns

Number of bytes to read for specified slave

See also

MPU6050_RA_I2C_SLV0_CTRL

Definition at line 1048 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.100 bool MPU6050::getSlaveDelayEnabled (uint8_t *num*)

Get slave delay enabled status.

When a particular slave delay is enabled, the rate of access for the that slave device is reduced. When a slave's access rate is decreased relative to the Sample Rate, the slave is accessed every:

```
1 / (1 + I2C_MST_DLY) Samples
```

This base Sample Rate in turn is determined by SMPLRT_DIV (register ∗ 25) and DLPF_CFG (register 26).

For further information regarding I2C_MST_DLY, please refer to register 52. For further information regarding the Sample Rate, please refer to register 25.

Parameters

num	Slave number (0-4)

Returns

Current slave delay enabled status.

See also

MPU6050_RA_I2C_MST_DELAY_CTRL
MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT

Definition at line 2120 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.101 bool MPU6050::getSlaveEnabled (uint8_t *num*)

Get the enabled value for the specified slave (0-3).

When set to 1, this bit enables Slave 0 for data transfer operations. When cleared to 0, this bit disables Slave 0 from data transfer operations.

Parameters

num	Slave number (0-3)

Returns

Current enabled value for specified slave

See also

MPU6050_RA_I2C_SLV0_CTRL

Definition at line 949 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.102 bool MPU6050::getSlaveReadWriteTransitionEnabled ( )

Get slave read/write transition enabled value.

The I2C_MST_P_NSR bit configures the I2C Master's transition from one slave read to the next slave read. If the bit equals 0, there will be a restart between reads. If the bit equals 1, there will be a stop followed by a start of the following read. When a write transaction follows a read transaction, the stop followed by a start of the successive write will be always used.

Returns

Current slave read/write transition enabled value

See also

MPU6050_RA_I2C_MST_CTRL

Definition at line 805 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.103 uint8_t MPU6050::getSlaveRegister (uint8_t *num*)

Get the active internal register for the specified slave (0-3).

Read/write operations for this slave will be done to whatever internal register address is stored in this MPU register.

The MPU-6050 supports a total of five slaves, but Slave 4 has unique characteristics, and so it has its own functions.

Parameters

num	Slave number (0-3)

Returns

Current active register for specified slave

See also

MPU6050_RA_I2C_SLV0_REG

Definition at line 927 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.104 bool MPU6050::getSlaveWordByteSwap (uint8_t *num*)

Get word pair byte-swapping enabled for the specified slave (0-3).

When set to 1, this bit enables byte swapping. When byte swapping is enabled, the high and low bytes of a word pair are swapped. Please refer to I2C_SLV0_GRP for the pairing convention of the word pairs. When cleared to 0, bytes transferred to and from Slave 0 will be written to EXT_SENS_DATA registers in the order they were transferred.

Parameters

num	Slave number (0-3)

Returns

Current word pair byte-swapping enabled value for specified slave

See also

MPU6050_RA_I2C_SLV0_CTRL

Definition at line 975 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.105 bool MPU6050::getSlaveWordGroupOffset (uint8_t *num*)

Get word pair grouping order offset for the specified slave (0-3).

This sets specifies the grouping order of word pairs received from registers. When cleared to 0, bytes from register addresses 0 and 1, 2 and 3, etc (even, then odd register addresses) are paired to form a word. When set to 1, bytes from register addresses are paired 1 and 2, 3 and 4, etc. (odd, then even register addresses) are paired to form a word.

Parameters

num	Slave number (0-3)

Returns

Current word pair grouping order offset for specified slave

See also

MPU6050_RA_I2C_SLV0_CTRL

Definition at line 1026 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.106 bool MPU6050::getSlaveWriteMode (uint8_t *num*)

Get write mode for the specified slave (0-3).

When set to 1, the transaction will read or write data only. When cleared to 0, the transaction will write a register address prior to reading or writing data. This should equal 0 when specifying the register address within the Slave device to/from which the ensuing data transaction will take place.

Parameters

num	Slave number (0-3)

Returns

Current write mode for specified slave (0 = register address + data, 1 = data only)

See also

MPU6050_RA_I2C_SLV0_CTRL

Definition at line 1000 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.107 bool MPU6050::getSleepEnabled ( )

Get sleep mode status.

Setting the SLEEP bit in the register puts the device into very low power sleep mode. In this mode, only the serial interface and internal registers remain active, allowing for a very low standby current. Clearing this bit puts the device back into normal mode. To save power, the individual standby selections for each of the gyros should be used if any gyro axis is not used by the application.

Returns

Current sleep mode enabled status

See also

MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_SLEEP_BIT

Definition at line 2380 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.108 bool MPU6050::getStandbyXAccelEnabled ( )

3.2.3.109 bool MPU6050::getStandbyXGyroEnabled ( )

3.2.3.110 bool MPU6050::getStandbyYAccelEnabled ( )

3.2.3.111 bool MPU6050::getStandbyYGyroEnabled ( )

3.2.3.112 bool MPU6050::getStandbyZAccelEnabled ( )

3.2.3.113 bool MPU6050::getStandbyZGyroEnabled ( )

3.2.3.114 int16_t MPU6050::getTemperature ( )

Get current internal temperature.

Returns

Temperature reading in 16-bit 2's complement format

See also

MPU6050_RA_TEMP_OUT_H

Definition at line 1819 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.115 bool MPU6050::getTempFIFOEnabled ( )

Get temperature FIFO enabled value.

When set to 1, this bit enables TEMP_OUT_H and TEMP_OUT_L (Registers 65 and 66) to be written into the FIFO buffer.

Returns

Current temperature FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 585 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.116 bool MPU6050::getTempSensorEnabled ( )

Get temperature sensor enabled status.

Control the usage of the internal temperature sensor.

Note: this register stores the *disabled* value, but for consistency with the rest of the code, the function is named and used with standard true/false values to indicate whether the sensor is enabled or disabled, respectively.

Returns

Current temperature sensor enabled status

See also

MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_TEMP_DIS_BIT

Definition at line 2425 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.117 bool MPU6050::getWaitForExternalSensorEnabled ( )

Get wait-for-external-sensor-data enabled value.

When the WAIT_FOR_ES bit is set to 1, the Data Ready interrupt will be delayed until External Sensor data from the Slave Devices are loaded into the EXT_SENS_DATA registers. This is used to ensure that both the internal sensor data (i.e. from gyro and accel) and external sensor data have been loaded to their respective data registers (i.e. the data is synced) when the Data Ready interrupt is triggered.

Returns

Current wait-for-external-sensor-data enabled value

See also

MPU6050_RA_I2C_MST_CTRL

Definition at line 765 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.118 bool MPU6050::getWakeCycleEnabled ( )

Get wake cycle enabled status.

When this bit is set to 1 and SLEEP is disabled, the MPU-60X0 will cycle between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by LP_WAKE_CTRL (register 108).

Returns

Current sleep mode enabled status

See also

MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_CYCLE_BIT

Definition at line 2401 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.119 uint8_t MPU6050::getWakeFrequency ( )

3.2.3.120 int16_t MPU6050::getXAccelOffset ( )

3.2.3.121 int8_t MPU6050::getXFineGain ( )

3.2.3.122 bool MPU6050::getXGyroFIFOEnabled ( )

Get gyroscope X-axis FIFO enabled value.

When set to 1, this bit enables GYRO_XOUT_H and GYRO_XOUT_L (Registers 67 and 68) to be written into the FIFO buffer.

Returns

Current gyroscope X-axis FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 603 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.123 int8_t MPU6050::getXGyroOffset ( )

3.2.3.124 int16_t MPU6050::getXGyroOffsetUser ( )

3.2.3.125 bool MPU6050::getXNegMotionDetected ( )

Get X-axis negative motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_XNEG_BIT

Definition at line 2005 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.126 bool MPU6050::getXPosMotionDetected ( )

Get X-axis positive motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_XPOS_BIT

Definition at line 2014 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.127 int16_t MPU6050::getYAccelOffset ( )

3.2.3.128 int8_t MPU6050::getYFineGain ( )

3.2.3.129 bool MPU6050::getYGyroFIFOEnabled ( )

Get gyroscope Y-axis FIFO enabled value.

When set to 1, this bit enables GYRO_YOUT_H and GYRO_YOUT_L (Registers 69 and 70) to be written into the FIFO buffer.

Returns

Current gyroscope Y-axis FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 621 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.130 int8_t MPU6050::getYGyroOffset ( )

3.2.3.131 int16_t MPU6050::getYGyroOffsetUser ( )

3.2.3.132 bool MPU6050::getYNegMotionDetected ( )

Get Y-axis negative motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_YNEG_BIT

Definition at line 2023 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.133 bool MPU6050::getYPosMotionDetected ( )

Get Y-axis positive motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_YPOS_BIT

Definition at line 2032 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.134 int16_t MPU6050::getZAccelOffset ( )

3.2.3.135 bool MPU6050::getZeroMotionDetected ( )

Get zero motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_ZRMOT_BIT

Definition at line 2059 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.136 uint8_t MPU6050::getZeroMotionDetectionDuration ( )

Get zero motion detection event duration threshold.

This register configures the duration counter threshold for Zero Motion interrupt generation. The duration counter ticks at 16 Hz, therefore ZRMOT_DUR has a unit of 1 LSB = 64 ms. The Zero Motion duration counter increments while the absolute value of the accelerometer measurements are each less than the detection threshold (Register 33). The Zero Motion interrupt is triggered when the Zero Motion duration counter reaches the time count specified in this register.

For more details on the Zero Motion detection interrupt, see Section 8.4 of the MPU-6000/MPU-6050 Product Specification document, as well as Registers 56 and 58 of this document.

Returns

Current zero motion detection duration threshold value (LSB = 64ms)

See also

MPU6050_RA_ZRMOT_DUR

Definition at line 564 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.137 uint8_t MPU6050::getZeroMotionDetectionThreshold ( )

Get zero motion detection event acceleration threshold.

This register configures the detection threshold for Zero Motion interrupt generation. The unit of ZRMOT_THR is 1LSB = 2mg. Zero Motion is detected when the absolute value of the accelerometer measurements for the 3 axes are each less than the detection threshold. This condition increments the Zero Motion duration counter (Register 34). The Zero Motion interrupt is triggered when the Zero Motion duration counter reaches the time count specified in ZRMOT_DUR (Register 34).

Unlike Free Fall or Motion detection, Zero Motion detection triggers an interrupt both when Zero Motion is first detected and when Zero Motion is no longer detected.

When a zero motion event is detected, a Zero Motion Status will be indicated in the MOT_DETECT_STATU↩ S register (Register 97). When a motion-to-zero-motion condition is detected, the status bit is set to 1. When a zero-motion-to- motion condition is detected, the status bit is set to 0.

For more details on the Zero Motion detection interrupt, see Section 8.4 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Returns

Current zero motion detection acceleration threshold value (LSB = 2mg)

See also

MPU6050_RA_ZRMOT_THR

Definition at line 533 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.138 int8_t MPU6050::getZFineGain ( )

3.2.3.139 bool MPU6050::getZGyroFIFOEnabled ( )

Get gyroscope Z-axis FIFO enabled value.

When set to 1, this bit enables GYRO_ZOUT_H and GYRO_ZOUT_L (Registers 71 and 72) to be written into the FIFO buffer.

Returns

Current gyroscope Z-axis FIFO enabled value

See also

MPU6050_RA_FIFO_EN

Definition at line 639 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.140 int8_t MPU6050::getZGyroOffset ( )

3.2.3.141 int16_t MPU6050::getZGyroOffsetUser ( )

3.2.3.142 bool MPU6050::getZNegMotionDetected ( )

Get Z-axis negative motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_ZNEG_BIT

Definition at line 2041 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.143 bool MPU6050::getZPosMotionDetected ( )

Get Z-axis positive motion detection interrupt status.

Returns

Motion detection status

See also

MPU6050_RA_MOT_DETECT_STATUS
MPU6050_MOTION_MOT_ZPOS_BIT

Definition at line 2050 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.144 void MPU6050::initialize ( )

Power on and prepare for general usage.

This will activate the device and take it out of sleep mode (which must be done after start-up). This function also sets both the accelerometer and the gyroscope to their most sensitive settings, namely +/- 2g and +/- 250 degrees/sec, and sets the clock source to use the X Gyro for reference, which is slightly better than the default internal clock source.

Definition at line 63 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.145 void MPU6050::readMemoryBlock (uint8_t * *data*, uint16_t *dataSize*, uint8_t *bank* = 0, uint8_t *address* = 0)

3.2.3.146 uint8_t MPU6050::readMemoryByte ( )

3.2.3.147 void MPU6050::reset ( )

Trigger a full device reset.

A small delay of ~50ms may be desirable after triggering a reset.

See also

MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_DEVICE_RESET_BIT

Definition at line 2366 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.148 void MPU6050::resetAccelerometerPath ( )

Reset accelerometer signal path.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

See also

MPU6050_RA_SIGNAL_PATH_RESET
MPU6050_PATHRESET_ACCEL_RESET_BIT

Definition at line 2153 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.149 void MPU6050::resetDMP ( )

3.2.3.150 void MPU6050::resetFIFO ( )

Reset the FIFO.

This bit resets the FIFO buffer when set to 1 while FIFO_EN equals 0. This bit automatically clears to 0 after the reset has been triggered.

See also

MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_FIFO_RESET_BIT

Definition at line 2331 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.151 void MPU6050::resetGyroscopePath ( )

Reset gyroscope signal path.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

See also

MPU6050_RA_SIGNAL_PATH_RESET
MPU6050_PATHRESET_GYRO_RESET_BIT

Definition at line 2144 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.152 void MPU6050::resetI2CMaster ( )

Reset the I2C Master.

This bit resets the I2C Master when set to 1 while I2C_MST_EN equals 0. This bit automatically clears to 0 after the reset has been triggered.

See also

MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_I2C_MST_RESET_BIT

Definition at line 2340 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.153 void MPU6050::resetSensors ( )

Reset all sensor registers and signal paths.

When set to 1, this bit resets the signal paths for all sensors (gyroscopes, accelerometers, and temperature sensor). This operation will also clear the sensor registers. This bit automatically clears to 0 after the reset has been triggered.

When resetting only the signal path (and not the sensor registers), please use Register 104, SIGNAL_PATH_RE↩ SET.

See also

MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_SIG_COND_RESET_BIT

Definition at line 2355 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.154 void MPU6050::resetTemperaturePath ( )

Reset temperature sensor signal path.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

See also

MPU6050_RA_SIGNAL_PATH_RESET
MPU6050_PATHRESET_TEMP_RESET_BIT

Definition at line 2162 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.155 void MPU6050::setAccelerometerPowerOnDelay (uint8_t *delay*)

Set accelerometer power-on delay.

Parameters

delay	New accelerometer power-on delay (0-3)

See also

getAccelerometerPowerOnDelay()
MPU6050_RA_MOT_DETECT_CTRL
MPU6050_DETECT_ACCEL_ON_DELAY_BIT

Definition at line 2192 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.156 void MPU6050::setAccelFIFOEnabled (bool *enabled*)

Set accelerometer FIFO enabled value.

Parameters

enabled	New accelerometer FIFO enabled value

See also

getAccelFIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 667 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.157 void MPU6050::setAccelXSelfTest (bool *enabled*)

Get self-test enabled setting for accelerometer X axis.

Parameters

enabled	Self-test enabled value

See also

MPU6050_RA_ACCEL_CONFIG

Definition at line 270 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.158 void MPU6050::setAccelYSelfTest (bool *enabled*)

Get self-test enabled value for accelerometer Y axis.

Parameters

enabled	Self-test enabled value

See also

MPU6050_RA_ACCEL_CONFIG

Definition at line 285 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.159 void MPU6050::setAccelZSelfTest (bool *enabled*)

Set self-test enabled value for accelerometer Z axis.

Parameters

enabled	Self-test enabled value

See also

MPU6050_RA_ACCEL_CONFIG

Definition at line 300 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.160 void MPU6050::setAuxVDDIOLevel (uint8_t *level*)

Set the auxiliary I2C supply voltage level.

When set to 1, the auxiliary I2C bus high logic level is VDD. When cleared to 0, the auxiliary I2C bus high logic level is VLOGIC. This does not apply to the MPU-6000, which does not have a VLOGIC pin.

Parameters

level	I2C supply voltage level (0=VLOGIC, 1=VDD)

Definition at line 96 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.161 void MPU6050::setClockOutputEnabled (bool *enabled*)

Set reference clock output enabled status.

When this bit is equal to 1, a reference clock output is provided at the CLKOUT pin. When this bit is equal to 0, the clock output is disabled. For further information regarding CLKOUT, please refer to the MPU-60X0 Product Specification document.

Parameters

enabled	New reference clock output enabled status

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_CLKOUT_EN_BIT

Definition at line 1474 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.162 void MPU6050::setClockSource (uint8_t *source*)

Set clock source setting.

An internal 8MHz oscillator, gyroscope based clock, or external sources can be selected as the MPU-60X0 clock source. When the internal 8 MHz oscillator or an external source is chosen as the clock source, the MPU-60X0 can operate in low power modes with the gyroscopes disabled.

Upon power up, the MPU-60X0 clock source defaults to the internal oscillator. However, it is highly recommended that the device be configured to use one of the gyroscopes (or an external clock source) as the clock reference for improved stability. The clock source can be selected according to the following table:

```
CLK_SEL | Clock Source
--------+--------------------------------------
0       | Internal oscillator
1       | PLL with X Gyro reference
2       | PLL with Y Gyro reference
3       | PLL with Z Gyro reference
4       | PLL with external 32.768kHz reference
5       | PLL with external 19.2MHz reference
6       | Reserved
7       | Stops the clock and keeps the timing generator in reset
```

Parameters

source	New clock source setting

See also

getClockSource()
MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_CLKSEL_BIT
MPU6050_PWR1_CLKSEL_LENGTH

Definition at line 2483 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.163 void MPU6050::setDeviceID (uint8_t *id*)

3.2.3.164 void MPU6050::setDHPFMode (uint8_t *bandwidth*)

Set the high-pass filter configuration.

Parameters

bandwidth	New high-pass filter configuration

See also

setDHPFMode()
MPU6050_DHPF_RESET
MPU6050_RA_ACCEL_CONFIG

Definition at line 376 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.165 void MPU6050::setDLPFMode (uint8_t *mode*)

Set digital low-pass filter configuration.

Parameters

mode	New DLFP configuration setting

See also

getDLPFBandwidth()
MPU6050_DLPF_BW_256
MPU6050_RA_CONFIG
MPU6050_CFG_DLPF_CFG_BIT
MPU6050_CFG_DLPF_CFG_LENGTH

Definition at line 217 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.166 void MPU6050::setDMPConfig1 (uint8_t *config*)

3.2.3.167 void MPU6050::setDMPConfig2 (uint8_t *config*)

3.2.3.168 void MPU6050::setDMPEnabled (bool *enabled*)

3.2.3.169 void MPU6050::setExternalFrameSync (uint8_t *sync*)

Set external FSYNC configuration.

See also

getExternalFrameSync()
MPU6050_RA_CONFIG

Parameters

sync	New FSYNC configuration value

Definition at line 174 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.170 void MPU6050::setExternalShadowDelayEnabled (bool *enabled*)

Set external data shadow delay enabled status.

Parameters

enabled	New external data shadow delay enabled status.

See also

getExternalShadowDelayEnabled()
MPU6050_RA_I2C_MST_DELAY_CTRL
MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT

Definition at line 2099 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.171 void MPU6050::setFIFOByte (uint8_t *data*)

3.2.3.172 void MPU6050::setFIFOEnabled (bool *enabled*)

Set FIFO enabled status.

Parameters

enabled	New FIFO enabled status

See also

getFIFOEnabled()
MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_FIFO_EN_BIT

Definition at line 2291 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.173 void MPU6050::setFreefallDetectionCounterDecrement (uint8_t *decrement*)

Set Free Fall detection counter decrement configuration.

Parameters

decrement	New decrement configuration value

See also

getFreefallDetectionCounterDecrement()
MPU6050_RA_MOT_DETECT_CTRL
MPU6050_DETECT_FF_COUNT_BIT

Definition at line 2231 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.174 void MPU6050::setFreefallDetectionDuration (uint8_t *duration*)

Get free-fall event duration threshold.

Parameters

duration	New free-fall duration threshold value (LSB = 1ms)

See also

getFreefallDetectionDuration()
MPU6050_RA_FF_DUR

Definition at line 438 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.175 void MPU6050::setFreefallDetectionThreshold (uint8_t *threshold*)

Get free-fall event acceleration threshold.

Parameters

threshold	New free-fall acceleration threshold value (LSB = 2mg)

See also

getFreefallDetectionThreshold()
MPU6050_RA_FF_THR

Definition at line 406 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.176 void MPU6050::setFSyncInterruptEnabled (bool *enabled*)

Set FSYNC pin interrupt enabled setting.

Parameters

enabled	New FSYNC pin interrupt enabled setting

See also

getFSyncInterruptEnabled()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_FSYNC_INT_EN_BIT

Definition at line 1420 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.177 void MPU6050::setFSyncInterruptLevel (bool *level*)

Set FSYNC interrupt logic level mode.

Parameters

mode	New FSYNC interrupt mode (0=active-high, 1=active-low)

See also

getFSyncInterruptMode()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT

Definition at line 1401 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.178 void MPU6050::setFullScaleAccelRange (uint8_t *range*)

Set full-scale accelerometer range.

Parameters

range	New full-scale accelerometer range setting

See also

getFullScaleAccelRange()

Definition at line 328 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.179 void MPU6050::setFullScaleGyroRange (uint8_t *range*)

Set full-scale gyroscope range.

Parameters

range	New full-scale gyroscope range value

See also

getFullScaleRange()
MPU6050_GYRO_FS_250
MPU6050_RA_GYRO_CONFIG
MPU6050_GCONFIG_FS_SEL_BIT
MPU6050_GCONFIG_FS_SEL_LENGTH

Definition at line 252 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.180 void MPU6050::setI2CBypassEnabled (bool *enabled*)

Set I2C bypass enabled status.

When this bit is equal to 1 and I2C_MST_EN (Register 106 bit[5]) is equal to 0, the host application processor will be able to directly access the auxiliary I2C bus of the MPU-60X0. When this bit is equal to 0, the host application processor will not be able to directly access the auxiliary I2C bus of the MPU-60X0 regardless of the state of I2C_MST_EN (Register 106 bit[5]).

Parameters

enabled	New I2C bypass enabled status

See also

MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_I2C_BYPASS_EN_BIT

Definition at line 1449 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.181 void MPU6050::setI2CMasterModeEnabled (bool *enabled*)

Set I2C Master Mode enabled status.

Parameters

enabled	New I2C Master Mode enabled status

See also

getI2CMasterModeEnabled()
MPU6050_RA_USER_CTRL
MPU6050_USERCTRL_I2C_MST_EN_BIT

Definition at line 2315 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.182 void MPU6050::setIntDataReadyEnabled (bool *enabled*)

Set Data Ready interrupt enabled status.

Parameters

enabled	New interrupt enabled status

See also

getIntDataReadyEnabled()
MPU6050_RA_INT_CFG
MPU6050_INTERRUPT_DATA_RDY_BIT

Definition at line 1615 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.183 void MPU6050::setIntDMPEnabled (bool *enabled*)

3.2.3.184 void MPU6050::setIntEnabled (uint8_t *enabled*)

Set full interrupt enabled status.

Full register byte for all interrupts, for quick reading. Each bit should be set 0 for disabled, 1 for enabled.

Parameters

enabled	New interrupt enabled status

See also

getIntFreefallEnabled()
MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_FF_BIT

Definition at line 1499 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.185 void MPU6050::setInterruptDrive (bool *drive*)

Set interrupt drive mode.

Parameters

drive	New interrupt drive mode (0=push-pull, 1=open-drain)

See also

getInterruptDrive()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_INT_OPEN_BIT

Definition at line 1344 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.186 void MPU6050::setInterruptLatch (bool *latch*)

Set interrupt latch mode.

Parameters

latch	New latch mode (0=50us-pulse, 1=latch-until-int-cleared)

See also

getInterruptLatch()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_LATCH_INT_EN_BIT

Definition at line 1363 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.187 void MPU6050::setInterruptLatchClear (bool *clear*)

Set interrupt latch clear mode.

Parameters

clear	New latch clear mode (0=status-read-only, 1=any-register-read)

See also

getInterruptLatchClear()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_INT_RD_CLEAR_BIT

Definition at line 1382 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.188 void MPU6050::setInterruptMode (bool *mode*)

Set interrupt logic level mode.

Parameters

mode	New interrupt mode (0=active-high, 1=active-low)

See also

getInterruptMode()
MPU6050_RA_INT_PIN_CFG
MPU6050_INTCFG_INT_LEVEL_BIT

Definition at line 1325 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.189 void MPU6050::setIntFIFOBufferOverflowEnabled (bool *enabled*)

Set FIFO Buffer Overflow interrupt enabled status.

Parameters

enabled	New interrupt enabled status

See also

getIntFIFOBufferOverflowEnabled()
MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_FIFO_OFLOW_BIT

Definition at line 1575 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.190 void MPU6050::setIntFreefallEnabled (bool *enabled*)

Set Free Fall interrupt enabled status.

Parameters

enabled	New interrupt enabled status

See also

getIntFreefallEnabled()
MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_FF_BIT

Definition at line 1518 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.191 void MPU6050::setIntI2CMasterEnabled (bool *enabled*)

Set I2C Master interrupt enabled status.

Parameters

enabled	New interrupt enabled status

See also

getIntI2CMasterEnabled()
MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_I2C_MST_INT_BIT

Definition at line 1595 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.192 void MPU6050::setIntMotionEnabled (bool *enabled*)

Set Motion Detection interrupt enabled status.

Parameters

enabled	New interrupt enabled status

See also

getIntMotionEnabled()
MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_MOT_BIT

Definition at line 1537 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.193 void MPU6050::setIntPLLReadyEnabled (bool *enabled*)

3.2.3.194 void MPU6050::setIntZeroMotionEnabled (bool *enabled*)

Set Zero Motion Detection interrupt enabled status.

Parameters

enabled	New interrupt enabled status

See also

getIntZeroMotionEnabled()
MPU6050_RA_INT_ENABLE
MPU6050_INTERRUPT_ZMOT_BIT

Definition at line 1556 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.195 void MPU6050::setMasterClockSpeed (uint8_t *speed*)

Set I2C master clock speed.

speed Current I2C master clock speed

See also

MPU6050_RA_I2C_MST_CTRL

Definition at line 854 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.196 void MPU6050::setMemoryBank (uint8_t *bank*, bool *prefetchEnabled* = `false`, bool *userBank* = `false`)

3.2.3.197 void MPU6050::setMemoryStartAddress (uint8_t *address*)

3.2.3.198 void MPU6050::setMotionDetectionCounterDecrement (uint8_t *decrement*)

Set Motion detection counter decrement configuration.

Parameters

decrement	New decrement configuration value

See also

getMotionDetectionCounterDecrement()
MPU6050_RA_MOT_DETECT_CTRL
MPU6050_DETECT_MOT_COUNT_BIT

Definition at line 2267 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.199 void MPU6050::setMotionDetectionDuration (uint8_t *duration*)

Set motion detection event duration threshold.

Parameters

duration	New motion detection duration threshold value (LSB = 1ms)

See also

getMotionDetectionDuration()
MPU6050_RA_MOT_DUR

Definition at line 502 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.200 void MPU6050::setMotionDetectionThreshold (uint8_t *threshold*)

Set free-fall event acceleration threshold.

Parameters

threshold	New motion detection acceleration threshold value (LSB = 2mg)

See also

getMotionDetectionThreshold()
MPU6050_RA_MOT_THR

Definition at line 472 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.201 void MPU6050::setMultiMasterEnabled (bool *enabled*)

Set multi-master enabled value.

Parameters

enabled	New multi-master enabled value

See also

getMultiMasterEnabled()
MPU6050_RA_I2C_MST_CTRL

Definition at line 751 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.202 void MPU6050::setOTPBankValid (bool *enabled*)

3.2.3.203 void MPU6050::setRate (uint8_t *rate*)

Set gyroscope sample rate divider.

Parameters

rate	New sample rate divider

See also

getRate()
MPU6050_RA_SMPLRT_DIV

Definition at line 132 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.204 void MPU6050::setSlave0FIFOEnabled (bool *enabled*)

Set Slave 0 FIFO enabled value.

Parameters

enabled	New Slave 0 FIFO enabled value

See also

getSlave0FIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 721 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.205 void MPU6050::setSlave1FIFOEnabled (bool *enabled*)

Set Slave 1 FIFO enabled value.

Parameters

enabled	New Slave 1 FIFO enabled value

See also

getSlave1FIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 703 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.206 void MPU6050::setSlave2FIFOEnabled (bool *enabled*)

Set Slave 2 FIFO enabled value.

Parameters

enabled	New Slave 2 FIFO enabled value

See also

getSlave2FIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 685 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.207 void MPU6050::setSlave3FIFOEnabled (bool *enabled*)

Set Slave 3 FIFO enabled value.

Parameters

enabled	New Slave 3 FIFO enabled value

See also

getSlave3FIFOEnabled()
MPU6050_RA_MST_CTRL

Definition at line 792 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.208 void MPU6050::setSlave4Address (uint8_t *address*)

Set the I2C address of Slave 4.

Parameters

address	New address for Slave 4

See also

getSlave4Address()
MPU6050_RA_I2C_SLV4_ADDR

Definition at line 1084 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.209 void MPU6050::setSlave4Enabled (bool *enabled*)

Set the enabled value for Slave 4.

Parameters

enabled	New enabled value for Slave 4

See also

getSlave4Enabled()
MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1130 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.210 void MPU6050::setSlave4InterruptEnabled (bool *enabled*)

Set the enabled value for Slave 4 transaction interrupts.

Parameters

enabled	New enabled value for Slave 4 transaction interrupts.

See also

getSlave4InterruptEnabled()
MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1151 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.211 void MPU6050::setSlave4MasterDelay (uint8_t *delay*)

Set Slave 4 master delay value.

Parameters

delay	New Slave 4 master delay value

See also

getSlave4MasterDelay()
MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1199 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.212 void MPU6050::setSlave4OutputByte (uint8_t *data*)

Set new byte to write to Slave 4.

This register stores the data to be written into the Slave 4. If I2C_SLV4_RW is set 1 (set to read), this register has no effect.

Parameters

data	New byte to write to Slave 4

See also

MPU6050_RA_I2C_SLV4_DO

Definition at line 1112 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.213 void MPU6050::setSlave4Register (uint8_t *reg*)

Set the active internal register for Slave 4.

Parameters

reg	New active register for Slave 4

See also

getSlave4Register()
MPU6050_RA_I2C_SLV4_REG

Definition at line 1103 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.214 void MPU6050::setSlave4WriteMode (bool *mode*)

Set write mode for the Slave 4.

Parameters

mode	New write mode for Slave 4 (0 = register address + data, 1 = data only)

See also

getSlave4WriteMode()
MPU6050_RA_I2C_SLV4_CTRL

Definition at line 1172 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.215 void MPU6050::setSlaveAddress (uint8_t *num*, uint8_t *address*)

Set the I2C address of the specified slave (0-3).

Parameters

num	Slave number (0-3)
address	New address for specified slave

See also

getSlaveAddress()
MPU6050_RA_I2C_SLV0_ADDR

Definition at line 912 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.216 void MPU6050::setSlaveDataLength (uint8_t *num*, uint8_t *length*)

Set number of bytes to read for the specified slave (0-3).

Parameters

num	Slave number (0-3)
length	Number of bytes to read for specified slave

See also

getSlaveDataLength()
MPU6050_RA_I2C_SLV0_CTRL

Definition at line 1059 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.217 void MPU6050::setSlaveDelayEnabled (uint8_t *num*, bool *enabled*)

Set slave delay enabled status.

Parameters

num	Slave number (0-4)
enabled	New slave delay enabled status.

See also

MPU6050_RA_I2C_MST_DELAY_CTRL
MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT

Definition at line 2132 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.218 void MPU6050::setSlaveEnabled (uint8_t *num*, bool *enabled*)

Set the enabled value for the specified slave (0-3).

Parameters

num	Slave number (0-3)
enabled	New enabled value for specified slave

See also

getSlaveEnabled()
MPU6050_RA_I2C_SLV0_CTRL

Definition at line 960 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.219 void MPU6050::setSlaveOutputByte (uint8_t *num*, uint8_t *data*)

Write byte to Data Output container for specified slave.

This register holds the output data written into Slave when Slave is set to write mode. For further information regarding Slave control, please refer to Registers 37 to 39 and immediately following.

Parameters

num	Slave number (0-3)
data	Byte to write

See also

MPU6050_RA_I2C_SLV0_DO

Definition at line 2074 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.220 void MPU6050::setSlaveReadWriteTransitionEnabled (bool *enabled*)

Set slave read/write transition enabled value.

Parameters

enabled	New slave read/write transition enabled value

See also

getSlaveReadWriteTransitionEnabled()
MPU6050_RA_I2C_MST_CTRL

Definition at line 814 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.221 void MPU6050::setSlaveRegister (uint8_t *num*, uint8_t *reg*)

Set the active internal register for the specified slave (0-3).

Parameters

num	Slave number (0-3)
reg	New active register for specified slave

See also

getSlaveRegister()
MPU6050_RA_I2C_SLV0_REG

Definition at line 938 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.222 void MPU6050::setSlaveWordByteSwap (uint8_t *num*, bool *enabled*)

Set word pair byte-swapping enabled for the specified slave (0-3).

Parameters

num	Slave number (0-3)
enabled	New word pair byte-swapping enabled value for specified slave

See also

getSlaveWordByteSwap()
MPU6050_RA_I2C_SLV0_CTRL

Definition at line 986 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.223 void MPU6050::setSlaveWordGroupOffset (uint8_t *num*, bool *enabled*)

Set word pair grouping order offset for the specified slave (0-3).

Parameters

num	Slave number (0-3)
enabled	New word pair grouping order offset for specified slave

See also

getSlaveWordGroupOffset()
MPU6050_RA_I2C_SLV0_CTRL

Definition at line 1037 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.224 void MPU6050::setSlaveWriteMode (uint8_t *num*, bool *mode*)

Set write mode for the specified slave (0-3).

Parameters

num	Slave number (0-3)
mode	New write mode for specified slave (0 = register address + data, 1 = data only)

See also

getSlaveWriteMode()
MPU6050_RA_I2C_SLV0_CTRL

Definition at line 1011 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.225 void MPU6050::setSleepEnabled (bool *enabled*)

Set sleep mode status.

Parameters

enabled	New sleep mode enabled status

See also

getSleepEnabled()
MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_SLEEP_BIT

Definition at line 2390 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.226 void MPU6050::setStandbyXAccelEnabled (bool *enabled*)

3.2.3.227 void MPU6050::setStandbyXGyroEnabled (bool *enabled*)

3.2.3.228 void MPU6050::setStandbyYAccelEnabled (bool *enabled*)

3.2.3.229 void MPU6050::setStandbyYGyroEnabled (bool *enabled*)

3.2.3.230 void MPU6050::setStandbyZAccelEnabled (bool *enabled*)

3.2.3.231 void MPU6050::setStandbyZGyroEnabled (bool *enabled*)

3.2.3.232 void MPU6050::setTempFIFOEnabled (bool *enabled*)

Set temperature FIFO enabled value.

Parameters

enabled	New temperature FIFO enabled value

See also

getTempFIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 594 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.233 void MPU6050::setTempSensorEnabled (bool *enabled*)

Set temperature sensor enabled status.

Note: this register stores the *disabled* value, but for consistency with the rest of the code, the function is named and used with standard true/false values to indicate whether the sensor is enabled or disabled, respectively.

Parameters

enabled	New temperature sensor enabled status

See also

getTempSensorEnabled()
MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_TEMP_DIS_BIT

Definition at line 2439 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.234 void MPU6050::setWaitForExternalSensorEnabled (bool *enabled*)

Set wait-for-external-sensor-data enabled value.

Parameters

enabled	New wait-for-external-sensor-data enabled value

See also

getWaitForExternalSensorEnabled()
MPU6050_RA_I2C_MST_CTRL

Definition at line 774 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.235 void MPU6050::setWakeCycleEnabled (bool *enabled*)

Set wake cycle enabled status.

Parameters

enabled	New sleep mode enabled status

See also

getWakeCycleEnabled()
MPU6050_RA_PWR_MGMT_1
MPU6050_PWR1_CYCLE_BIT

Definition at line 2411 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.236 void MPU6050::setWakeFrequency (uint8_t *frequency*)

3.2.3.237 void MPU6050::setXAccelOffset (int16_t *offset*)

3.2.3.238 void MPU6050::setXFineGain (int8_t *gain*)

3.2.3.239 void MPU6050::setXGyroFIFOEnabled (bool *enabled*)

Set gyroscope X-axis FIFO enabled value.

Parameters

enabled	New gyroscope X-axis FIFO enabled value

See also

getXGyroFIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 612 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.240 void MPU6050::setXGyroOffset (int8_t *offset*)

3.2.3.241 void MPU6050::setXGyroOffsetUser (int16_t *offset*)

3.2.3.242 void MPU6050::setYAccelOffset (int16_t *offset*)

3.2.3.243 void MPU6050::setYFineGain (int8_t *gain*)

3.2.3.244 void MPU6050::setYGyroFIFOEnabled (bool *enabled*)

Set gyroscope Y-axis FIFO enabled value.

Parameters

enabled	New gyroscope Y-axis FIFO enabled value

See also

getYGyroFIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 630 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.245 void MPU6050::setYGyroOffset (int8_t *offset*)

3.2.3.246 void MPU6050::setYGyroOffsetUser (int16_t *offset*)

3.2.3.247 void MPU6050::setZAccelOffset (int16_t *offset*)

3.2.3.248 void MPU6050::setZeroMotionDetectionDuration (uint8_t *duration*)

Set zero motion detection event duration threshold.

Parameters

duration	New zero motion detection duration threshold value (LSB = 1ms)

See also

getZeroMotionDetectionDuration()
MPU6050_RA_ZRMOT_DUR

Definition at line 573 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.249 void MPU6050::setZeroMotionDetectionThreshold (uint8_t *threshold*)

Set zero motion detection event acceleration threshold.

Parameters

threshold	New zero motion detection acceleration threshold value (LSB = 2mg)

See also

getZeroMotionDetectionThreshold()
MPU6050_RA_ZRMOT_THR

Definition at line 542 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.250 void MPU6050::setZFineGain (int8_t *gain*)

3.2.3.251 void MPU6050::setZGyroFIFOEnabled (bool *enabled*)

Set gyroscope Z-axis FIFO enabled value.

Parameters

enabled	New gyroscope Z-axis FIFO enabled value

See also

getZGyroFIFOEnabled()
MPU6050_RA_FIFO_EN

Definition at line 648 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.252 void MPU6050::setZGyroOffset (int8_t *offset*)

3.2.3.253 void MPU6050::setZGyroOffsetUser (int16_t *offset*)

3.2.3.254 void MPU6050::switchSPIEnabled (bool *enabled*)

Switch from I2C to SPI mode (MPU-6000 only) If this is set, the primary SPI interface will be enabled in place of the disabled primary I2C interface.

Definition at line 2322 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.255 bool MPU6050::testConnection ( )

Verify the I2C connection.

Make sure the device is connected and responds as expected.

Returns

True if connection is valid, false otherwise

Definition at line 74 of file ArduinoGyroscopeMPU6050.cpp.

3.2.3.256 bool MPU6050::writeDMPConfigurationSet (const uint8_t * *data*, uint16_t *dataSize*, bool *useProgMem* = `false`)

3.2.3.257 bool MPU6050::writeMemoryBlock (const uint8_t * *data*, uint16_t *dataSize*, uint8_t *bank* = `0`, uint8_t *address* = `0`, bool *verify* = `true`, bool *useProgMem* = `false`)

3.2.3.258 void MPU6050::writeMemoryByte (uint8_t *data*)

3.2.3.259 bool MPU6050::writeProgDMPConfigurationSet (const uint8_t * *data*, uint16_t *dataSize*)

3.2.3.260 bool MPU6050::writeProgMemoryBlock (const uint8_t * *data*, uint16_t *dataSize*, uint8_t *bank* = `0`, uint8_t *address* = `0`, bool *verify* = `true`)

3.2.4 Member Data Documentation

3.2.4.1 uint8_t MPU6050::buffer[14] `[private]`

Definition at line 988 of file ArduinoGyroscopeMPU6050.h.

3.2.4.2 uint8_t MPU6050::devAddr `[private]`

Definition at line 987 of file ArduinoGyroscopeMPU6050.h.

The documentation for this class was generated from the following files:

- ArduinoGyroscopeMPU6050.h
- ArduinoGyroscopeMPU6050.cpp

4 File Documentation

4.1 ArduinoGyroscope.cpp File Reference

4.2 ArduinoGyroscope.cpp

4.3 ArduinoGyroscope.h File Reference

Classes

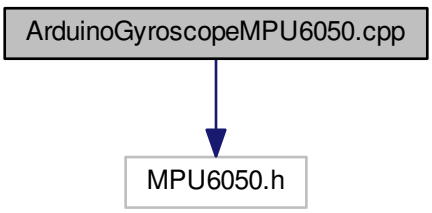
- class ArduinoGyroscope

4.4 ArduinoGyroscope.h

```
00001
00009 #ifndef __ARDUINO_DRIVER_GYROSCOPE_H__
00010 #define __ARDUINO_DRIVER_GYROSCOPE_H__ 1
00011
00012 class ArduinoGyroscope {
00013
00014     virtual int getRotationX() = 0;
00015
00016     virtual int getRotationY() = 0;
00017
00018     virtual int getRotationZ() = 0;
00019 };
00020
00021 #endif /* __ARDUINO_DRIVER_GYROSCOPE_H__ */
```

4.5 ArduinoGyroscopeMPU6050.cpp File Reference

`#include "MPU6050.h"`
Include dependency graph for ArduinoGyroscopeMPU6050.cpp:



4.6 ArduinoGyroscopeMPU6050.cpp

```
00001 // I2Cdev library collection - MPU6050 I2C device class
00002 // Based on InvenSense MPU-6050 register map document rev. 2.0, 5/19/2011 (RM-MPU-6000A-00)
00003 // 8/24/2011 by Jeff Rowberg <jeff@rowberg.net>
00004 // Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
00005 //
00006 // Changelog:
00007 //     ... - ongoing debug release
00008
00009 // NOTE: THIS IS ONLY A PARIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING ACTIVE
```

```
00010 // DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP THIS IN MIND IF
00011 // YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.
00012
00013 /* ============================================
00014 I2Cdev device library code is placed under the MIT license
00015 Copyright (c) 2012 Jeff Rowberg
00016
00017 Permission is hereby granted, free of charge, to any person obtaining a copy
00018 of this software and associated documentation files (the "Software"), to deal
00019 in the Software without restriction, including without limitation the rights
00020 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00021 copies of the Software, and to permit persons to whom the Software is
00022 furnished to do so, subject to the following conditions:
00023
00024 The above copyright notice and this permission notice shall be included in
00025 all copies or substantial portions of the Software.
00026
00027 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00028 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00029 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00030 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00031 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00032 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00033 THE SOFTWARE.
00034 ===============================================
00035 */
00036
00037 #include "MPU6050.h"
00038
00042 MPU6050::MPU6050() {
00043     devAddr = MPU6050_DEFAULT_ADDRESS;
00044 }
00045
00052 MPU6050::MPU6050(uint8_t address) {
00053     devAddr = address;
00054 }
00055
00063 void MPU6050::initialize() {
00064     setClockSource(MPU6050_CLOCK_PLL_XGYRO);
00065     setFullScaleGyroRange(MPU6050_GYRO_FS_250);
00066     setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
00067     setSleepEnabled(false); // thanks to Jack Elston for pointing this one out!
00068 }
00069
00074 bool MPU6050::testConnection() {
00075     return getDeviceID() == 0x34;
00076 }
00077
00078 // AUX_VDDIO register (InvenSense demo code calls this RA_*G_OFFS_TC)
00079
00086 uint8_t MPU6050::getAuxVDDIOLevel() {
00087     I2Cdev::readBit(devAddr, MPU6050_RA_YG_OFFS_TC,
      MPU6050_TC_PWR_MODE_BIT, buffer);
00088     return buffer[0];
00089 }
00096 void MPU6050::setAuxVDDIOLevel(uint8_t level) {
00097     I2Cdev::writeBit(devAddr, MPU6050_RA_YG_OFFS_TC,
      MPU6050_TC_PWR_MODE_BIT, level);
00098 }
00099
00100 // SMPLRT_DIV register
00101
00123 uint8_t MPU6050::getRate() {
00124     I2Cdev::readByte(devAddr, MPU6050_RA_SMPLRT_DIV,
      buffer);
00125     return buffer[0];
00126 }
00132 void MPU6050::setRate(uint8_t rate) {
00133     I2Cdev::writeByte(devAddr, MPU6050_RA_SMPLRT_DIV, rate);
00134 }
00135
00136 // CONFIG register
00137
00165 uint8_t MPU6050::getExternalFrameSync() {
00166     I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG,
      MPU6050_CFG_EXT_SYNC_SET_BIT,
      MPU6050_CFG_EXT_SYNC_SET_LENGTH, buffer);
00167     return buffer[0];
00168 }
00174 void MPU6050::setExternalFrameSync(uint8_t sync) {
00175     I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG,
      MPU6050_CFG_EXT_SYNC_SET_BIT,
      MPU6050_CFG_EXT_SYNC_SET_LENGTH, sync);
00176 }
00205 uint8_t MPU6050::getDLPFMode() {
00206     I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG,
      MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH,
```

```
      buffer);
00207     return buffer[0];
00208 }
00217 void MPU6050::setDLPFMode(uint8_t mode) {
00218     I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG,
      MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH,
      mode);
00219 }
00220
00221 // GYRO_CONFIG register
00222
00240 uint8_t MPU6050::getFullScaleGyroRange() {
00241     I2Cdev::readBits(devAddr, MPU6050_RA_GYRO_CONFIG,
      MPU6050_GCONFIG_FS_SEL_BIT,
      MPU6050_GCONFIG_FS_SEL_LENGTH, buffer);
00242     return buffer[0];
00243 }
00252 void MPU6050::setFullScaleGyroRange(uint8_t range) {
00253     I2Cdev::writeBits(devAddr, MPU6050_RA_GYRO_CONFIG,
      MPU6050_GCONFIG_FS_SEL_BIT,
      MPU6050_GCONFIG_FS_SEL_LENGTH, range);
00254 }
00255
00256 // ACCEL_CONFIG register
00257
00262 bool MPU6050::getAccelXSelfTest() {
00263     I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_XA_ST_BIT, buffer);
00264     return buffer[0];
00265 }
00270 void MPU6050::setAccelXSelfTest(bool enabled) {
00271     I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_XA_ST_BIT, enabled);
00272 }
00277 bool MPU6050::getAccelYSelfTest() {
00278     I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_YA_ST_BIT, buffer);
00279     return buffer[0];
00280 }
00285 void MPU6050::setAccelYSelfTest(bool enabled) {
00286     I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_YA_ST_BIT, enabled);
00287 }
00292 bool MPU6050::getAccelZSelfTest() {
00293     I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_ZA_ST_BIT, buffer);
00294     return buffer[0];
00295 }
00300 void MPU6050::setAccelZSelfTest(bool enabled) {
00301     I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_ZA_ST_BIT, enabled);
00302 }
00320 uint8_t MPU6050::getFullScaleAccelRange() {
00321     I2Cdev::readBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_AFS_SEL_BIT,
      MPU6050_ACONFIG_AFS_SEL_LENGTH, buffer);
00322     return buffer[0];
00323 }
00328 void MPU6050::setFullScaleAccelRange(uint8_t range) {
00329     I2Cdev::writeBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_AFS_SEL_BIT,
      MPU6050_ACONFIG_AFS_SEL_LENGTH, range);
00330 }
00366 uint8_t MPU6050::getDHPFMode() {
00367     I2Cdev::readBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_ACCEL_HPF_BIT,
      MPU6050_ACONFIG_ACCEL_HPF_LENGTH, buffer);
00368     return buffer[0];
00369 }
00376 void MPU6050::setDHPFMode(uint8_t bandwidth) {
00377     I2Cdev::writeBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
      MPU6050_ACONFIG_ACCEL_HPF_BIT,
      MPU6050_ACONFIG_ACCEL_HPF_LENGTH, bandwidth);
00378 }
00379
00380 // FF_THR register
00381
00397 uint8_t MPU6050::getFreefallDetectionThreshold() {
00398     I2Cdev::readByte(devAddr, MPU6050_RA_FF_THR, buffer);
00399     return buffer[0];
00400 }
00406 void MPU6050::setFreefallDetectionThreshold(uint8_t threshold) {
00407     I2Cdev::writeByte(devAddr, MPU6050_RA_FF_THR, threshold);
00408 }
00409
00410 // FF_DUR register
00411
```

```
00429 uint8_t MPU6050::getFreefallDetectionDuration() {
00430     I2Cdev::readByte(devAddr, MPU6050_RA_FF_DUR, buffer);
00431     return buffer[0];
00432 }
00438 void MPU6050::setFreefallDetectionDuration(uint8_t duration) {
00439     I2Cdev::writeByte(devAddr, MPU6050_RA_FF_DUR, duration);
00440 }
00441
00442 // MOT_THR register
00443
00463 uint8_t MPU6050::getMotionDetectionThreshold() {
00464     I2Cdev::readByte(devAddr, MPU6050_RA_MOT_THR,
      buffer);
00465     return buffer[0];
00466 }
00472 void MPU6050::setMotionDetectionThreshold(uint8_t threshold) {
00473     I2Cdev::writeByte(devAddr, MPU6050_RA_MOT_THR, threshold);
00474 }
00475
00476 // MOT_DUR register
00477
00493 uint8_t MPU6050::getMotionDetectionDuration() {
00494     I2Cdev::readByte(devAddr, MPU6050_RA_MOT_DUR,
      buffer);
00495     return buffer[0];
00496 }
00502 void MPU6050::setMotionDetectionDuration(uint8_t duration) {
00503     I2Cdev::writeByte(devAddr, MPU6050_RA_MOT_DUR, duration);
00504 }
00505
00506 // ZRMOT_THR register
00507
00533 uint8_t MPU6050::getZeroMotionDetectionThreshold() {
00534     I2Cdev::readByte(devAddr, MPU6050_RA_ZRMOT_THR,
      buffer);
00535     return buffer[0];
00536 }
00542 void MPU6050::setZeroMotionDetectionThreshold(uint8_t threshold) {
00543     I2Cdev::writeByte(devAddr, MPU6050_RA_ZRMOT_THR, threshold);
00544 }
00545
00546 // ZRMOT_DUR register
00547
00564 uint8_t MPU6050::getZeroMotionDetectionDuration() {
00565     I2Cdev::readByte(devAddr, MPU6050_RA_ZRMOT_DUR,
      buffer);
00566     return buffer[0];
00567 }
00573 void MPU6050::setZeroMotionDetectionDuration(uint8_t duration) {
00574     I2Cdev::writeByte(devAddr, MPU6050_RA_ZRMOT_DUR, duration);
00575 }
00576
00577 // FIFO_EN register
00578
00585 bool MPU6050::getTempFIFOEnabled() {
00586     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_TEMP_FIFO_EN_BIT, buffer);
00587     return buffer[0];
00588 }
00594 void MPU6050::setTempFIFOEnabled(bool enabled) {
00595     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_TEMP_FIFO_EN_BIT, enabled);
00596 }
00603 bool MPU6050::getXGyroFIFOEnabled() {
00604     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_XG_FIFO_EN_BIT, buffer);
00605     return buffer[0];
00606 }
00612 void MPU6050::setXGyroFIFOEnabled(bool enabled) {
00613     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_XG_FIFO_EN_BIT, enabled);
00614 }
00621 bool MPU6050::getYGyroFIFOEnabled() {
00622     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_YG_FIFO_EN_BIT, buffer);
00623     return buffer[0];
00624 }
00630 void MPU6050::setYGyroFIFOEnabled(bool enabled) {
00631     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_YG_FIFO_EN_BIT, enabled);
00632 }
00639 bool MPU6050::getZGyroFIFOEnabled() {
00640     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_ZG_FIFO_EN_BIT, buffer);
00641     return buffer[0];
00642 }
00648 void MPU6050::setZGyroFIFOEnabled(bool enabled) {
```

```
00649     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_ZG_FIFO_EN_BIT, enabled);
00650 }
00658 bool MPU6050::getAccelFIFOEnabled() {
00659     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_ACCEL_FIFO_EN_BIT, buffer);
00660     return buffer[0];
00661 }
00667 void MPU6050::setAccelFIFOEnabled(bool enabled) {
00668     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_ACCEL_FIFO_EN_BIT, enabled);
00669 }
00676 bool MPU6050::getSlave2FIFOEnabled() {
00677     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_SLV2_FIFO_EN_BIT, buffer);
00678     return buffer[0];
00679 }
00685 void MPU6050::setSlave2FIFOEnabled(bool enabled) {
00686     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_SLV2_FIFO_EN_BIT, enabled);
00687 }
00694 bool MPU6050::getSlave1FIFOEnabled() {
00695     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_SLV1_FIFO_EN_BIT, buffer);
00696     return buffer[0];
00697 }
00703 void MPU6050::setSlave1FIFOEnabled(bool enabled) {
00704     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_SLV1_FIFO_EN_BIT, enabled);
00705 }
00712 bool MPU6050::getSlave0FIFOEnabled() {
00713     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_SLV0_FIFO_EN_BIT, buffer);
00714     return buffer[0];
00715 }
00721 void MPU6050::setSlave0FIFOEnabled(bool enabled) {
00722     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
      MPU6050_SLV0_FIFO_EN_BIT, enabled);
00723 }
00724
00725 // I2C_MST_CTRL register
00726
00742 bool MPU6050::getMultiMasterEnabled() {
00743     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_MULT_MST_EN_BIT, buffer);
00744     return buffer[0];
00745 }
00751 void MPU6050::setMultiMasterEnabled(bool enabled) {
00752     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_MULT_MST_EN_BIT, enabled);
00753 }
00765 bool MPU6050::getWaitForExternalSensorEnabled() {
00766     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_WAIT_FOR_ES_BIT, buffer);
00767     return buffer[0];
00768 }
00774 void MPU6050::setWaitForExternalSensorEnabled(bool enabled) {
00775     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_WAIT_FOR_ES_BIT, enabled);
00776 }
00783 bool MPU6050::getSlave3FIFOEnabled() {
00784     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_SLV_3_FIFO_EN_BIT, buffer);
00785     return buffer[0];
00786 }
00792 void MPU6050::setSlave3FIFOEnabled(bool enabled) {
00793     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_SLV_3_FIFO_EN_BIT, enabled);
00794 }
00805 bool MPU6050::getSlaveReadWriteTransitionEnabled() {
00806     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_I2C_MST_P_NSR_BIT, buffer);
00807     return buffer[0];
00808 }
00814 void MPU6050::setSlaveReadWriteTransitionEnabled(bool enabled) {
00815     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_I2C_MST_P_NSR_BIT, enabled);
00816 }
00846 uint8_t MPU6050::getMasterClockSpeed() {
00847     I2Cdev::readBits(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_I2C_MST_CLK_BIT, MPU6050_I2C_MST_CLK_LENGTH,
      buffer);
00848     return buffer[0];
00849 }
00854 void MPU6050::setMasterClockSpeed(uint8_t speed) {
00855     I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_MST_CTRL,
      MPU6050_I2C_MST_CLK_BIT, MPU6050_I2C_MST_CLK_LENGTH, speed
```

```
    );
00856 }
00857
00858 // I2C_SLV* registers (Slave 0-3)
00859
00901 uint8_t MPU6050::getSlaveAddress(uint8_t num) {
00902     if (num > 3) return 0;
00903     I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV0_ADDR + num*3,
    buffer);
00904     return buffer[0];
00905 }
00912 void MPU6050::setSlaveAddress(uint8_t num, uint8_t address) {
00913     if (num > 3) return;
00914     I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_ADDR + num*3, address);
00915 }
00927 uint8_t MPU6050::getSlaveRegister(uint8_t num) {
00928     if (num > 3) return 0;
00929     I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV0_REG + num*3,
    buffer);
00930     return buffer[0];
00931 }
00938 void MPU6050::setSlaveRegister(uint8_t num, uint8_t reg) {
00939     if (num > 3) return;
00940     I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_REG + num*3, reg);
00941 }
00949 bool MPU6050::getSlaveEnabled(uint8_t num) {
00950     if (num > 3) return 0;
00951     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_EN_BIT, buffer);
00952     return buffer[0];
00953 }
00960 void MPU6050::setSlaveEnabled(uint8_t num, bool enabled) {
00961     if (num > 3) return;
00962     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_EN_BIT, enabled);
00963 }
00975 bool MPU6050::getSlaveWordByteSwap(uint8_t num) {
00976     if (num > 3) return 0;
00977     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_BYTE_SW_BIT, buffer);
00978     return buffer[0];
00979 }
00986 void MPU6050::setSlaveWordByteSwap(uint8_t num, bool enabled) {
00987     if (num > 3) return;
00988     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_BYTE_SW_BIT, enabled);
00989 }
01000 bool MPU6050::getSlaveWriteMode(uint8_t num) {
01001     if (num > 3) return 0;
01002     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_REG_DIS_BIT, buffer);
01003     return buffer[0];
01004 }
01011 void MPU6050::setSlaveWriteMode(uint8_t num, bool mode) {
01012     if (num > 3) return;
01013     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_REG_DIS_BIT, mode);
01014 }
01026 bool MPU6050::getSlaveWordGroupOffset(uint8_t num) {
01027     if (num > 3) return 0;
01028     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_GRP_BIT, buffer);
01029     return buffer[0];
01030 }
01037 void MPU6050::setSlaveWordGroupOffset(uint8_t num, bool enabled) {
01038     if (num > 3) return;
01039     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_GRP_BIT, enabled);
01040 }
01048 uint8_t MPU6050::getSlaveDataLength(uint8_t num) {
01049     if (num > 3) return 0;
01050     I2Cdev::readBits(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_LEN_BIT, MPU6050_I2C_SLV_LEN_LENGTH,
    buffer);
01051     return buffer[0];
01052 }
01059 void MPU6050::setSlaveDataLength(uint8_t num, uint8_t length) {
01060     if (num > 3) return;
01061     I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
    MPU6050_I2C_SLV_LEN_BIT, MPU6050_I2C_SLV_LEN_LENGTH,
    length);
01062 }
01063
01064 // I2C_SLV* registers (Slave 4)
01065
01075 uint8_t MPU6050::getSlave4Address() {
01076     I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_ADDR,
```

```
      buffer);
01077     return buffer[0];
01078 }
01084 void MPU6050::setSlave4Address(uint8_t address) {
01085     I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_ADDR, address);
01086 }
01094 uint8_t MPU6050::getSlave4Register() {
01095     I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_REG,
      buffer);
01096     return buffer[0];
01097 }
01103 void MPU6050::setSlave4Register(uint8_t reg) {
01104     I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_REG, reg);
01105 }
01112 void MPU6050::setSlave4OutputByte(uint8_t data) {
01113     I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_DO, data);
01114 }
01121 bool MPU6050::getSlave4Enabled() {
01122     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_EN_BIT, buffer);
01123     return buffer[0];
01124 }
01130 void MPU6050::setSlave4Enabled(bool enabled) {
01131     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_EN_BIT, enabled);
01132 }
01142 bool MPU6050::getSlave4InterruptEnabled() {
01143     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_INT_EN_BIT, buffer);
01144     return buffer[0];
01145 }
01151 void MPU6050::setSlave4InterruptEnabled(bool enabled) {
01152     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_INT_EN_BIT, enabled);
01153 }
01163 bool MPU6050::getSlave4WriteMode() {
01164     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_REG_DIS_BIT, buffer);
01165     return buffer[0];
01166 }
01172 void MPU6050::setSlave4WriteMode(bool mode) {
01173     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_REG_DIS_BIT, mode);
01174 }
01190 uint8_t MPU6050::getSlave4MasterDelay() {
01191     I2Cdev::readBits(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_MST_DLY_BIT,
      MPU6050_I2C_SLV4_MST_DLY_LENGTH, buffer);
01192     return buffer[0];
01193 }
01199 void MPU6050::setSlave4MasterDelay(uint8_t delay) {
01200     I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
      MPU6050_I2C_SLV4_MST_DLY_BIT,
      MPU6050_I2C_SLV4_MST_DLY_LENGTH, delay);
01201 }
01208 uint8_t MPU6050::getSlate4InputByte() {
01209     I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_DI,
      buffer);
01210     return buffer[0];
01211 }
01212
01213 // I2C_MST_STATUS register
01214
01224 bool MPU6050::getPassthroughStatus() {
01225     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_PASS_THROUGH_BIT, buffer);
01226     return buffer[0];
01227 }
01236 bool MPU6050::getSlave4IsDone() {
01237     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_SLV4_DONE_BIT, buffer);
01238     return buffer[0];
01239 }
01247 bool MPU6050::getLostArbitration() {
01248     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_LOST_ARB_BIT, buffer);
01249     return buffer[0];
01250 }
01258 bool MPU6050::getSlave4Nack() {
01259     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_SLV4_NACK_BIT, buffer);
01260     return buffer[0];
01261 }
01269 bool MPU6050::getSlave3Nack() {
01270     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_SLV3_NACK_BIT, buffer);
01271     return buffer[0];
```

```
01272 }
01280 bool MPU6050::getSlave2Nack() {
01281     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_SLV2_NACK_BIT, buffer);
01282     return buffer[0];
01283 }
01291 bool MPU6050::getSlave1Nack() {
01292     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_SLV1_NACK_BIT, buffer);
01293     return buffer[0];
01294 }
01302 bool MPU6050::getSlave0Nack() {
01303     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
      MPU6050_MST_I2C_SLV0_NACK_BIT, buffer);
01304     return buffer[0];
01305 }
01306
01307 // INT_PIN_CFG register
01308
01315 bool MPU6050::getInterruptMode() {
01316     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_INT_LEVEL_BIT, buffer);
01317     return buffer[0];
01318 }
01325 void MPU6050::setInterruptMode(bool mode) {
01326    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_INT_LEVEL_BIT, mode);
01327 }
01334 bool MPU6050::getInterruptDrive() {
01335     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_INT_OPEN_BIT, buffer);
01336     return buffer[0];
01337 }
01344 void MPU6050::setInterruptDrive(bool drive) {
01345     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_INT_OPEN_BIT, drive);
01346 }
01353 bool MPU6050::getInterruptLatch() {
01354     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_LATCH_INT_EN_BIT, buffer);
01355     return buffer[0];
01356 }
01363 void MPU6050::setInterruptLatch(bool latch) {
01364     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_LATCH_INT_EN_BIT, latch);
01365 }
01372 bool MPU6050::getInterruptLatchClear() {
01373     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_INT_RD_CLEAR_BIT, buffer);
01374     return buffer[0];
01375 }
01382 void MPU6050::setInterruptLatchClear(bool clear) {
01383     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_INT_RD_CLEAR_BIT, clear);
01384 }
01391 bool MPU6050::getFSyncInterruptLevel() {
01392     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT, buffer);
01393     return buffer[0];
01394 }
01401 void MPU6050::setFSyncInterruptLevel(bool level) {
01402     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT, level);
01403 }
01410 bool MPU6050::getFSyncInterruptEnabled() {
01411     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_FSYNC_INT_EN_BIT, buffer);
01412     return buffer[0];
01413 }
01420 void MPU6050::setFSyncInterruptEnabled(bool enabled) {
01421     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_FSYNC_INT_EN_BIT, enabled);
01422 }
01434 bool MPU6050::getI2CBypassEnabled() {
01435     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_I2C_BYPASS_EN_BIT, buffer);
01436     return buffer[0];
01437 }
01449 void MPU6050::setI2CBypassEnabled(bool enabled) {
01450     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_I2C_BYPASS_EN_BIT, enabled);
01451 }
01461 bool MPU6050::getClockOutputEnabled() {
01462     I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_CLKOUT_EN_BIT, buffer);
01463     return buffer[0];
01464 }
```

```
01474 void MPU6050::setClockOutputEnabled(bool enabled) {
01475     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
      MPU6050_INTCFG_CLKOUT_EN_BIT, enabled);
01476 }
01477
01478 // INT_ENABLE register
01479
01487 uint8_t MPU6050::getIntEnabled() {
01488     I2Cdev::readByte(devAddr, MPU6050_RA_INT_ENABLE,
      buffer);
01489     return buffer[0];
01490 }
01499 void MPU6050::setIntEnabled(uint8_t enabled) {
01500     I2Cdev::writeByte(devAddr, MPU6050_RA_INT_ENABLE, enabled);
01501 }
01508 bool MPU6050::getIntFreefallEnabled() {
01509     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_FF_BIT, buffer);
01510     return buffer[0];
01511 }
01518 void MPU6050::setIntFreefallEnabled(bool enabled) {
01519     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_FF_BIT, enabled);
01520 }
01527 bool MPU6050::getIntMotionEnabled() {
01528     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_MOT_BIT, buffer);
01529     return buffer[0];
01530 }
01537 void MPU6050::setIntMotionEnabled(bool enabled) {
01538     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_MOT_BIT, enabled);
01539 }
01546 bool MPU6050::getIntZeroMotionEnabled() {
01547     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_ZMOT_BIT, buffer);
01548     return buffer[0];
01549 }
01556 void MPU6050::setIntZeroMotionEnabled(bool enabled) {
01557     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_ZMOT_BIT, enabled);
01558 }
01565 bool MPU6050::getIntFIFOBufferOverflowEnabled() {
01566     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_FIFO_OFLOW_BIT, buffer);
01567     return buffer[0];
01568 }
01575 void MPU6050::setIntFIFOBufferOverflowEnabled(bool enabled) {
01576     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_FIFO_OFLOW_BIT, enabled);
01577 }
01585 bool MPU6050::getIntI2CMasterEnabled() {
01586     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_I2C_MST_INT_BIT, buffer);
01587     return buffer[0];
01588 }
01595 void MPU6050::setIntI2CMasterEnabled(bool enabled) {
01596     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_I2C_MST_INT_BIT, enabled);
01597 }
01605 bool MPU6050::getIntDataReadyEnabled() {
01606     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_DATA_RDY_BIT, buffer);
01607     return buffer[0];
01608 }
01615 void MPU6050::setIntDataReadyEnabled(bool enabled) {
01616     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_DATA_RDY_BIT, enabled);
01617 }
01618
01619 // INT_STATUS register
01620
01628 uint8_t MPU6050::getIntStatus() {
01629     I2Cdev::readByte(devAddr, MPU6050_RA_INT_STATUS,
      buffer);
01630     return buffer[0];
01631 }
01639 bool MPU6050::getIntFreefallStatus() {
01640     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_FF_BIT, buffer);
01641     return buffer[0];
01642 }
01650 bool MPU6050::getIntMotionStatus() {
01651     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_MOT_BIT, buffer);
01652     return buffer[0];
01653 }
```

```
01661 bool MPU6050::getIntZeroMotionStatus() {
01662     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_ZMOT_BIT, buffer);
01663     return buffer[0];
01664 }
01672 bool MPU6050::getIntFIFOBufferOverflowStatus() {
01673     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_FIFO_OFLOW_BIT, buffer);
01674     return buffer[0];
01675 }
01684 bool MPU6050::getIntI2CMasterStatus() {
01685     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_I2C_MST_INT_BIT, buffer);
01686     return buffer[0];
01687 }
01695 bool MPU6050::getIntDataReadyStatus() {
01696     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_DATA_RDY_BIT, buffer);
01697     return buffer[0];
01698 }
01699
01700 // ACCEL_*OUT_* registers
01701
01718 void MPU6050::getMotion9(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy
      , int16_t* gz, int16_t* mx, int16_t* my, int16_t* mz) {
01719     getMotion6(ax, ay, az, gx, gy, gz);
01720     // TODO: magnetometer integration
01721 }
01734 void MPU6050::getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy
      , int16_t* gz) {
01735     I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14,
      buffer);
01736     *ax = (((int16_t)buffer[0]) << 8) | buffer[1];
01737     *ay = (((int16_t)buffer[2]) << 8) | buffer[3];
01738     *az = (((int16_t)buffer[4]) << 8) | buffer[5];
01739     *gx = (((int16_t)buffer[8]) << 8) | buffer[9];
01740     *gy = (((int16_t)buffer[10]) << 8) | buffer[11];
01741     *gz = (((int16_t)buffer[12]) << 8) | buffer[13];
01742 }
01779 void MPU6050::getAcceleration(int16_t* x, int16_t* y, int16_t* z) {
01780     I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 6,
      buffer);
01781     *x = (((int16_t)buffer[0]) << 8) | buffer[1];
01782     *y = (((int16_t)buffer[2]) << 8) | buffer[3];
01783     *z = (((int16_t)buffer[4]) << 8) | buffer[5];
01784 }
01790 int16_t MPU6050::getAccelerationX() {
01791     I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 2,
      buffer);
01792     return (((int16_t)buffer[0]) << 8) | buffer[1];
01793 }
01799 int16_t MPU6050::getAccelerationY() {
01800     I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_YOUT_H, 2,
      buffer);
01801     return (((int16_t)buffer[0]) << 8) | buffer[1];
01802 }
01808 int16_t MPU6050::getAccelerationZ() {
01809     I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_ZOUT_H, 2,
      buffer);
01810     return (((int16_t)buffer[0]) << 8) | buffer[1];
01811 }
01812
01813 // TEMP_OUT_* registers
01814
01819 int16_t MPU6050::getTemperature() {
01820     I2Cdev::readBytes(devAddr, MPU6050_RA_TEMP_OUT_H, 2,
      buffer);
01821     return (((int16_t)buffer[0]) << 8) | buffer[1];
01822 }
01823
01824 // GYRO_*OUT_* registers
01825
01858 void MPU6050::getRotation(int16_t* x, int16_t* y, int16_t* z) {
01859     I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 6,
      buffer);
01860     *x = (((int16_t)buffer[0]) << 8) | buffer[1];
01861     *y = (((int16_t)buffer[2]) << 8) | buffer[3];
01862     *z = (((int16_t)buffer[4]) << 8) | buffer[5];
01863 }
01864
01865 void MPU6050::getRotationXY(int16_t* x, int16_t* y) {
01866     I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 4,
      buffer);
01867     *x = (((int16_t)buffer[0]) << 8) | buffer[1];
01868     *y = (((int16_t)buffer[2]) << 8) | buffer[3];
01869 }
01870
```

```
01876 int16_t MPU6050::getRotationX() {
01877     I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 2,
      buffer);
01878     return (((int16_t)buffer[0]) << 8) | buffer[1];
01879 }
01885 int16_t MPU6050::getRotationY() {
01886     I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_YOUT_H, 2,
      buffer);
01887     return (((int16_t)buffer[0]) << 8) | buffer[1];
01888 }
01894 int16_t MPU6050::getRotationZ() {
01895     I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_ZOUT_H, 2,
      buffer);
01896     return (((int16_t)buffer[0]) << 8) | buffer[1];
01897 }
01898
01899 // EXT_SENS_DATA_* registers
01900
01975 uint8_t MPU6050::getExternalSensorByte(int position) {
01976     I2Cdev::readByte(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position,
      buffer);
01977     return buffer[0];
01978 }
01984 uint16_t MPU6050::getExternalSensorWord(int position) {
01985     I2Cdev::readBytes(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, 2,
       buffer);
01986     return (((uint16_t)buffer[0]) << 8) | buffer[1];
01987 }
01993 uint32_t MPU6050::getExternalSensorDWord(int position) {
01994     I2Cdev::readBytes(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, 4,
       buffer);
01995     return (((uint32_t)buffer[0]) << 24) | (((uint32_t)buffer[1]) << 16) | (((uint16_t)buffer[2]) <<
      8) | buffer[3];
01996 }
01997
01998 // MOT_DETECT_STATUS register
01999
02005 bool MPU6050::getXNegMotionDetected() {
02006     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_XNEG_BIT, buffer);
02007     return buffer[0];
02008 }
02014 bool MPU6050::getXPosMotionDetected() {
02015     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_XPOS_BIT, buffer);
02016     return buffer[0];
02017 }
02023 bool MPU6050::getYNegMotionDetected() {
02024     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_YNEG_BIT, buffer);
02025     return buffer[0];
02026 }
02032 bool MPU6050::getYPosMotionDetected() {
02033     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_YPOS_BIT, buffer);
02034     return buffer[0];
02035 }
02041 bool MPU6050::getZNegMotionDetected() {
02042     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_ZNEG_BIT, buffer);
02043     return buffer[0];
02044 }
02050 bool MPU6050::getZPosMotionDetected() {
02051     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_ZPOS_BIT, buffer);
02052     return buffer[0];
02053 }
02059 bool MPU6050::getZeroMotionDetected() {
02060     I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
      MPU6050_MOTION_MOT_ZRMOT_BIT, buffer);
02061     return buffer[0];
02062 }
02063
02064 // I2C_SLV*_DO register
02065
02074 void MPU6050::setSlaveOutputByte(uint8_t num, uint8_t data) {
02075     if (num > 3) return;
02076     I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_DO + num, data);
02077 }
02078
02079 // I2C_MST_DELAY_CTRL register
02080
02089 bool MPU6050::getExternalShadowDelayEnabled() {
02090     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL,
      MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT,
      buffer);
02091     return buffer[0];
```

```
02092 }
02099 void MPU6050::setExternalShadowDelayEnabled(bool enabled) {
02100     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL,
      MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT, enabled);
02101 }
02120 bool MPU6050::getSlaveDelayEnabled(uint8_t num) {
02121     // MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT is 4, SLV3 is 3, etc.
02122     if (num > 4) return 0;
02123     I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, num,
      buffer);
02124     return buffer[0];
02125 }
02132 void MPU6050::setSlaveDelayEnabled(uint8_t num, bool enabled) {
02133     I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, num,
      enabled);
02134 }
02135
02136 // SIGNAL_PATH_RESET register
02137
02144 void MPU6050::resetGyroscopePath() {
02145     I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET,
      MPU6050_PATHRESET_GYRO_RESET_BIT, true);
02146 }
02153 void MPU6050::resetAccelerometerPath() {
02154     I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET,
      MPU6050_PATHRESET_ACCEL_RESET_BIT, true);
02155 }
02162 void MPU6050::resetTemperaturePath() {
02163     I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET,
      MPU6050_PATHRESET_TEMP_RESET_BIT, true);
02164 }
02165
02166 // MOT_DETECT_CTRL register
02167
02182 uint8_t MPU6050::getAccelerometerPowerOnDelay() {
02183     I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
      MPU6050_DETECT_ACCEL_ON_DELAY_BIT,
      MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH, buffer);
02184     return buffer[0];
02185 }
02192 void MPU6050::setAccelerometerPowerOnDelay(uint8_t delay) {
02193     I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
      MPU6050_DETECT_ACCEL_ON_DELAY_BIT,
      MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH, delay);
02194 }
02221 uint8_t MPU6050::getFreefallDetectionCounterDecrement() {
02222     I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
      MPU6050_DETECT_FF_COUNT_BIT,
      MPU6050_DETECT_FF_COUNT_LENGTH, buffer);
02223     return buffer[0];
02224 }
02231 void MPU6050::setFreefallDetectionCounterDecrement(uint8_t
      decrement) {
02232     I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
      MPU6050_DETECT_FF_COUNT_BIT,
      MPU6050_DETECT_FF_COUNT_LENGTH, decrement);
02233 }
02257 uint8_t MPU6050::getMotionDetectionCounterDecrement() {
02258     I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
      MPU6050_DETECT_MOT_COUNT_BIT,
      MPU6050_DETECT_MOT_COUNT_LENGTH, buffer);
02259     return buffer[0];
02260 }
02267 void MPU6050::setMotionDetectionCounterDecrement(uint8_t
      decrement) {
02268     I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
      MPU6050_DETECT_MOT_COUNT_BIT,
      MPU6050_DETECT_MOT_COUNT_LENGTH, decrement);
02269 }
02270
02271 // USER_CTRL register
02272
02281 bool MPU6050::getFIFOEnabled() {
02282     I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_FIFO_EN_BIT, buffer);
02283     return buffer[0];
02284 }
02291 void MPU6050::setFIFOEnabled(bool enabled) {
02292     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_FIFO_EN_BIT, enabled);
02293 }
02305 bool MPU6050::getI2CMasterModeEnabled() {
02306     I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_I2C_MST_EN_BIT, buffer);
02307     return buffer[0];
02308 }
02315 void MPU6050::setI2CMasterModeEnabled(bool enabled) {
```

```
02316     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_I2C_MST_EN_BIT, enabled);
02317 }
02322 void MPU6050::switchSPIEnabled(bool enabled) {
02323     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_I2C_IF_DIS_BIT, enabled);
02324 }
02331 void MPU6050::resetFIFO() {
02332     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_FIFO_RESET_BIT, true);
02333 }
02340 void MPU6050::resetI2CMaster() {
02341     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_I2C_MST_RESET_BIT, true);
02342 }
02355 void MPU6050::resetSensors() {
02356     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_SIG_COND_RESET_BIT, true);
02357 }
02358
02359 // PWR_MGMT_1 register
02360
02366 void MPU6050::reset() {
02367     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_DEVICE_RESET_BIT, true);
02368 }
02380 bool MPU6050::getSleepEnabled() {
02381     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_SLEEP_BIT, buffer);
02382     return buffer[0];
02383 }
02390 void MPU6050::setSleepEnabled(bool enabled) {
02391     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_SLEEP_BIT, enabled);
02392 }
02401 bool MPU6050::getWakeCycleEnabled() {
02402     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_CYCLE_BIT, buffer);
02403     return buffer[0];
02404 }
02411 void MPU6050::setWakeCycleEnabled(bool enabled) {
02412     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_CYCLE_BIT, enabled);
02413 }
02425 bool MPU6050::getTempSensorEnabled() {
02426     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_TEMP_DIS_BIT, buffer);
02427     return buffer[0] == 0; // 1 is actually disabled here
02428 }
02439 void MPU6050::setTempSensorEnabled(bool enabled) {
02440     // 1 is actually disabled here
02441     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_TEMP_DIS_BIT, !enabled);
02442 }
02449 uint8_t MPU6050::getClockSource() {
02450     I2Cdev::readBits(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_CLKSEL_BIT, MPU6050_PWR1_CLKSEL_LENGTH,
      buffer);
02451     return buffer[0];
02452 }
02483 void MPU6050::setClockSource(uint8_t source) {
02484     I2Cdev::writeBits(devAddr, MPU6050_RA_PWR_MGMT_1,
      MPU6050_PWR1_CLKSEL_BIT, MPU6050_PWR1_CLKSEL_LENGTH,
      source);
02485 }
02486
02487 // PWR_MGMT_2 register
02488
02512 uint8_t MPU6050::getWakeFrequency() {
02513     I2Cdev::readBits(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_LP_WAKE_CTRL_BIT,
      MPU6050_PWR2_LP_WAKE_CTRL_LENGTH, buffer);
02514     return buffer[0];
02515 }
02520 void MPU6050::setWakeFrequency(uint8_t frequency) {
02521     I2Cdev::writeBits(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_LP_WAKE_CTRL_BIT,
      MPU6050_PWR2_LP_WAKE_CTRL_LENGTH, frequency);
02522 }
02523
02530 bool MPU6050::getStandbyXAccelEnabled() {
02531     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_XA_BIT, buffer);
02532     return buffer[0];
02533 }
02540 void MPU6050::setStandbyXAccelEnabled(bool enabled) {
02541     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
```

```
      MPU6050_PWR2_STBY_XA_BIT, enabled);
02542 }
02549 bool MPU6050::getStandbyYAccelEnabled() {
02550     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_YA_BIT, buffer);
02551     return buffer[0];
02552 }
02559 void MPU6050::setStandbyYAccelEnabled(bool enabled) {
02560     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_YA_BIT, enabled);
02561 }
02568 bool MPU6050::getStandbyZAccelEnabled() {
02569     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_ZA_BIT, buffer);
02570     return buffer[0];
02571 }
02578 void MPU6050::setStandbyZAccelEnabled(bool enabled) {
02579     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_ZA_BIT, enabled);
02580 }
02587 bool MPU6050::getStandbyXGyroEnabled() {
02588     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_XG_BIT, buffer);
02589     return buffer[0];
02590 }
02597 void MPU6050::setStandbyXGyroEnabled(bool enabled) {
02598     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_XG_BIT, enabled);
02599 }
02606 bool MPU6050::getStandbyYGyroEnabled() {
02607     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_YG_BIT, buffer);
02608     return buffer[0];
02609 }
02616 void MPU6050::setStandbyYGyroEnabled(bool enabled) {
02617     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_YG_BIT, enabled);
02618 }
02625 bool MPU6050::getStandbyZGyroEnabled() {
02626     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_ZG_BIT, buffer);
02627     return buffer[0];
02628 }
02635 void MPU6050::setStandbyZGyroEnabled(bool enabled) {
02636     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
      MPU6050_PWR2_STBY_ZG_BIT, enabled);
02637 }
02638
02639 // FIFO_COUNT* registers
02640
02648 uint16_t MPU6050::getFIFOCount() {
02649     I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_COUNTH, 2,
      buffer);
02650     return (((uint16_t)buffer[0]) << 8) | buffer[1];
02651 }
02652
02653 // FIFO_R_W register
02654
02680 uint8_t MPU6050::getFIFOByte() {
02681     I2Cdev::readByte(devAddr, MPU6050_RA_FIFO_R_W,
      buffer);
02682     return buffer[0];
02683 }
02684 void MPU6050::getFIFOBytes(uint8_t *data, uint8_t length) {
02685     I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_R_W, length, data);
02686 }
02691 void MPU6050::setFIFOByte(uint8_t data) {
02692     I2Cdev::writeByte(devAddr, MPU6050_RA_FIFO_R_W, data);
02693 }
02694
02695 // WHO_AM_I register
02696
02704 uint8_t MPU6050::getDeviceID() {
02705     I2Cdev::readBits(devAddr, MPU6050_RA_WHO_AM_I,
      MPU6050_WHO_AM_I_BIT, MPU6050_WHO_AM_I_LENGTH,
      buffer);
02706     return buffer[0];
02707 }
02717 void MPU6050::setDeviceID(uint8_t id) {
02718     I2Cdev::writeBits(devAddr, MPU6050_RA_WHO_AM_I,
      MPU6050_WHO_AM_I_BIT, MPU6050_WHO_AM_I_LENGTH, id);
02719 }
02720
02721 // ======== UNDOCUMENTED/DMP REGISTERS/METHODS ========
02722
02723 // XG_OFFS_TC register
02724
```

```
02725 uint8_t MPU6050::getOTPBankValid() {
02726     I2Cdev::readBit(devAddr, MPU6050_RA_XG_OFFS_TC,
      MPU6050_TC_OTP_BNK_VLD_BIT, buffer);
02727     return buffer[0];
02728 }
02729 void MPU6050::setOTPBankValid(bool enabled) {
02730     I2Cdev::writeBit(devAddr, MPU6050_RA_XG_OFFS_TC,
      MPU6050_TC_OTP_BNK_VLD_BIT, enabled);
02731 }
02732 int8_t MPU6050::getXGyroOffset() {
02733     I2Cdev::readBits(devAddr, MPU6050_RA_XG_OFFS_TC,
      MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH,
      buffer);
02734     return buffer[0];
02735 }
02736 void MPU6050::setXGyroOffset(int8_t offset) {
02737     I2Cdev::writeBits(devAddr, MPU6050_RA_XG_OFFS_TC,
      MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
02738 }
02739
02740 // YG_OFFS_TC register
02741
02742 int8_t MPU6050::getYGyroOffset() {
02743     I2Cdev::readBits(devAddr, MPU6050_RA_YG_OFFS_TC,
      MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH,
      buffer);
02744     return buffer[0];
02745 }
02746 void MPU6050::setYGyroOffset(int8_t offset) {
02747     I2Cdev::writeBits(devAddr, MPU6050_RA_YG_OFFS_TC,
      MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
02748 }
02749
02750 // ZG_OFFS_TC register
02751
02752 int8_t MPU6050::getZGyroOffset() {
02753     I2Cdev::readBits(devAddr, MPU6050_RA_ZG_OFFS_TC,
      MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH,
      buffer);
02754     return buffer[0];
02755 }
02756 void MPU6050::setZGyroOffset(int8_t offset) {
02757     I2Cdev::writeBits(devAddr, MPU6050_RA_ZG_OFFS_TC,
      MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
02758 }
02759
02760 // X_FINE_GAIN register
02761
02762 int8_t MPU6050::getXFineGain() {
02763     I2Cdev::readByte(devAddr, MPU6050_RA_X_FINE_GAIN,
      buffer);
02764     return buffer[0];
02765 }
02766 void MPU6050::setXFineGain(int8_t gain) {
02767     I2Cdev::writeByte(devAddr, MPU6050_RA_X_FINE_GAIN, gain);
02768 }
02769
02770 // Y_FINE_GAIN register
02771
02772 int8_t MPU6050::getYFineGain() {
02773     I2Cdev::readByte(devAddr, MPU6050_RA_Y_FINE_GAIN,
      buffer);
02774     return buffer[0];
02775 }
02776 void MPU6050::setYFineGain(int8_t gain) {
02777     I2Cdev::writeByte(devAddr, MPU6050_RA_Y_FINE_GAIN, gain);
02778 }
02779
02780 // Z_FINE_GAIN register
02781
02782 int8_t MPU6050::getZFineGain() {
02783     I2Cdev::readByte(devAddr, MPU6050_RA_Z_FINE_GAIN,
      buffer);
02784     return buffer[0];
02785 }
02786 void MPU6050::setZFineGain(int8_t gain) {
02787     I2Cdev::writeByte(devAddr, MPU6050_RA_Z_FINE_GAIN, gain);
02788 }
02789
02790 // XA_OFFS_* registers
02791
02792 int16_t MPU6050::getXAccelOffset() {
02793     I2Cdev::readBytes(devAddr, MPU6050_RA_XA_OFFS_H, 2,
      buffer);
02794     return (((int16_t)buffer[0]) << 8) | buffer[1];
02795 }
02796 void MPU6050::setXAccelOffset(int16_t offset) {
```

```
02797     I2Cdev::writeWord(devAddr, MPU6050_RA_XA_OFFS_H, offset);
02798 }
02799
02800 // YA_OFFS_* register
02801
02802 int16_t MPU6050::getYAccelOffset() {
02803     I2Cdev::readBytes(devAddr, MPU6050_RA_YA_OFFS_H, 2,
      buffer);
02804     return (((int16_t)buffer[0]) << 8) | buffer[1];
02805 }
02806 void MPU6050::setYAccelOffset(int16_t offset) {
02807     I2Cdev::writeWord(devAddr, MPU6050_RA_YA_OFFS_H, offset);
02808 }
02809
02810 // ZA_OFFS_* register
02811
02812 int16_t MPU6050::getZAccelOffset() {
02813     I2Cdev::readBytes(devAddr, MPU6050_RA_ZA_OFFS_H, 2,
      buffer);
02814     return (((int16_t)buffer[0]) << 8) | buffer[1];
02815 }
02816 void MPU6050::setZAccelOffset(int16_t offset) {
02817     I2Cdev::writeWord(devAddr, MPU6050_RA_ZA_OFFS_H, offset);
02818 }
02819
02820 // XG_OFFS_USR* registers
02821
02822 int16_t MPU6050::getXGyroOffsetUser() {
02823     I2Cdev::readBytes(devAddr, MPU6050_RA_XG_OFFS_USRH, 2,
      buffer);
02824     return (((int16_t)buffer[0]) << 8) | buffer[1];
02825 }
02826 void MPU6050::setXGyroOffsetUser(int16_t offset) {
02827     I2Cdev::writeWord(devAddr, MPU6050_RA_XG_OFFS_USRH, offset);
02828 }
02829
02830 // YG_OFFS_USR* register
02831
02832 int16_t MPU6050::getYGyroOffsetUser() {
02833     I2Cdev::readBytes(devAddr, MPU6050_RA_YG_OFFS_USRH, 2,
      buffer);
02834     return (((int16_t)buffer[0]) << 8) | buffer[1];
02835 }
02836 void MPU6050::setYGyroOffsetUser(int16_t offset) {
02837     I2Cdev::writeWord(devAddr, MPU6050_RA_YG_OFFS_USRH, offset);
02838 }
02839
02840 // ZG_OFFS_USR* register
02841
02842 int16_t MPU6050::getZGyroOffsetUser() {
02843     I2Cdev::readBytes(devAddr, MPU6050_RA_ZG_OFFS_USRH, 2,
      buffer);
02844     return (((int16_t)buffer[0]) << 8) | buffer[1];
02845 }
02846 void MPU6050::setZGyroOffsetUser(int16_t offset) {
02847     I2Cdev::writeWord(devAddr, MPU6050_RA_ZG_OFFS_USRH, offset);
02848 }
02849
02850 // INT_ENABLE register (DMP functions)
02851
02852 bool MPU6050::getIntPLLReadyEnabled() {
02853     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
02854     return buffer[0];
02855 }
02856 void MPU6050::setIntPLLReadyEnabled(bool enabled) {
02857     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_PLL_RDY_INT_BIT, enabled);
02858 }
02859 bool MPU6050::getIntDMPEnabled() {
02860     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
02861     return buffer[0];
02862 }
02863 void MPU6050::setIntDMPEnabled(bool enabled) {
02864     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
      MPU6050_INTERRUPT_DMP_INT_BIT, enabled);
02865 }
02866
02867 // DMP_INT_STATUS
02868
02869 bool MPU6050::getDMPInt5Status() {
02870     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
      MPU6050_DMPINT_5_BIT, buffer);
02871     return buffer[0];
02872 }
02873 bool MPU6050::getDMPInt4Status() {
```

```
02874     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
      MPU6050_DMPINT_4_BIT, buffer);
02875     return buffer[0];
02876 }
02877 bool MPU6050::getDMPInt3Status() {
02878     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
      MPU6050_DMPINT_3_BIT, buffer);
02879     return buffer[0];
02880 }
02881 bool MPU6050::getDMPInt2Status() {
02882     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
      MPU6050_DMPINT_2_BIT, buffer);
02883     return buffer[0];
02884 }
02885 bool MPU6050::getDMPInt1Status() {
02886     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
      MPU6050_DMPINT_1_BIT, buffer);
02887     return buffer[0];
02888 }
02889 bool MPU6050::getDMPInt0Status() {
02890     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
      MPU6050_DMPINT_0_BIT, buffer);
02891     return buffer[0];
02892 }
02893
02894 // INT_STATUS register (DMP functions)
02895
02896 bool MPU6050::getIntPLLReadyStatus() {
02897     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
02898     return buffer[0];
02899 }
02900 bool MPU6050::getIntDMPStatus() {
02901     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
      MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
02902     return buffer[0];
02903 }
02904
02905 // USER_CTRL register (DMP functions)
02906
02907 bool MPU6050::getDMPEnabled() {
02908     I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_DMP_EN_BIT, buffer);
02909     return buffer[0];
02910 }
02911 void MPU6050::setDMPEnabled(bool enabled) {
02912     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_DMP_EN_BIT, enabled);
02913 }
02914 void MPU6050::resetDMP() {
02915 //    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_DMP_RESET_BIT, true);
02916     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
      MPU6050_USERCTRL_DMP_RESET_BIT, true);
02917     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, 0x00, true);
02918     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, 0x80 | 0x40 | 0x08, true);
02919 }
02920
02921 // BANK_SEL register
02922
02923 void MPU6050::setMemoryBank(uint8_t bank, bool prefetchEnabled, bool userBank) {
02924     bank &= 0x1F;
02925     if (userBank) bank |= 0x20;
02926     if (prefetchEnabled) bank |= 0x40;
02927     I2Cdev::writeByte(devAddr, MPU6050_RA_BANK_SEL, bank);
02928 }
02929
02930 // MEM_START_ADDR register
02931
02932 void MPU6050::setMemoryStartAddress(uint8_t address) {
02933     I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_START_ADDR, address);
02934 }
02935
02936 // MEM_R_W register
02937
02938 uint8_t MPU6050::readMemoryByte() {
02939     I2Cdev::readByte(devAddr, MPU6050_RA_MEM_R_W,
      buffer);
02940     return buffer[0];
02941 }
02942 void MPU6050::writeMemoryByte(uint8_t data) {
02943     I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_R_W, data);
02944 }
02945 void MPU6050::readMemoryBlock(uint8_t *data, uint16_t dataSize, uint8_t bank,
      uint8_t address) {
02946     setMemoryBank(bank);
02947     setMemoryStartAddress(address);
02948     uint8_t chunkSize;
```

```
02949     for (uint16_t i = 0; i < dataSize;) {
02950         // determine correct chunk size according to bank position and data size
02951         chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;
02952 
02953         // make sure we don't go past the data size
02954         if (i + chunkSize > dataSize) chunkSize = dataSize - i;
02955 
02956         // make sure this chunk doesn't go past the bank boundary (256 bytes)
02957         if (chunkSize > 256 - address) chunkSize = 256 - address;
02958 
02959         // read the chunk of data as specified
02960         I2Cdev::readBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, data + i);
02961 
02962         // increase byte index by [chunkSize]
02963         i += chunkSize;
02964 
02965         // uint8_t automatically wraps to 0 at 256
02966         address += chunkSize;
02967 
02968         // if we aren't done, update bank (if necessary) and address
02969         if (i < dataSize) {
02970             if (address == 0) bank++;
02971             setMemoryBank(bank);
02972             setMemoryStartAddress(address);
02973         }
02974     }
02975 }
02976 bool MPU6050::writeMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t
    bank, uint8_t address, bool verify, bool useProgMem) {
02977     setMemoryBank(bank);
02978     setMemoryStartAddress(address);
02979     uint8_t chunkSize;
02980     uint8_t *verifyBuffer;
02981     uint8_t *progBuffer;
02982     uint16_t i;
02983     uint8_t j;
02984     if (verify) verifyBuffer = (uint8_t *)malloc(MPU6050_DMP_MEMORY_CHUNK_SIZE
    );
02985     if (useProgMem) progBuffer = (uint8_t *)malloc(MPU6050_DMP_MEMORY_CHUNK_SIZE
    );
02986     for (i = 0; i < dataSize;) {
02987         // determine correct chunk size according to bank position and data size
02988         chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;
02989 
02990         // make sure we don't go past the data size
02991         if (i + chunkSize > dataSize) chunkSize = dataSize - i;
02992 
02993         // make sure this chunk doesn't go past the bank boundary (256 bytes)
02994         if (chunkSize > 256 - address) chunkSize = 256 - address;
02995 
02996         if (useProgMem) {
02997             // write the chunk of data as specified
02998             for (j = 0; j < chunkSize; j++) progBuffer[j] = pgm_read_byte(data + i + j);
02999         } else {
03000             // write the chunk of data as specified
03001             progBuffer = (uint8_t *)data + i;
03002         }
03003 
03004         I2Cdev::writeBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, progBuffer);
03005 
03006         // verify data if needed
03007         if (verify && verifyBuffer) {
03008             setMemoryBank(bank);
03009             setMemoryStartAddress(address);
03010             I2Cdev::readBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, verifyBuffer
    );
03011             if (memcmp(progBuffer, verifyBuffer, chunkSize) != 0) {
03012                 /*Serial.print("Block write verification error, bank ");
03013                 Serial.print(bank, DEC);
03014                 Serial.print(", address ");
03015                 Serial.print(address, DEC);
03016                 Serial.print("!\nExpected:");
03017                 for (j = 0; j < chunkSize; j++) {
03018                     Serial.print(" 0x");
03019                     if (progBuffer[j] < 16) Serial.print("0");
03020                     Serial.print(progBuffer[j], HEX);
03021                 }
03022                 Serial.print("\nReceived:");
03023                 for (uint8_t j = 0; j < chunkSize; j++) {
03024                     Serial.print(" 0x");
03025                     if (verifyBuffer[i + j] < 16) Serial.print("0");
03026                     Serial.print(verifyBuffer[i + j], HEX);
03027                 }
03028                 Serial.print("\n");*/
03029                 free(verifyBuffer);
03030                 if (useProgMem) free(progBuffer);
03031                 return false; // uh oh.
```

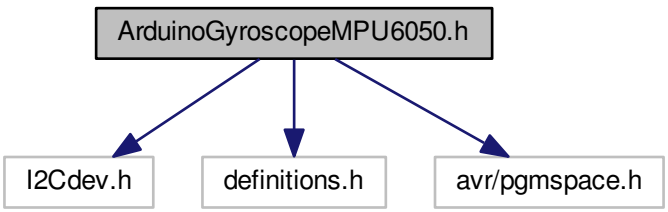
```
03032             }
03033         }
03034
03035         // increase byte index by [chunkSize]
03036         i += chunkSize;
03037
03038         // uint8_t automatically wraps to 0 at 256
03039         address += chunkSize;
03040
03041         // if we aren't done, update bank (if necessary) and address
03042         if (i < dataSize) {
03043             if (address == 0) bank++;
03044             setMemoryBank(bank);
03045             setMemoryStartAddress(address);
03046         }
03047     }
03048     if (verify) free(verifyBuffer);
03049     if (useProgMem) free(progBuffer);
03050     return true;
03051 }
03052 bool MPU6050::writeProgMemoryBlock(const uint8_t *data, uint16_t dataSize,
      uint8_t bank, uint8_t address, bool verify) {
03053     return writeMemoryBlock(data, dataSize, bank, address, verify, true);
03054 }
03055 bool MPU6050::writeDMPConfigurationSet(const uint8_t *data, uint16_t
      dataSize, bool useProgMem) {
03056     uint8_t *progBuffer, success, special;
03057     uint16_t i, j;
03058     if (useProgMem) {
03059         progBuffer = (uint8_t *)malloc(8); // assume 8-byte blocks, realloc later if necessary
03060     }
03061
03062     // config set data is a long string of blocks with the following structure:
03063     // [bank] [offset] [length] [byte[0], byte[1], ..., byte[length]]
03064     uint8_t bank, offset, length;
03065     for (i = 0; i < dataSize;) {
03066         if (useProgMem) {
03067             bank = pgm_read_byte(data + i++);
03068             offset = pgm_read_byte(data + i++);
03069             length = pgm_read_byte(data + i++);
03070         } else {
03071             bank = data[i++];
03072             offset = data[i++];
03073             length = data[i++];
03074         }
03075
03076         // write data or perform special action
03077         if (length > 0) {
03078             // regular block of data to write
03079             /*Serial.print("Writing config block to bank ");
03080             Serial.print(bank);
03081             Serial.print(", offset ");
03082             Serial.print(offset);
03083             Serial.print(", length=");
03084             Serial.println(length);*/
03085             if (useProgMem) {
03086                 if (sizeof(progBuffer) < length) progBuffer = (uint8_t *)realloc(progBuffer, length);
03087                 for (j = 0; j < length; j++) progBuffer[j] = pgm_read_byte(data + i + j);
03088             } else {
03089                 progBuffer = (uint8_t *)data + i;
03090             }
03091             success = writeMemoryBlock(progBuffer, length, bank, offset, true);
03092             i += length;
03093         } else {
03094             // special instruction
03095             // NOTE: this kind of behavior (what and when to do certain things)
03096             // is totally undocumented. This code is in here based on observed
03097             // behavior only, and exactly why (or even whether) it has to be here
03098             // is anybody's guess for now.
03099             if (useProgMem) {
03100                 special = pgm_read_byte(data + i++);
03101             } else {
03102                 special = data[i++];
03103             }
03104             /*Serial.print("Special command code ");
03105             Serial.print(special, HEX);
03106             Serial.println(" found...");*/
03107             if (special == 0x01) {
03108                 // enable DMP-related interrupts
03109
03110                 //setIntZeroMotionEnabled(true);
03111                 //setIntFIFOBufferOverflowEnabled(true);
03112                 //setIntDMPEnabled(true);
03113                 I2Cdev::writeByte(devAddr, MPU6050_RA_INT_ENABLE, 0x32);  //
       single operation
03114
03115                 success = true;
```

```
03116             } else {
03117                 // unknown special command
03118                 success = false;
03119             }
03120         }
03121
03122         if (!success) {
03123             if (useProgMem) free(progBuffer);
03124             return false; // uh oh
03125         }
03126     }
03127     if (useProgMem) free(progBuffer);
03128     return true;
03129 }
03130 bool MPU6050::writeProgDMPConfigurationSet(const uint8_t *data,
      uint16_t dataSize) {
03131     return writeDMPConfigurationSet(data, dataSize, true);
03132 }
03133
03134 // DMP_CFG_1 register
03135
03136 uint8_t MPU6050::getDMPConfig1() {
03137     I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_1,
      buffer);
03138     return buffer[0];
03139 }
03140 void MPU6050::setDMPConfig1(uint8_t config) {
03141     I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_1, config);
03142 }
03143
03144 // DMP_CFG_2 register
03145
03146 uint8_t MPU6050::getDMPConfig2() {
03147     I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_2,
      buffer);
03148     return buffer[0];
03149 }
03150 void MPU6050::setDMPConfig2(uint8_t config) {
03151     I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_2, config);
03152 }
03153
03154
03155
03156
```

4.7 ArduinoGyroscopeMPU6050.h File Reference

```
#include "I2Cdev.h"
#include "definitions.h"
#include <avr/pgmspace.h>
```

Include dependency graph for ArduinoGyroscopeMPU6050.h:



Classes

- class MPU6050

Macros

- #define MPU6050_RA_XG_OFFS_TC 0x00
- #define MPU6050_RA_YG_OFFS_TC 0x01
- #define MPU6050_RA_ZG_OFFS_TC 0x02
- #define MPU6050_RA_X_FINE_GAIN 0x03
- #define MPU6050_RA_Y_FINE_GAIN 0x04
- #define MPU6050_RA_Z_FINE_GAIN 0x05
- #define MPU6050_RA_XA_OFFS_H 0x06
- #define MPU6050_RA_XA_OFFS_L_TC 0x07
- #define MPU6050_RA_YA_OFFS_H 0x08
- #define MPU6050_RA_YA_OFFS_L_TC 0x09
- #define MPU6050_RA_ZA_OFFS_H 0x0A
- #define MPU6050_RA_ZA_OFFS_L_TC 0x0B
- #define MPU6050_RA_XG_OFFS_USRH 0x13
- #define MPU6050_RA_XG_OFFS_USRL 0x14
- #define MPU6050_RA_YG_OFFS_USRH 0x15
- #define MPU6050_RA_YG_OFFS_USRL 0x16
- #define MPU6050_RA_ZG_OFFS_USRH 0x17
- #define MPU6050_RA_ZG_OFFS_USRL 0x18
- #define MPU6050_RA_SMPLRT_DIV 0x19
- #define MPU6050_RA_CONFIG 0x1A
- #define MPU6050_RA_GYRO_CONFIG 0x1B
- #define MPU6050_RA_ACCEL_CONFIG 0x1C
- #define MPU6050_RA_FF_THR 0x1D
- #define MPU6050_RA_FF_DUR 0x1E
- #define MPU6050_RA_MOT_THR 0x1F
- #define MPU6050_RA_MOT_DUR 0x20
- #define MPU6050_RA_ZRMOT_THR 0x21
- #define MPU6050_RA_ZRMOT_DUR 0x22
- #define MPU6050_RA_FIFO_EN 0x23
- #define MPU6050_RA_I2C_MST_CTRL 0x24
- #define MPU6050_RA_I2C_SLV0_ADDR 0x25
- #define MPU6050_RA_I2C_SLV0_REG 0x26
- #define MPU6050_RA_I2C_SLV0_CTRL 0x27
- #define MPU6050_RA_I2C_SLV1_ADDR 0x28
- #define MPU6050_RA_I2C_SLV1_REG 0x29
- #define MPU6050_RA_I2C_SLV1_CTRL 0x2A
- #define MPU6050_RA_I2C_SLV2_ADDR 0x2B
- #define MPU6050_RA_I2C_SLV2_REG 0x2C
- #define MPU6050_RA_I2C_SLV2_CTRL 0x2D
- #define MPU6050_RA_I2C_SLV3_ADDR 0x2E
- #define MPU6050_RA_I2C_SLV3_REG 0x2F
- #define MPU6050_RA_I2C_SLV3_CTRL 0x30
- #define MPU6050_RA_I2C_SLV4_ADDR 0x31
- #define MPU6050_RA_I2C_SLV4_REG 0x32
- #define MPU6050_RA_I2C_SLV4_DO 0x33
- #define MPU6050_RA_I2C_SLV4_CTRL 0x34
- #define MPU6050_RA_I2C_SLV4_DI 0x35
- #define MPU6050_RA_I2C_MST_STATUS 0x36
- #define MPU6050_RA_INT_PIN_CFG 0x37
- #define MPU6050_RA_INT_ENABLE 0x38
- #define MPU6050_RA_DMP_INT_STATUS 0x39
- #define MPU6050_RA_INT_STATUS 0x3A
- #define MPU6050_RA_ACCEL_XOUT_H 0x3B

- #define MPU6050_RA_ACCEL_XOUT_L 0x3C
- #define MPU6050_RA_ACCEL_YOUT_H 0x3D
- #define MPU6050_RA_ACCEL_YOUT_L 0x3E
- #define MPU6050_RA_ACCEL_ZOUT_H 0x3F
- #define MPU6050_RA_ACCEL_ZOUT_L 0x40
- #define MPU6050_RA_TEMP_OUT_H 0x41
- #define MPU6050_RA_TEMP_OUT_L 0x42
- #define MPU6050_RA_GYRO_XOUT_H 0x43
- #define MPU6050_RA_GYRO_XOUT_L 0x44
- #define MPU6050_RA_GYRO_YOUT_H 0x45
- #define MPU6050_RA_GYRO_YOUT_L 0x46
- #define MPU6050_RA_GYRO_ZOUT_H 0x47
- #define MPU6050_RA_GYRO_ZOUT_L 0x48
- #define MPU6050_RA_EXT_SENS_DATA_00 0x49
- #define MPU6050_RA_EXT_SENS_DATA_01 0x4A
- #define MPU6050_RA_EXT_SENS_DATA_02 0x4B
- #define MPU6050_RA_EXT_SENS_DATA_03 0x4C
- #define MPU6050_RA_EXT_SENS_DATA_04 0x4D
- #define MPU6050_RA_EXT_SENS_DATA_05 0x4E
- #define MPU6050_RA_EXT_SENS_DATA_06 0x4F
- #define MPU6050_RA_EXT_SENS_DATA_07 0x50
- #define MPU6050_RA_EXT_SENS_DATA_08 0x51
- #define MPU6050_RA_EXT_SENS_DATA_09 0x52
- #define MPU6050_RA_EXT_SENS_DATA_10 0x53
- #define MPU6050_RA_EXT_SENS_DATA_11 0x54
- #define MPU6050_RA_EXT_SENS_DATA_12 0x55
- #define MPU6050_RA_EXT_SENS_DATA_13 0x56
- #define MPU6050_RA_EXT_SENS_DATA_14 0x57
- #define MPU6050_RA_EXT_SENS_DATA_15 0x58
- #define MPU6050_RA_EXT_SENS_DATA_16 0x59
- #define MPU6050_RA_EXT_SENS_DATA_17 0x5A
- #define MPU6050_RA_EXT_SENS_DATA_18 0x5B
- #define MPU6050_RA_EXT_SENS_DATA_19 0x5C
- #define MPU6050_RA_EXT_SENS_DATA_20 0x5D
- #define MPU6050_RA_EXT_SENS_DATA_21 0x5E
- #define MPU6050_RA_EXT_SENS_DATA_22 0x5F
- #define MPU6050_RA_EXT_SENS_DATA_23 0x60
- #define MPU6050_RA_MOT_DETECT_STATUS 0x61
- #define MPU6050_RA_I2C_SLV0_DO 0x63
- #define MPU6050_RA_I2C_SLV1_DO 0x64
- #define MPU6050_RA_I2C_SLV2_DO 0x65
- #define MPU6050_RA_I2C_SLV3_DO 0x66
- #define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67
- #define MPU6050_RA_SIGNAL_PATH_RESET 0x68
- #define MPU6050_RA_MOT_DETECT_CTRL 0x69
- #define MPU6050_RA_USER_CTRL 0x6A
- #define MPU6050_RA_PWR_MGMT_1 0x6B
- #define MPU6050_RA_PWR_MGMT_2 0x6C
- #define MPU6050_RA_BANK_SEL 0x6D
- #define MPU6050_RA_MEM_START_ADDR 0x6E
- #define MPU6050_RA_MEM_R_W 0x6F
- #define MPU6050_RA_DMP_CFG_1 0x70
- #define MPU6050_RA_DMP_CFG_2 0x71
- #define MPU6050_RA_FIFO_COUNTH 0x72
- #define MPU6050_RA_FIFO_COUNTL 0x73

- #define MPU6050_RA_FIFO_R_W 0x74
- #define MPU6050_RA_WHO_AM_I 0x75
- #define MPU6050_TC_PWR_MODE_BIT 7
- #define MPU6050_TC_OFFSET_BIT 6
- #define MPU6050_TC_OFFSET_LENGTH 6
- #define MPU6050_TC_OTP_BNK_VLD_BIT 0
- #define MPU6050_VDDIO_LEVEL_VLOGIC 0
- #define MPU6050_VDDIO_LEVEL_VDD 1
- #define MPU6050_CFG_EXT_SYNC_SET_BIT 5
- #define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
- #define MPU6050_CFG_DLPF_CFG_BIT 2
- #define MPU6050_CFG_DLPF_CFG_LENGTH 3
- #define MPU6050_EXT_SYNC_DISABLED 0x0
- #define MPU6050_EXT_SYNC_TEMP_OUT_L 0x1
- #define MPU6050_EXT_SYNC_GYRO_XOUT_L 0x2
- #define MPU6050_EXT_SYNC_GYRO_YOUT_L 0x3
- #define MPU6050_EXT_SYNC_GYRO_ZOUT_L 0x4
- #define MPU6050_EXT_SYNC_ACCEL_XOUT_L 0x5
- #define MPU6050_EXT_SYNC_ACCEL_YOUT_L 0x6
- #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L 0x7
- #define MPU6050_DLPF_BW_256 0x00
- #define MPU6050_DLPF_BW_188 0x01
- #define MPU6050_DLPF_BW_98 0x02
- #define MPU6050_DLPF_BW_42 0x03
- #define MPU6050_DLPF_BW_20 0x04
- #define MPU6050_DLPF_BW_10 0x05
- #define MPU6050_DLPF_BW_5 0x06
- #define MPU6050_GCONFIG_FS_SEL_BIT 4
- #define MPU6050_GCONFIG_FS_SEL_LENGTH 2
- #define MPU6050_GYRO_FS_250 0x00
- #define MPU6050_GYRO_FS_500 0x01
- #define MPU6050_GYRO_FS_1000 0x02
- #define MPU6050_GYRO_FS_2000 0x03
- #define MPU6050_ACONFIG_XA_ST_BIT 7
- #define MPU6050_ACONFIG_YA_ST_BIT 6
- #define MPU6050_ACONFIG_ZA_ST_BIT 5
- #define MPU6050_ACONFIG_AFS_SEL_BIT 4
- #define MPU6050_ACONFIG_AFS_SEL_LENGTH 2
- #define MPU6050_ACONFIG_ACCEL_HPF_BIT 2
- #define MPU6050_ACONFIG_ACCEL_HPF_LENGTH 3
- #define MPU6050_ACCEL_FS_2 0x00
- #define MPU6050_ACCEL_FS_4 0x01
- #define MPU6050_ACCEL_FS_8 0x02
- #define MPU6050_ACCEL_FS_16 0x03
- #define MPU6050_DHPF_RESET 0x00
- #define MPU6050_DHPF_5 0x01
- #define MPU6050_DHPF_2P5 0x02
- #define MPU6050_DHPF_1P25 0x03
- #define MPU6050_DHPF_0P63 0x04
- #define MPU6050_DHPF_HOLD 0x07
- #define MPU6050_TEMP_FIFO_EN_BIT 7
- #define MPU6050_XG_FIFO_EN_BIT 6
- #define MPU6050_YG_FIFO_EN_BIT 5
- #define MPU6050_ZG_FIFO_EN_BIT 4
- #define MPU6050_ACCEL_FIFO_EN_BIT 3

- #define MPU6050_SLV2_FIFO_EN_BIT 2
- #define MPU6050_SLV1_FIFO_EN_BIT 1
- #define MPU6050_SLV0_FIFO_EN_BIT 0
- #define MPU6050_MULT_MST_EN_BIT 7
- #define MPU6050_WAIT_FOR_ES_BIT 6
- #define MPU6050_SLV_3_FIFO_EN_BIT 5
- #define MPU6050_I2C_MST_P_NSR_BIT 4
- #define MPU6050_I2C_MST_CLK_BIT 3
- #define MPU6050_I2C_MST_CLK_LENGTH 4
- #define MPU6050_CLOCK_DIV_348 0x0
- #define MPU6050_CLOCK_DIV_333 0x1
- #define MPU6050_CLOCK_DIV_320 0x2
- #define MPU6050_CLOCK_DIV_308 0x3
- #define MPU6050_CLOCK_DIV_296 0x4
- #define MPU6050_CLOCK_DIV_286 0x5
- #define MPU6050_CLOCK_DIV_276 0x6
- #define MPU6050_CLOCK_DIV_267 0x7
- #define MPU6050_CLOCK_DIV_258 0x8
- #define MPU6050_CLOCK_DIV_500 0x9
- #define MPU6050_CLOCK_DIV_471 0xA
- #define MPU6050_CLOCK_DIV_444 0xB
- #define MPU6050_CLOCK_DIV_421 0xC
- #define MPU6050_CLOCK_DIV_400 0xD
- #define MPU6050_CLOCK_DIV_381 0xE
- #define MPU6050_CLOCK_DIV_364 0xF
- #define MPU6050_I2C_SLV_RW_BIT 7
- #define MPU6050_I2C_SLV_ADDR_BIT 6
- #define MPU6050_I2C_SLV_ADDR_LENGTH 7
- #define MPU6050_I2C_SLV_EN_BIT 7
- #define MPU6050_I2C_SLV_BYTE_SW_BIT 6
- #define MPU6050_I2C_SLV_REG_DIS_BIT 5
- #define MPU6050_I2C_SLV_GRP_BIT 4
- #define MPU6050_I2C_SLV_LEN_BIT 3
- #define MPU6050_I2C_SLV_LEN_LENGTH 4
- #define MPU6050_I2C_SLV4_RW_BIT 7
- #define MPU6050_I2C_SLV4_ADDR_BIT 6
- #define MPU6050_I2C_SLV4_ADDR_LENGTH 7
- #define MPU6050_I2C_SLV4_EN_BIT 7
- #define MPU6050_I2C_SLV4_INT_EN_BIT 6
- #define MPU6050_I2C_SLV4_REG_DIS_BIT 5
- #define MPU6050_I2C_SLV4_MST_DLY_BIT 4
- #define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5
- #define MPU6050_MST_PASS_THROUGH_BIT 7
- #define MPU6050_MST_I2C_SLV4_DONE_BIT 6
- #define MPU6050_MST_I2C_LOST_ARB_BIT 5
- #define MPU6050_MST_I2C_SLV4_NACK_BIT 4
- #define MPU6050_MST_I2C_SLV3_NACK_BIT 3
- #define MPU6050_MST_I2C_SLV2_NACK_BIT 2
- #define MPU6050_MST_I2C_SLV1_NACK_BIT 1
- #define MPU6050_MST_I2C_SLV0_NACK_BIT 0
- #define MPU6050_INTCFG_INT_LEVEL_BIT 7
- #define MPU6050_INTCFG_INT_OPEN_BIT 6
- #define MPU6050_INTCFG_LATCH_INT_EN_BIT 5
- #define MPU6050_INTCFG_INT_RD_CLEAR_BIT 4
- #define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT 3

- #define MPU6050_INTCFG_FSYNC_INT_EN_BIT 2
- #define MPU6050_INTCFG_I2C_BYPASS_EN_BIT 1
- #define MPU6050_INTCFG_CLKOUT_EN_BIT 0
- #define MPU6050_INTMODE_ACTIVEHIGH 0x00
- #define MPU6050_INTMODE_ACTIVELOW 0x01
- #define MPU6050_INTDRV_PUSHPULL 0x00
- #define MPU6050_INTDRV_OPENDRAIN 0x01
- #define MPU6050_INTLATCH_50USPULSE 0x00
- #define MPU6050_INTLATCH_WAITCLEAR 0x01
- #define MPU6050_INTCLEAR_STATUSREAD 0x00
- #define MPU6050_INTCLEAR_ANYREAD 0x01
- #define MPU6050_INTERRUPT_FF_BIT 7
- #define MPU6050_INTERRUPT_MOT_BIT 6
- #define MPU6050_INTERRUPT_ZMOT_BIT 5
- #define MPU6050_INTERRUPT_FIFO_OFLOW_BIT 4
- #define MPU6050_INTERRUPT_I2C_MST_INT_BIT 3
- #define MPU6050_INTERRUPT_PLL_RDY_INT_BIT 2
- #define MPU6050_INTERRUPT_DMP_INT_BIT 1
- #define MPU6050_INTERRUPT_DATA_RDY_BIT 0
- #define MPU6050_DMPINT_5_BIT 5
- #define MPU6050_DMPINT_4_BIT 4
- #define MPU6050_DMPINT_3_BIT 3
- #define MPU6050_DMPINT_2_BIT 2
- #define MPU6050_DMPINT_1_BIT 1
- #define MPU6050_DMPINT_0_BIT 0
- #define MPU6050_MOTION_MOT_XNEG_BIT 7
- #define MPU6050_MOTION_MOT_XPOS_BIT 6
- #define MPU6050_MOTION_MOT_YNEG_BIT 5
- #define MPU6050_MOTION_MOT_YPOS_BIT 4
- #define MPU6050_MOTION_MOT_ZNEG_BIT 3
- #define MPU6050_MOTION_MOT_ZPOS_BIT 2
- #define MPU6050_MOTION_MOT_ZRMOT_BIT 0
- #define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT 7
- #define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT 4
- #define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT 3
- #define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT 2
- #define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT 1
- #define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT 0
- #define MPU6050_PATHRESET_GYRO_RESET_BIT 2
- #define MPU6050_PATHRESET_ACCEL_RESET_BIT 1
- #define MPU6050_PATHRESET_TEMP_RESET_BIT 0
- #define MPU6050_DETECT_ACCEL_ON_DELAY_BIT 5
- #define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH 2
- #define MPU6050_DETECT_FF_COUNT_BIT 3
- #define MPU6050_DETECT_FF_COUNT_LENGTH 2
- #define MPU6050_DETECT_MOT_COUNT_BIT 1
- #define MPU6050_DETECT_MOT_COUNT_LENGTH 2
- #define MPU6050_DETECT_DECREMENT_RESET 0x0
- #define MPU6050_DETECT_DECREMENT_1 0x1
- #define MPU6050_DETECT_DECREMENT_2 0x2
- #define MPU6050_DETECT_DECREMENT_4 0x3
- #define MPU6050_USERCTRL_DMP_EN_BIT 7
- #define MPU6050_USERCTRL_FIFO_EN_BIT 6
- #define MPU6050_USERCTRL_I2C_MST_EN_BIT 5
- #define MPU6050_USERCTRL_I2C_IF_DIS_BIT 4

- #define MPU6050_USERCTRL_DMP_RESET_BIT 3
- #define MPU6050_USERCTRL_FIFO_RESET_BIT 2
- #define MPU6050_USERCTRL_I2C_MST_RESET_BIT 1
- #define MPU6050_USERCTRL_SIG_COND_RESET_BIT 0
- #define MPU6050_PWR1_DEVICE_RESET_BIT 7
- #define MPU6050_PWR1_SLEEP_BIT 6
- #define MPU6050_PWR1_CYCLE_BIT 5
- #define MPU6050_PWR1_TEMP_DIS_BIT 3
- #define MPU6050_PWR1_CLKSEL_BIT 2
- #define MPU6050_PWR1_CLKSEL_LENGTH 3
- #define MPU6050_CLOCK_INTERNAL 0x00
- #define MPU6050_CLOCK_PLL_XGYRO 0x01
- #define MPU6050_CLOCK_PLL_YGYRO 0x02
- #define MPU6050_CLOCK_PLL_ZGYRO 0x03
- #define MPU6050_CLOCK_PLL_EXT32K 0x04
- #define MPU6050_CLOCK_PLL_EXT19M 0x05
- #define MPU6050_CLOCK_KEEP_RESET 0x07
- #define MPU6050_PWR2_LP_WAKE_CTRL_BIT 7
- #define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH 2
- #define MPU6050_PWR2_STBY_XA_BIT 5
- #define MPU6050_PWR2_STBY_YA_BIT 4
- #define MPU6050_PWR2_STBY_ZA_BIT 3
- #define MPU6050_PWR2_STBY_XG_BIT 2
- #define MPU6050_PWR2_STBY_YG_BIT 1
- #define MPU6050_PWR2_STBY_ZG_BIT 0
- #define MPU6050_WAKE_FREQ_1P25 0x0
- #define MPU6050_WAKE_FREQ_2P5 0x1
- #define MPU6050_WAKE_FREQ_5 0x2
- #define MPU6050_WAKE_FREQ_10 0x3
- #define MPU6050_BANKSEL_PRFTCH_EN_BIT 6
- #define MPU6050_BANKSEL_CFG_USER_BANK_BIT 5
- #define MPU6050_BANKSEL_MEM_SEL_BIT 4
- #define MPU6050_BANKSEL_MEM_SEL_LENGTH 5
- #define MPU6050_WHO_AM_I_BIT 6
- #define MPU6050_WHO_AM_I_LENGTH 6
- #define MPU6050_DMP_MEMORY_BANKS 8
- #define MPU6050_DMP_MEMORY_BANK_SIZE 256
- #define MPU6050_DMP_MEMORY_CHUNK_SIZE 16

4.7.1 Macro Definition Documentation

4.7.1.1 #define MPU6050_ACCEL_FIFO_EN_BIT 3

Definition at line 223 of file ArduinoGyroscopeMPU6050.h.

4.7.1.2 #define MPU6050_ACCEL_FS_16 0x03

Definition at line 210 of file ArduinoGyroscopeMPU6050.h.

4.7.1.3 #define MPU6050_ACCEL_FS_2 0x00

Definition at line 207 of file ArduinoGyroscopeMPU6050.h.

4.7.1.4 #define MPU6050_ACCEL_FS_4 0x01

Definition at line 208 of file ArduinoGyroscopeMPU6050.h.

4.7.1.5 #define MPU6050_ACCEL_FS_8 0x02

Definition at line 209 of file ArduinoGyroscopeMPU6050.h.

4.7.1.6 #define MPU6050_ACONFIG_ACCEL_HPF_BIT 2

Definition at line 204 of file ArduinoGyroscopeMPU6050.h.

4.7.1.7 #define MPU6050_ACONFIG_ACCEL_HPF_LENGTH 3

Definition at line 205 of file ArduinoGyroscopeMPU6050.h.

4.7.1.8 #define MPU6050_ACONFIG_AFS_SEL_BIT 4

Definition at line 202 of file ArduinoGyroscopeMPU6050.h.

4.7.1.9 #define MPU6050_ACONFIG_AFS_SEL_LENGTH 2

Definition at line 203 of file ArduinoGyroscopeMPU6050.h.

4.7.1.10 #define MPU6050_ACONFIG_XA_ST_BIT 7

Definition at line 199 of file ArduinoGyroscopeMPU6050.h.

4.7.1.11 #define MPU6050_ACONFIG_YA_ST_BIT 6

Definition at line 200 of file ArduinoGyroscopeMPU6050.h.

4.7.1.12 #define MPU6050_ACONFIG_ZA_ST_BIT 5

Definition at line 201 of file ArduinoGyroscopeMPU6050.h.

4.7.1.13 #define MPU6050_BANKSEL_CFG_USER_BANK_BIT 5

Definition at line 389 of file ArduinoGyroscopeMPU6050.h.

4.7.1.14 #define MPU6050_BANKSEL_MEM_SEL_BIT 4

Definition at line 390 of file ArduinoGyroscopeMPU6050.h.

4.7.1.15 #define MPU6050_BANKSEL_MEM_SEL_LENGTH 5

Definition at line 391 of file ArduinoGyroscopeMPU6050.h.

4.7.1.16 #define MPU6050_BANKSEL_PRFTCH_EN_BIT 6

Definition at line 388 of file ArduinoGyroscopeMPU6050.h.

4.7.1.17 #define MPU6050_CFG_DLPF_CFG_BIT 2

Definition at line 171 of file ArduinoGyroscopeMPU6050.h.

4.7.1.18 #define MPU6050_CFG_DLPF_CFG_LENGTH 3

Definition at line 172 of file ArduinoGyroscopeMPU6050.h.

4.7.1.19 #define MPU6050_CFG_EXT_SYNC_SET_BIT 5

Definition at line 169 of file ArduinoGyroscopeMPU6050.h.

4.7.1.20 #define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3

Definition at line 170 of file ArduinoGyroscopeMPU6050.h.

4.7.1.21 #define MPU6050_CLOCK_DIV_258 0x8

Definition at line 243 of file ArduinoGyroscopeMPU6050.h.

4.7.1.22 #define MPU6050_CLOCK_DIV_267 0x7

Definition at line 242 of file ArduinoGyroscopeMPU6050.h.

4.7.1.23 #define MPU6050_CLOCK_DIV_276 0x6

Definition at line 241 of file ArduinoGyroscopeMPU6050.h.

4.7.1.24 #define MPU6050_CLOCK_DIV_286 0x5

Definition at line 240 of file ArduinoGyroscopeMPU6050.h.

4.7.1.25 #define MPU6050_CLOCK_DIV_296 0x4

Definition at line 239 of file ArduinoGyroscopeMPU6050.h.

4.7.1.26 #define MPU6050_CLOCK_DIV_308 0x3

Definition at line 238 of file ArduinoGyroscopeMPU6050.h.

4.7.1.27 #define MPU6050_CLOCK_DIV_320 0x2

Definition at line 237 of file ArduinoGyroscopeMPU6050.h.

4.7.1.28 #define MPU6050_CLOCK_DIV_333 0x1

Definition at line 236 of file ArduinoGyroscopeMPU6050.h.

4.7.1.29 #define MPU6050_CLOCK_DIV_348 0x0

Definition at line 235 of file ArduinoGyroscopeMPU6050.h.

4.7.1.30 #define MPU6050_CLOCK_DIV_364 0xF

Definition at line 250 of file ArduinoGyroscopeMPU6050.h.

4.7.1.31 #define MPU6050_CLOCK_DIV_381 0xE

Definition at line 249 of file ArduinoGyroscopeMPU6050.h.

4.7.1.32 #define MPU6050_CLOCK_DIV_400 0xD

Definition at line 248 of file ArduinoGyroscopeMPU6050.h.

4.7.1.33 #define MPU6050_CLOCK_DIV_421 0xC

Definition at line 247 of file ArduinoGyroscopeMPU6050.h.

4.7.1.34 #define MPU6050_CLOCK_DIV_444 0xB

Definition at line 246 of file ArduinoGyroscopeMPU6050.h.

4.7.1.35 #define MPU6050_CLOCK_DIV_471 0xA

Definition at line 245 of file ArduinoGyroscopeMPU6050.h.

4.7.1.36 #define MPU6050_CLOCK_DIV_500 0x9

Definition at line 244 of file ArduinoGyroscopeMPU6050.h.

4.7.1.37 #define MPU6050_CLOCK_INTERNAL 0x00

Definition at line 366 of file ArduinoGyroscopeMPU6050.h.

4.7.1.38 #define MPU6050_CLOCK_KEEP_RESET 0x07

Definition at line 372 of file ArduinoGyroscopeMPU6050.h.

4.7.1.39 #define MPU6050_CLOCK_PLL_EXT19M 0x05

Definition at line 371 of file ArduinoGyroscopeMPU6050.h.

4.7.1.40 #define MPU6050_CLOCK_PLL_EXT32K 0x04

Definition at line 370 of file ArduinoGyroscopeMPU6050.h.

4.7.1.41 #define MPU6050_CLOCK_PLL_XGYRO 0x01

Definition at line 367 of file ArduinoGyroscopeMPU6050.h.

4.7.1.42 #define MPU6050_CLOCK_PLL_YGYRO 0x02

Definition at line 368 of file ArduinoGyroscopeMPU6050.h.

4.7.1.43 #define MPU6050_CLOCK_PLL_ZGYRO 0x03

Definition at line 369 of file ArduinoGyroscopeMPU6050.h.

4.7.1.44 #define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT 7

Definition at line 327 of file ArduinoGyroscopeMPU6050.h.

4.7.1.45 #define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT 0

Definition at line 332 of file ArduinoGyroscopeMPU6050.h.

4.7.1.46 #define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT 1

Definition at line 331 of file ArduinoGyroscopeMPU6050.h.

4.7.1.47 #define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT 2

Definition at line 330 of file ArduinoGyroscopeMPU6050.h.

4.7.1.48 #define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT 3

Definition at line 329 of file ArduinoGyroscopeMPU6050.h.

4.7.1.49 #define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT 4

Definition at line 328 of file ArduinoGyroscopeMPU6050.h.

4.7.1.50 #define MPU6050_DETECT_ACCEL_ON_DELAY_BIT 5

Definition at line 338 of file ArduinoGyroscopeMPU6050.h.

4.7.1.51 #define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH 2

Definition at line 339 of file ArduinoGyroscopeMPU6050.h.

4.7.1.52 #define MPU6050_DETECT_DECREMENT_1 0x1

Definition at line 346 of file ArduinoGyroscopeMPU6050.h.

4.7.1.53 #define MPU6050_DETECT_DECREMENT_2 0x2

Definition at line 347 of file ArduinoGyroscopeMPU6050.h.

4.7.1.54 #define MPU6050_DETECT_DECREMENT_4 0x3

Definition at line 348 of file ArduinoGyroscopeMPU6050.h.

4.7.1.55 #define MPU6050_DETECT_DECREMENT_RESET 0x0

Definition at line 345 of file ArduinoGyroscopeMPU6050.h.

4.7.1.56 #define MPU6050_DETECT_FF_COUNT_BIT 3

Definition at line 340 of file ArduinoGyroscopeMPU6050.h.

4.7.1.57 #define MPU6050_DETECT_FF_COUNT_LENGTH 2

Definition at line 341 of file ArduinoGyroscopeMPU6050.h.

4.7.1.58 #define MPU6050_DETECT_MOT_COUNT_BIT 1

Definition at line 342 of file ArduinoGyroscopeMPU6050.h.

4.7.1.59 #define MPU6050_DETECT_MOT_COUNT_LENGTH 2

Definition at line 343 of file ArduinoGyroscopeMPU6050.h.

4.7.1.60 #define MPU6050_DHPF_0P63 0x04

Definition at line 216 of file ArduinoGyroscopeMPU6050.h.

4.7.1.61 #define MPU6050_DHPF_1P25 0x03

Definition at line 215 of file ArduinoGyroscopeMPU6050.h.

4.7.1.62 #define MPU6050_DHPF_2P5 0x02

Definition at line 214 of file ArduinoGyroscopeMPU6050.h.

4.7.1.63 #define MPU6050_DHPF_5 0x01

Definition at line 213 of file ArduinoGyroscopeMPU6050.h.

4.7.1.64 #define MPU6050_DHPF_HOLD 0x07

Definition at line 217 of file ArduinoGyroscopeMPU6050.h.

4.7.1.65 #define MPU6050_DHPF_RESET 0x00

Definition at line 212 of file ArduinoGyroscopeMPU6050.h.

4.7.1.66 #define MPU6050_DLPF_BW_10 0x05

Definition at line 188 of file ArduinoGyroscopeMPU6050.h.

4.7.1.67 #define MPU6050_DLPF_BW_188 0x01

Definition at line 184 of file ArduinoGyroscopeMPU6050.h.

4.7.1.68 #define MPU6050_DLPF_BW_20 0x04

Definition at line 187 of file ArduinoGyroscopeMPU6050.h.

4.7.1.69 #define MPU6050_DLPF_BW_256 0x00

Definition at line 183 of file ArduinoGyroscopeMPU6050.h.

4.7.1.70 #define MPU6050_DLPF_BW_42 0x03

Definition at line 186 of file ArduinoGyroscopeMPU6050.h.

4.7.1.71 #define MPU6050_DLPF_BW_5 0x06

Definition at line 189 of file ArduinoGyroscopeMPU6050.h.

4.7.1.72 #define MPU6050_DLPF_BW_98 0x02

Definition at line 185 of file ArduinoGyroscopeMPU6050.h.

4.7.1.73 #define MPU6050_DMP_MEMORY_BANK_SIZE 256

Definition at line 397 of file ArduinoGyroscopeMPU6050.h.

4.7.1.74 #define MPU6050_DMP_MEMORY_BANKS 8

Definition at line 396 of file ArduinoGyroscopeMPU6050.h.

4.7.1.75 #define MPU6050_DMP_MEMORY_CHUNK_SIZE 16

Definition at line 398 of file ArduinoGyroscopeMPU6050.h.

4.7.1.76 #define MPU6050_DMPINT_0_BIT 0

Definition at line 317 of file ArduinoGyroscopeMPU6050.h.

4.7.1.77 #define MPU6050_DMPINT_1_BIT 1

Definition at line 316 of file ArduinoGyroscopeMPU6050.h.

4.7.1.78 #define MPU6050_DMPINT_2_BIT 2

Definition at line 315 of file ArduinoGyroscopeMPU6050.h.

4.7.1.79 #define MPU6050_DMPINT_3_BIT 3

Definition at line 314 of file ArduinoGyroscopeMPU6050.h.

4.7.1.80 #define MPU6050_DMPINT_4_BIT 4

Definition at line 313 of file ArduinoGyroscopeMPU6050.h.

4.7.1.81 #define MPU6050_DMPINT_5_BIT 5

Definition at line 312 of file ArduinoGyroscopeMPU6050.h.

4.7.1.82 #define MPU6050_EXT_SYNC_ACCEL_XOUT_L 0x5

Definition at line 179 of file ArduinoGyroscopeMPU6050.h.

4.7.1.83 #define MPU6050_EXT_SYNC_ACCEL_YOUT_L 0x6

Definition at line 180 of file ArduinoGyroscopeMPU6050.h.

4.7.1.84 #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L 0x7

Definition at line 181 of file ArduinoGyroscopeMPU6050.h.

4.7.1.85 #define MPU6050_EXT_SYNC_DISABLED 0x0

Definition at line 174 of file ArduinoGyroscopeMPU6050.h.

4.7.1.86 #define MPU6050_EXT_SYNC_GYRO_XOUT_L 0x2

Definition at line 176 of file ArduinoGyroscopeMPU6050.h.

4.7.1.87 #define MPU6050_EXT_SYNC_GYRO_YOUT_L 0x3

Definition at line 177 of file ArduinoGyroscopeMPU6050.h.

4.7.1.88 #define MPU6050_EXT_SYNC_GYRO_ZOUT_L 0x4

Definition at line 178 of file ArduinoGyroscopeMPU6050.h.

4.7.1.89 #define MPU6050_EXT_SYNC_TEMP_OUT_L 0x1

Definition at line 175 of file ArduinoGyroscopeMPU6050.h.

4.7.1.90 #define MPU6050_GCONFIG_FS_SEL_BIT 4

Definition at line 191 of file ArduinoGyroscopeMPU6050.h.

4.7.1.91 #define MPU6050_GCONFIG_FS_SEL_LENGTH 2

Definition at line 192 of file ArduinoGyroscopeMPU6050.h.

4.7.1.92 #define MPU6050_GYRO_FS_1000 0x02

Definition at line 196 of file ArduinoGyroscopeMPU6050.h.

4.7.1.93 #define MPU6050_GYRO_FS_2000 0x03

Definition at line 197 of file ArduinoGyroscopeMPU6050.h.

4.7.1.94 #define MPU6050_GYRO_FS_250 0x00

Definition at line 194 of file ArduinoGyroscopeMPU6050.h.

4.7.1.95 #define MPU6050_GYRO_FS_500 0x01

Definition at line 195 of file ArduinoGyroscopeMPU6050.h.

4.7.1.96 #define MPU6050_I2C_MST_CLK_BIT 3

Definition at line 232 of file ArduinoGyroscopeMPU6050.h.

4.7.1.97 #define MPU6050_I2C_MST_CLK_LENGTH 4

Definition at line 233 of file ArduinoGyroscopeMPU6050.h.

4.7.1.98 #define MPU6050_I2C_MST_P_NSR_BIT 4

Definition at line 231 of file ArduinoGyroscopeMPU6050.h.

4.7.1.99 #define MPU6050_I2C_SLV4_ADDR_BIT 6

Definition at line 263 of file ArduinoGyroscopeMPU6050.h.

4.7.1.100 #define MPU6050_I2C_SLV4_ADDR_LENGTH 7

Definition at line 264 of file ArduinoGyroscopeMPU6050.h.

4.7.1.101 #define MPU6050_I2C_SLV4_EN_BIT 7

Definition at line 265 of file ArduinoGyroscopeMPU6050.h.

4.7.1.102 #define MPU6050_I2C_SLV4_INT_EN_BIT 6

Definition at line 266 of file ArduinoGyroscopeMPU6050.h.

4.7.1.103 #define MPU6050_I2C_SLV4_MST_DLY_BIT 4

Definition at line 268 of file ArduinoGyroscopeMPU6050.h.

4.7.1.104 #define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5

Definition at line 269 of file ArduinoGyroscopeMPU6050.h.

4.7.1.105 #define MPU6050_I2C_SLV4_REG_DIS_BIT 5

Definition at line 267 of file ArduinoGyroscopeMPU6050.h.

4.7.1.106 #define MPU6050_I2C_SLV4_RW_BIT 7

Definition at line 262 of file ArduinoGyroscopeMPU6050.h.

4.7.1.107 #define MPU6050_I2C_SLV_ADDR_BIT 6

Definition at line 253 of file ArduinoGyroscopeMPU6050.h.

4.7.1.108 #define MPU6050_I2C_SLV_ADDR_LENGTH 7

Definition at line 254 of file ArduinoGyroscopeMPU6050.h.

4.7.1.109 #define MPU6050_I2C_SLV_BYTE_SW_BIT 6

Definition at line 256 of file ArduinoGyroscopeMPU6050.h.

4.7.1.110 #define MPU6050_I2C_SLV_EN_BIT 7

Definition at line 255 of file ArduinoGyroscopeMPU6050.h.

4.7.1.111 #define MPU6050_I2C_SLV_GRP_BIT 4

Definition at line 258 of file ArduinoGyroscopeMPU6050.h.

4.7.1.112 #define MPU6050_I2C_SLV_LEN_BIT 3

Definition at line 259 of file ArduinoGyroscopeMPU6050.h.

4.7.1.113 #define MPU6050_I2C_SLV_LEN_LENGTH 4

Definition at line 260 of file ArduinoGyroscopeMPU6050.h.

4.7.1.114 #define MPU6050_I2C_SLV_REG_DIS_BIT 5

Definition at line 257 of file ArduinoGyroscopeMPU6050.h.

4.7.1.115 #define MPU6050_I2C_SLV_RW_BIT 7

Definition at line 252 of file ArduinoGyroscopeMPU6050.h.

4.7.1.116 #define MPU6050_INTCFG_CLKOUT_EN_BIT 0

Definition at line 287 of file ArduinoGyroscopeMPU6050.h.

4.7.1.117 #define MPU6050_INTCFG_FSYNC_INT_EN_BIT 2

Definition at line 285 of file ArduinoGyroscopeMPU6050.h.

4.7.1.118 #define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT 3

Definition at line 284 of file ArduinoGyroscopeMPU6050.h.

4.7.1.119 #define MPU6050_INTCFG_I2C_BYPASS_EN_BIT 1

Definition at line 286 of file ArduinoGyroscopeMPU6050.h.

4.7.1.120 #define MPU6050_INTCFG_INT_LEVEL_BIT 7

Definition at line 280 of file ArduinoGyroscopeMPU6050.h.

4.7.1.121 #define MPU6050_INTCFG_INT_OPEN_BIT 6

Definition at line 281 of file ArduinoGyroscopeMPU6050.h.

4.7.1.122 #define MPU6050_INTCFG_INT_RD_CLEAR_BIT 4

Definition at line 283 of file ArduinoGyroscopeMPU6050.h.

4.7.1.123 #define MPU6050_INTCFG_LATCH_INT_EN_BIT 5

Definition at line 282 of file ArduinoGyroscopeMPU6050.h.

4.7.1.124 #define MPU6050_INTCLEAR_ANYREAD 0x01

Definition at line 299 of file ArduinoGyroscopeMPU6050.h.

4.7.1.125 #define MPU6050_INTCLEAR_STATUSREAD 0x00

Definition at line 298 of file ArduinoGyroscopeMPU6050.h.

4.7.1.126 #define MPU6050_INTDRV_OPENDRAIN 0x01

Definition at line 293 of file ArduinoGyroscopeMPU6050.h.

4.7.1.127 #define MPU6050_INTDRV_PUSHPULL 0x00

Definition at line 292 of file ArduinoGyroscopeMPU6050.h.

4.7.1.128 #define MPU6050_INTERRUPT_DATA_RDY_BIT 0

Definition at line 308 of file ArduinoGyroscopeMPU6050.h.

4.7.1.129 #define MPU6050_INTERRUPT_DMP_INT_BIT 1

Definition at line 307 of file ArduinoGyroscopeMPU6050.h.

4.7.1.130 #define MPU6050_INTERRUPT_FF_BIT 7

Definition at line 301 of file ArduinoGyroscopeMPU6050.h.

4.7.1.131 #define MPU6050_INTERRUPT_FIFO_OFLOW_BIT 4

Definition at line 304 of file ArduinoGyroscopeMPU6050.h.

4.7.1.132 #define MPU6050_INTERRUPT_I2C_MST_INT_BIT 3

Definition at line 305 of file ArduinoGyroscopeMPU6050.h.

4.7.1.133 #define MPU6050_INTERRUPT_MOT_BIT 6

Definition at line 302 of file ArduinoGyroscopeMPU6050.h.

4.7.1.134 #define MPU6050_INTERRUPT_PLL_RDY_INT_BIT 2

Definition at line 306 of file ArduinoGyroscopeMPU6050.h.

4.7.1.135 #define MPU6050_INTERRUPT_ZMOT_BIT 5

Definition at line 303 of file ArduinoGyroscopeMPU6050.h.

4.7.1.136 #define MPU6050_INTLATCH_50USPULSE 0x00

Definition at line 295 of file ArduinoGyroscopeMPU6050.h.

4.7.1.137 #define MPU6050_INTLATCH_WAITCLEAR 0x01

Definition at line 296 of file ArduinoGyroscopeMPU6050.h.

4.7.1.138 #define MPU6050_INTMODE_ACTIVEHIGH 0x00

Definition at line 289 of file ArduinoGyroscopeMPU6050.h.

4.7.1.139 #define MPU6050_INTMODE_ACTIVELOW 0x01

Definition at line 290 of file ArduinoGyroscopeMPU6050.h.

4.7.1.140 #define MPU6050_MOTION_MOT_XNEG_BIT 7

Definition at line 319 of file ArduinoGyroscopeMPU6050.h.

4.7.1.141 #define MPU6050_MOTION_MOT_XPOS_BIT 6

Definition at line 320 of file ArduinoGyroscopeMPU6050.h.

4.7.1.142 #define MPU6050_MOTION_MOT_YNEG_BIT 5

Definition at line 321 of file ArduinoGyroscopeMPU6050.h.

4.7.1.143 #define MPU6050_MOTION_MOT_YPOS_BIT 4

Definition at line 322 of file ArduinoGyroscopeMPU6050.h.

4.7.1.144 #define MPU6050_MOTION_MOT_ZNEG_BIT 3

Definition at line 323 of file ArduinoGyroscopeMPU6050.h.

4.7.1.145 #define MPU6050_MOTION_MOT_ZPOS_BIT 2

Definition at line 324 of file ArduinoGyroscopeMPU6050.h.

4.7.1.146 #define MPU6050_MOTION_MOT_ZRMOT_BIT 0

Definition at line 325 of file ArduinoGyroscopeMPU6050.h.

4.7.1.147 #define MPU6050_MST_I2C_LOST_ARB_BIT 5

Definition at line 273 of file ArduinoGyroscopeMPU6050.h.

4.7.1.148 #define MPU6050_MST_I2C_SLV0_NACK_BIT 0

Definition at line 278 of file ArduinoGyroscopeMPU6050.h.

4.7.1.149 #define MPU6050_MST_I2C_SLV1_NACK_BIT 1

Definition at line 277 of file ArduinoGyroscopeMPU6050.h.

4.7.1.150 #define MPU6050_MST_I2C_SLV2_NACK_BIT 2

Definition at line 276 of file ArduinoGyroscopeMPU6050.h.

4.7.1.151 #define MPU6050_MST_I2C_SLV3_NACK_BIT 3

Definition at line 275 of file ArduinoGyroscopeMPU6050.h.

4.7.1.152 #define MPU6050_MST_I2C_SLV4_DONE_BIT 6

Definition at line 272 of file ArduinoGyroscopeMPU6050.h.

4.7.1.153 #define MPU6050_MST_I2C_SLV4_NACK_BIT 4

Definition at line 274 of file ArduinoGyroscopeMPU6050.h.

4.7.1.154 #define MPU6050_MST_PASS_THROUGH_BIT 7

Definition at line 271 of file ArduinoGyroscopeMPU6050.h.

4.7.1.155 #define MPU6050_MULT_MST_EN_BIT 7

Definition at line 228 of file ArduinoGyroscopeMPU6050.h.

4.7.1.156 #define MPU6050_PATHRESET_ACCEL_RESET_BIT 1

Definition at line 335 of file ArduinoGyroscopeMPU6050.h.

4.7.1.157 #define MPU6050_PATHRESET_GYRO_RESET_BIT 2

Definition at line 334 of file ArduinoGyroscopeMPU6050.h.

4.7.1.158 #define MPU6050_PATHRESET_TEMP_RESET_BIT 0

Definition at line 336 of file ArduinoGyroscopeMPU6050.h.

4.7.1.159 #define MPU6050_PWR1_CLKSEL_BIT 2

Definition at line 363 of file ArduinoGyroscopeMPU6050.h.

4.7.1.160 #define MPU6050_PWR1_CLKSEL_LENGTH 3

Definition at line 364 of file ArduinoGyroscopeMPU6050.h.

4.7.1.161 #define MPU6050_PWR1_CYCLE_BIT 5

Definition at line 361 of file ArduinoGyroscopeMPU6050.h.

4.7.1.162 #define MPU6050_PWR1_DEVICE_RESET_BIT 7

Definition at line 359 of file ArduinoGyroscopeMPU6050.h.

4.7.1.163 #define MPU6050_PWR1_SLEEP_BIT 6

Definition at line 360 of file ArduinoGyroscopeMPU6050.h.

4.7.1.164 #define MPU6050_PWR1_TEMP_DIS_BIT 3

Definition at line 362 of file ArduinoGyroscopeMPU6050.h.

4.7.1.165 #define MPU6050_PWR2_LP_WAKE_CTRL_BIT 7

Definition at line 374 of file ArduinoGyroscopeMPU6050.h.

4.7.1.166 #define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH 2

Definition at line 375 of file ArduinoGyroscopeMPU6050.h.

4.7.1.167 #define MPU6050_PWR2_STBY_XA_BIT 5

Definition at line 376 of file ArduinoGyroscopeMPU6050.h.

4.7.1.168 #define MPU6050_PWR2_STBY_XG_BIT 2

Definition at line 379 of file ArduinoGyroscopeMPU6050.h.

4.7.1.169 #define MPU6050_PWR2_STBY_YA_BIT 4

Definition at line 377 of file ArduinoGyroscopeMPU6050.h.

4.7.1.170 #define MPU6050_PWR2_STBY_YG_BIT 1

Definition at line 380 of file ArduinoGyroscopeMPU6050.h.

4.7.1.171 #define MPU6050_PWR2_STBY_ZA_BIT 3

Definition at line 378 of file ArduinoGyroscopeMPU6050.h.

4.7.1.172 #define MPU6050_PWR2_STBY_ZG_BIT 0

Definition at line 381 of file ArduinoGyroscopeMPU6050.h.

4.7.1.173 #define MPU6050_RA_ACCEL_CONFIG 0x1C

Definition at line 71 of file ArduinoGyroscopeMPU6050.h.

4.7.1.174 #define MPU6050_RA_ACCEL_XOUT_H 0x3B

Definition at line 102 of file ArduinoGyroscopeMPU6050.h.

4.7.1.175 #define MPU6050_RA_ACCEL_XOUT_L 0x3C

Definition at line 103 of file ArduinoGyroscopeMPU6050.h.

4.7.1.176 #define MPU6050_RA_ACCEL_YOUT_H 0x3D

Definition at line 104 of file ArduinoGyroscopeMPU6050.h.

4.7.1.177 #define MPU6050_RA_ACCEL_YOUT_L 0x3E

Definition at line 105 of file ArduinoGyroscopeMPU6050.h.

4.7.1.178 #define MPU6050_RA_ACCEL_ZOUT_H 0x3F

Definition at line 106 of file ArduinoGyroscopeMPU6050.h.

4.7.1.179 #define MPU6050_RA_ACCEL_ZOUT_L 0x40

Definition at line 107 of file ArduinoGyroscopeMPU6050.h.

4.7.1.180 #define MPU6050_RA_BANK_SEL 0x6D

Definition at line 151 of file ArduinoGyroscopeMPU6050.h.

4.7.1.181 #define MPU6050_RA_CONFIG 0x1A

Definition at line 69 of file ArduinoGyroscopeMPU6050.h.

4.7.1.182 #define MPU6050_RA_DMP_CFG_1 0x70

Definition at line 154 of file ArduinoGyroscopeMPU6050.h.

4.7.1.183 #define MPU6050_RA_DMP_CFG_2 0x71

Definition at line 155 of file ArduinoGyroscopeMPU6050.h.

4.7.1.184 #define MPU6050_RA_DMP_INT_STATUS 0x39

Definition at line 100 of file ArduinoGyroscopeMPU6050.h.

4.7.1.185 #define MPU6050_RA_EXT_SENS_DATA_00 0x49

Definition at line 116 of file ArduinoGyroscopeMPU6050.h.

4.7.1.186 #define MPU6050_RA_EXT_SENS_DATA_01 0x4A

Definition at line 117 of file ArduinoGyroscopeMPU6050.h.

4.7.1.187 #define MPU6050_RA_EXT_SENS_DATA_02 0x4B

Definition at line 118 of file ArduinoGyroscopeMPU6050.h.

4.7.1.188 #define MPU6050_RA_EXT_SENS_DATA_03 0x4C

Definition at line 119 of file ArduinoGyroscopeMPU6050.h.

4.7.1.189 #define MPU6050_RA_EXT_SENS_DATA_04 0x4D

Definition at line 120 of file ArduinoGyroscopeMPU6050.h.

4.7.1.190 #define MPU6050_RA_EXT_SENS_DATA_05 0x4E

Definition at line 121 of file ArduinoGyroscopeMPU6050.h.

4.7.1.191 #define MPU6050_RA_EXT_SENS_DATA_06 0x4F

Definition at line 122 of file ArduinoGyroscopeMPU6050.h.

4.7.1.192 #define MPU6050_RA_EXT_SENS_DATA_07 0x50

Definition at line 123 of file ArduinoGyroscopeMPU6050.h.

4.7.1.193 #define MPU6050_RA_EXT_SENS_DATA_08 0x51

Definition at line 124 of file ArduinoGyroscopeMPU6050.h.

4.7.1.194 #define MPU6050_RA_EXT_SENS_DATA_09 0x52

Definition at line 125 of file ArduinoGyroscopeMPU6050.h.

4.7.1.195 #define MPU6050_RA_EXT_SENS_DATA_10 0x53

Definition at line 126 of file ArduinoGyroscopeMPU6050.h.

4.7.1.196 #define MPU6050_RA_EXT_SENS_DATA_11 0x54

Definition at line 127 of file ArduinoGyroscopeMPU6050.h.

4.7.1.197 #define MPU6050_RA_EXT_SENS_DATA_12 0x55

Definition at line 128 of file ArduinoGyroscopeMPU6050.h.

4.7.1.198 #define MPU6050_RA_EXT_SENS_DATA_13 0x56

Definition at line 129 of file ArduinoGyroscopeMPU6050.h.

4.7.1.199 #define MPU6050_RA_EXT_SENS_DATA_14 0x57

Definition at line 130 of file ArduinoGyroscopeMPU6050.h.

4.7.1.200 #define MPU6050_RA_EXT_SENS_DATA_15 0x58

Definition at line 131 of file ArduinoGyroscopeMPU6050.h.

4.7.1.201 #define MPU6050_RA_EXT_SENS_DATA_16 0x59

Definition at line 132 of file ArduinoGyroscopeMPU6050.h.

4.7.1.202 #define MPU6050_RA_EXT_SENS_DATA_17 0x5A

Definition at line 133 of file ArduinoGyroscopeMPU6050.h.

4.7.1.203 #define MPU6050_RA_EXT_SENS_DATA_18 0x5B

Definition at line 134 of file ArduinoGyroscopeMPU6050.h.

4.7.1.204 #define MPU6050_RA_EXT_SENS_DATA_19 0x5C

Definition at line 135 of file ArduinoGyroscopeMPU6050.h.

4.7.1.205 #define MPU6050_RA_EXT_SENS_DATA_20 0x5D

Definition at line 136 of file ArduinoGyroscopeMPU6050.h.

4.7.1.206 #define MPU6050_RA_EXT_SENS_DATA_21 0x5E

Definition at line 137 of file ArduinoGyroscopeMPU6050.h.

4.7.1.207 #define MPU6050_RA_EXT_SENS_DATA_22 0x5F

Definition at line 138 of file ArduinoGyroscopeMPU6050.h.

4.7.1.208 #define MPU6050_RA_EXT_SENS_DATA_23 0x60

Definition at line 139 of file ArduinoGyroscopeMPU6050.h.

4.7.1.209 #define MPU6050_RA_FF_DUR 0x1E

Definition at line 73 of file ArduinoGyroscopeMPU6050.h.

4.7.1.210 #define MPU6050_RA_FF_THR 0x1D

Definition at line 72 of file ArduinoGyroscopeMPU6050.h.

4.7.1.211 #define MPU6050_RA_FIFO_COUNTH 0x72

Definition at line 156 of file ArduinoGyroscopeMPU6050.h.

4.7.1.212 #define MPU6050_RA_FIFO_COUNTL 0x73

Definition at line 157 of file ArduinoGyroscopeMPU6050.h.

4.7.1.213 #define MPU6050_RA_FIFO_EN 0x23

Definition at line 78 of file ArduinoGyroscopeMPU6050.h.

4.7.1.214 #define MPU6050_RA_FIFO_R_W 0x74

Definition at line 158 of file ArduinoGyroscopeMPU6050.h.

4.7.1.215 #define MPU6050_RA_GYRO_CONFIG 0x1B

Definition at line 70 of file ArduinoGyroscopeMPU6050.h.

4.7.1.216 #define MPU6050_RA_GYRO_XOUT_H 0x43

Definition at line 110 of file ArduinoGyroscopeMPU6050.h.

4.7.1.217 #define MPU6050_RA_GYRO_XOUT_L 0x44

Definition at line 111 of file ArduinoGyroscopeMPU6050.h.

4.7.1.218 #define MPU6050_RA_GYRO_YOUT_H 0x45

Definition at line 112 of file ArduinoGyroscopeMPU6050.h.

4.7.1.219 #define MPU6050_RA_GYRO_YOUT_L 0x46

Definition at line 113 of file ArduinoGyroscopeMPU6050.h.

4.7.1.220 #define MPU6050_RA_GYRO_ZOUT_H 0x47

Definition at line 114 of file ArduinoGyroscopeMPU6050.h.

4.7.1.221 #define MPU6050_RA_GYRO_ZOUT_L 0x48

Definition at line 115 of file ArduinoGyroscopeMPU6050.h.

4.7.1.222 #define MPU6050_RA_I2C_MST_CTRL 0x24

Definition at line 79 of file ArduinoGyroscopeMPU6050.h.

4.7.1.223 #define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67

Definition at line 145 of file ArduinoGyroscopeMPU6050.h.

4.7.1.224 #define MPU6050_RA_I2C_MST_STATUS 0x36

Definition at line 97 of file ArduinoGyroscopeMPU6050.h.

4.7.1.225 #define MPU6050_RA_I2C_SLV0_ADDR 0x25

Definition at line 80 of file ArduinoGyroscopeMPU6050.h.

4.7.1.226 #define MPU6050_RA_I2C_SLV0_CTRL 0x27

Definition at line 82 of file ArduinoGyroscopeMPU6050.h.

4.7.1.227 #define MPU6050_RA_I2C_SLV0_DO 0x63

Definition at line 141 of file ArduinoGyroscopeMPU6050.h.

4.7.1.228 #define MPU6050_RA_I2C_SLV0_REG 0x26

Definition at line 81 of file ArduinoGyroscopeMPU6050.h.

4.7.1.229 #define MPU6050_RA_I2C_SLV1_ADDR 0x28

Definition at line 83 of file ArduinoGyroscopeMPU6050.h.

4.7.1.230 #define MPU6050_RA_I2C_SLV1_CTRL 0x2A

Definition at line 85 of file ArduinoGyroscopeMPU6050.h.

4.7.1.231 #define MPU6050_RA_I2C_SLV1_DO 0x64

Definition at line 142 of file ArduinoGyroscopeMPU6050.h.

4.7.1.232 #define MPU6050_RA_I2C_SLV1_REG 0x29

Definition at line 84 of file ArduinoGyroscopeMPU6050.h.

4.7.1.233 #define MPU6050_RA_I2C_SLV2_ADDR 0x2B

Definition at line 86 of file ArduinoGyroscopeMPU6050.h.

4.7.1.234 #define MPU6050_RA_I2C_SLV2_CTRL 0x2D

Definition at line 88 of file ArduinoGyroscopeMPU6050.h.

4.7.1.235 #define MPU6050_RA_I2C_SLV2_DO 0x65

Definition at line 143 of file ArduinoGyroscopeMPU6050.h.

4.7.1.236 #define MPU6050_RA_I2C_SLV2_REG 0x2C

Definition at line 87 of file ArduinoGyroscopeMPU6050.h.

4.7.1.237 #define MPU6050_RA_I2C_SLV3_ADDR 0x2E

Definition at line 89 of file ArduinoGyroscopeMPU6050.h.

4.7.1.238 #define MPU6050_RA_I2C_SLV3_CTRL 0x30

Definition at line 91 of file ArduinoGyroscopeMPU6050.h.

4.7.1.239 #define MPU6050_RA_I2C_SLV3_DO 0x66

Definition at line 144 of file ArduinoGyroscopeMPU6050.h.

4.7.1.240 #define MPU6050_RA_I2C_SLV3_REG 0x2F

Definition at line 90 of file ArduinoGyroscopeMPU6050.h.

4.7.1.241 #define MPU6050_RA_I2C_SLV4_ADDR 0x31

Definition at line 92 of file ArduinoGyroscopeMPU6050.h.

4.7.1.242 #define MPU6050_RA_I2C_SLV4_CTRL 0x34

Definition at line 95 of file ArduinoGyroscopeMPU6050.h.

4.7.1.243 #define MPU6050_RA_I2C_SLV4_DI 0x35

Definition at line 96 of file ArduinoGyroscopeMPU6050.h.

4.7.1.244 #define MPU6050_RA_I2C_SLV4_DO 0x33

Definition at line 94 of file ArduinoGyroscopeMPU6050.h.

4.7.1.245 #define MPU6050_RA_I2C_SLV4_REG 0x32

Definition at line 93 of file ArduinoGyroscopeMPU6050.h.

4.7.1.246 #define MPU6050_RA_INT_ENABLE 0x38

Definition at line 99 of file ArduinoGyroscopeMPU6050.h.

4.7.1.247 #define MPU6050_RA_INT_PIN_CFG 0x37

Definition at line 98 of file ArduinoGyroscopeMPU6050.h.

4.7.1.248 #define MPU6050_RA_INT_STATUS 0x3A

Definition at line 101 of file ArduinoGyroscopeMPU6050.h.

4.7.1.249 #define MPU6050_RA_MEM_R_W 0x6F

Definition at line 153 of file ArduinoGyroscopeMPU6050.h.

4.7.1.250 #define MPU6050_RA_MEM_START_ADDR 0x6E

Definition at line 152 of file ArduinoGyroscopeMPU6050.h.

4.7.1.251 #define MPU6050_RA_MOT_DETECT_CTRL 0x69

Definition at line 147 of file ArduinoGyroscopeMPU6050.h.

4.7.1.252 #define MPU6050_RA_MOT_DETECT_STATUS 0x61

Definition at line 140 of file ArduinoGyroscopeMPU6050.h.

4.7.1.253 #define MPU6050_RA_MOT_DUR 0x20

Definition at line 75 of file ArduinoGyroscopeMPU6050.h.

4.7.1.254 #define MPU6050_RA_MOT_THR 0x1F

Definition at line 74 of file ArduinoGyroscopeMPU6050.h.

4.7.1.255 #define MPU6050_RA_PWR_MGMT_1 0x6B

Definition at line 149 of file ArduinoGyroscopeMPU6050.h.

4.7.1.256 #define MPU6050_RA_PWR_MGMT_2 0x6C

Definition at line 150 of file ArduinoGyroscopeMPU6050.h.

4.7.1.257 #define MPU6050_RA_SIGNAL_PATH_RESET 0x68

Definition at line 146 of file ArduinoGyroscopeMPU6050.h.

4.7.1.258 #define MPU6050_RA_SMPLRT_DIV 0x19

Definition at line 68 of file ArduinoGyroscopeMPU6050.h.

4.7.1.259 #define MPU6050_RA_TEMP_OUT_H 0x41

Definition at line 108 of file ArduinoGyroscopeMPU6050.h.

4.7.1.260 #define MPU6050_RA_TEMP_OUT_L 0x42

Definition at line 109 of file ArduinoGyroscopeMPU6050.h.

4.7.1.261 #define MPU6050_RA_USER_CTRL 0x6A

Definition at line 148 of file ArduinoGyroscopeMPU6050.h.

4.7.1.262 #define MPU6050_RA_WHO_AM_I 0x75

Definition at line 159 of file ArduinoGyroscopeMPU6050.h.

4.7.1.263 #define MPU6050_RA_X_FINE_GAIN 0x03

Definition at line 53 of file ArduinoGyroscopeMPU6050.h.

4.7.1.264 #define MPU6050_RA_XA_OFFS_H 0x06

Definition at line 56 of file ArduinoGyroscopeMPU6050.h.

4.7.1.265 #define MPU6050_RA_XA_OFFS_L_TC 0x07

Definition at line 57 of file ArduinoGyroscopeMPU6050.h.

4.7.1.266 #define MPU6050_RA_XG_OFFS_TC 0x00

Definition at line 50 of file ArduinoGyroscopeMPU6050.h.

4.7.1.267 #define MPU6050_RA_XG_OFFS_USRH 0x13

Definition at line 62 of file ArduinoGyroscopeMPU6050.h.

4.7.1.268 #define MPU6050_RA_XG_OFFS_USRL 0x14

Definition at line 63 of file ArduinoGyroscopeMPU6050.h.

4.7.1.269 #define MPU6050_RA_Y_FINE_GAIN 0x04

Definition at line 54 of file ArduinoGyroscopeMPU6050.h.

4.7.1.270 #define MPU6050_RA_YA_OFFS_H 0x08

Definition at line 58 of file ArduinoGyroscopeMPU6050.h.

4.7.1.271 #define MPU6050_RA_YA_OFFS_L_TC 0x09

Definition at line 59 of file ArduinoGyroscopeMPU6050.h.

4.7.1.272 #define MPU6050_RA_YG_OFFS_TC 0x01

Definition at line 51 of file ArduinoGyroscopeMPU6050.h.

4.7.1.273 #define MPU6050_RA_YG_OFFS_USRH 0x15

Definition at line 64 of file ArduinoGyroscopeMPU6050.h.

4.7.1.274 #define MPU6050_RA_YG_OFFS_USRL 0x16

Definition at line 65 of file ArduinoGyroscopeMPU6050.h.

4.7.1.275 #define MPU6050_RA_Z_FINE_GAIN 0x05

Definition at line 55 of file ArduinoGyroscopeMPU6050.h.

4.7.1.276 #define MPU6050_RA_ZA_OFFS_H 0x0A

Definition at line 60 of file ArduinoGyroscopeMPU6050.h.

4.7.1.277 #define MPU6050_RA_ZA_OFFS_L_TC 0x0B

Definition at line 61 of file ArduinoGyroscopeMPU6050.h.

4.7.1.278 #define MPU6050_RA_ZG_OFFS_TC 0x02

Definition at line 52 of file ArduinoGyroscopeMPU6050.h.

4.7.1.279 #define MPU6050_RA_ZG_OFFS_USRH 0x17

Definition at line 66 of file ArduinoGyroscopeMPU6050.h.

4.7.1.280 #define MPU6050_RA_ZG_OFFS_USRL 0x18

Definition at line 67 of file ArduinoGyroscopeMPU6050.h.

4.7.1.281 #define MPU6050_RA_ZRMOT_DUR 0x22

Definition at line 77 of file ArduinoGyroscopeMPU6050.h.

4.7.1.282 #define MPU6050_RA_ZRMOT_THR 0x21

Definition at line 76 of file ArduinoGyroscopeMPU6050.h.

4.7.1.283 #define MPU6050_SLV0_FIFO_EN_BIT 0

Definition at line 226 of file ArduinoGyroscopeMPU6050.h.

4.7.1.284 #define MPU6050_SLV1_FIFO_EN_BIT 1

Definition at line 225 of file ArduinoGyroscopeMPU6050.h.

4.7.1.285 #define MPU6050_SLV2_FIFO_EN_BIT 2

Definition at line 224 of file ArduinoGyroscopeMPU6050.h.

4.7.1.286 #define MPU6050_SLV_3_FIFO_EN_BIT 5

Definition at line 230 of file ArduinoGyroscopeMPU6050.h.

4.7.1.287 #define MPU6050_TC_OFFSET_BIT 6

Definition at line 162 of file ArduinoGyroscopeMPU6050.h.

4.7.1.288 #define MPU6050_TC_OFFSET_LENGTH 6

Definition at line 163 of file ArduinoGyroscopeMPU6050.h.

4.7.1.289 #define MPU6050_TC_OTP_BNK_VLD_BIT 0

Definition at line 164 of file ArduinoGyroscopeMPU6050.h.

4.7.1.290 #define MPU6050_TC_PWR_MODE_BIT 7

Definition at line 161 of file ArduinoGyroscopeMPU6050.h.

4.7.1.291 #define MPU6050_TEMP_FIFO_EN_BIT 7

Definition at line 219 of file ArduinoGyroscopeMPU6050.h.

4.7.1.292 #define MPU6050_USERCTRL_DMP_EN_BIT 7

Definition at line 350 of file ArduinoGyroscopeMPU6050.h.

4.7.1.293 #define MPU6050_USERCTRL_DMP_RESET_BIT 3

Definition at line 354 of file ArduinoGyroscopeMPU6050.h.

4.7.1.294 #define MPU6050_USERCTRL_FIFO_EN_BIT 6

Definition at line 351 of file ArduinoGyroscopeMPU6050.h.

4.7.1.295 #define MPU6050_USERCTRL_FIFO_RESET_BIT 2

Definition at line 355 of file ArduinoGyroscopeMPU6050.h.

4.7.1.296 #define MPU6050_USERCTRL_I2C_IF_DIS_BIT 4

Definition at line 353 of file ArduinoGyroscopeMPU6050.h.

4.7.1.297 #define MPU6050_USERCTRL_I2C_MST_EN_BIT 5

Definition at line 352 of file ArduinoGyroscopeMPU6050.h.

4.7.1.298 #define MPU6050_USERCTRL_I2C_MST_RESET_BIT 1

Definition at line 356 of file ArduinoGyroscopeMPU6050.h.

4.7.1.299 #define MPU6050_USERCTRL_SIG_COND_RESET_BIT 0

Definition at line 357 of file ArduinoGyroscopeMPU6050.h.

4.7.1.300 #define MPU6050_VDDIO_LEVEL_VDD 1

Definition at line 167 of file ArduinoGyroscopeMPU6050.h.

4.7.1.301 #define MPU6050_VDDIO_LEVEL_VLOGIC 0

Definition at line 166 of file ArduinoGyroscopeMPU6050.h.

4.7.1.302 #define MPU6050_WAIT_FOR_ES_BIT 6

Definition at line 229 of file ArduinoGyroscopeMPU6050.h.

4.7.1.303 #define MPU6050_WAKE_FREQ_10 0x3

Definition at line 386 of file ArduinoGyroscopeMPU6050.h.

4.7.1.304 #define MPU6050_WAKE_FREQ_1P25 0x0

Definition at line 383 of file ArduinoGyroscopeMPU6050.h.

4.7.1.305 #define MPU6050_WAKE_FREQ_2P5 0x1

Definition at line 384 of file ArduinoGyroscopeMPU6050.h.

4.7.1.306 #define MPU6050_WAKE_FREQ_5 0x2

Definition at line 385 of file ArduinoGyroscopeMPU6050.h.

4.7.1.307 #define MPU6050_WHO_AM_I_BIT 6

Definition at line 393 of file ArduinoGyroscopeMPU6050.h.

4.7.1.308 #define MPU6050_WHO_AM_I_LENGTH 6

Definition at line 394 of file ArduinoGyroscopeMPU6050.h.

4.7.1.309 #define MPU6050_XG_FIFO_EN_BIT 6

Definition at line 220 of file ArduinoGyroscopeMPU6050.h.

4.7.1.310 #define MPU6050_YG_FIFO_EN_BIT 5

Definition at line 221 of file ArduinoGyroscopeMPU6050.h.

4.7.1.311 #define MPU6050_ZG_FIFO_EN_BIT 4

Definition at line 222 of file ArduinoGyroscopeMPU6050.h.

4.8 ArduinoGyroscopeMPU6050.h

```
00001 // I2Cdev library collection - MPU6050 I2C device class
00002 // Based on InvenSense MPU-6050 register map document rev. 2.0, 5/19/2011 (RM-MPU-6000A-00)
00003 // 10/3/2011 by Jeff Rowberg <jeff@rowberg.net>
00004 // Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
00005 //
00006 // Changelog:
00007 //     ... - ongoing debug release
00008
00009 // NOTE: THIS IS ONLY A PARIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING ACTIVE
00010 // DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP THIS IN MIND IF
00011 // YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.
00012
00013 /* ============================================
00014 I2Cdev device library code is placed under the MIT license
00015 Copyright (c) 2012 Jeff Rowberg
00016
00017 Permission is hereby granted, free of charge, to any person obtaining a copy
00018 of this software and associated documentation files (the "Software"), to deal
00019 in the Software without restriction, including without limitation the rights
00020 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00021 copies of the Software, and to permit persons to whom the Software is
00022 furnished to do so, subject to the following conditions:
00023
00024 The above copyright notice and this permission notice shall be included in
00025 all copies or substantial portions of the Software.
00026
00027 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00028 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00029 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00030 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00031 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00032 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00033 THE SOFTWARE.
00034 ===============================================
00035 */
00036
00037 #ifndef _MPU6050_H_
00038 #define _MPU6050_H_
00039
00040 #include "I2Cdev.h"
00041 #include "definitions.h"
00042 #include <avr/pgmspace.h>
00043
00044
```

```
00045 // !!! Moved to config.h
00046 //#define MPU6050_ADDRESS_AD0_LOW     0x68 // address pin low (GND), default for InvenSense evaluation
       board
00047 //#define MPU6050_ADDRESS_AD0_HIGH    0x69 // address pin high (VCC)
00048 //#define MPU6050_DEFAULT_ADDRESS     MPU6050_ADDRESS_AD0_HIGH
00049
00050 #define MPU6050_RA_XG_OFFS_TC       0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
00051 #define MPU6050_RA_YG_OFFS_TC       0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
00052 #define MPU6050_RA_ZG_OFFS_TC       0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
00053 #define MPU6050_RA_X_FINE_GAIN      0x03 //[7:0] X_FINE_GAIN
00054 #define MPU6050_RA_Y_FINE_GAIN      0x04 //[7:0] Y_FINE_GAIN
00055 #define MPU6050_RA_Z_FINE_GAIN      0x05 //[7:0] Z_FINE_GAIN
00056 #define MPU6050_RA_XA_OFFS_H        0x06 //[15:0] XA_OFFS
00057 #define MPU6050_RA_XA_OFFS_L_TC     0x07
00058 #define MPU6050_RA_YA_OFFS_H        0x08 //[15:0] YA_OFFS
00059 #define MPU6050_RA_YA_OFFS_L_TC     0x09
00060 #define MPU6050_RA_ZA_OFFS_H        0x0A //[15:0] ZA_OFFS
00061 #define MPU6050_RA_ZA_OFFS_L_TC     0x0B
00062 #define MPU6050_RA_XG_OFFS_USRH     0x13 //[15:0] XG_OFFS_USR
00063 #define MPU6050_RA_XG_OFFS_USRL     0x14
00064 #define MPU6050_RA_YG_OFFS_USRH     0x15 //[15:0] YG_OFFS_USR
00065 #define MPU6050_RA_YG_OFFS_USRL     0x16
00066 #define MPU6050_RA_ZG_OFFS_USRH     0x17 //[15:0] ZG_OFFS_USR
00067 #define MPU6050_RA_ZG_OFFS_USRL     0x18
00068 #define MPU6050_RA_SMPLRT_DIV       0x19
00069 #define MPU6050_RA_CONFIG           0x1A
00070 #define MPU6050_RA_GYRO_CONFIG      0x1B
00071 #define MPU6050_RA_ACCEL_CONFIG     0x1C
00072 #define MPU6050_RA_FF_THR           0x1D
00073 #define MPU6050_RA_FF_DUR           0x1E
00074 #define MPU6050_RA_MOT_THR          0x1F
00075 #define MPU6050_RA_MOT_DUR          0x20
00076 #define MPU6050_RA_ZRMOT_THR        0x21
00077 #define MPU6050_RA_ZRMOT_DUR        0x22
00078 #define MPU6050_RA_FIFO_EN          0x23
00079 #define MPU6050_RA_I2C_MST_CTRL     0x24
00080 #define MPU6050_RA_I2C_SLV0_ADDR    0x25
00081 #define MPU6050_RA_I2C_SLV0_REG     0x26
00082 #define MPU6050_RA_I2C_SLV0_CTRL    0x27
00083 #define MPU6050_RA_I2C_SLV1_ADDR    0x28
00084 #define MPU6050_RA_I2C_SLV1_REG     0x29
00085 #define MPU6050_RA_I2C_SLV1_CTRL    0x2A
00086 #define MPU6050_RA_I2C_SLV2_ADDR    0x2B
00087 #define MPU6050_RA_I2C_SLV2_REG     0x2C
00088 #define MPU6050_RA_I2C_SLV2_CTRL    0x2D
00089 #define MPU6050_RA_I2C_SLV3_ADDR    0x2E
00090 #define MPU6050_RA_I2C_SLV3_REG     0x2F
00091 #define MPU6050_RA_I2C_SLV3_CTRL    0x30
00092 #define MPU6050_RA_I2C_SLV4_ADDR    0x31
00093 #define MPU6050_RA_I2C_SLV4_REG     0x32
00094 #define MPU6050_RA_I2C_SLV4_DO      0x33
00095 #define MPU6050_RA_I2C_SLV4_CTRL    0x34
00096 #define MPU6050_RA_I2C_SLV4_DI      0x35
00097 #define MPU6050_RA_I2C_MST_STATUS   0x36
00098 #define MPU6050_RA_INT_PIN_CFG      0x37
00099 #define MPU6050_RA_INT_ENABLE       0x38
00100 #define MPU6050_RA_DMP_INT_STATUS   0x39
00101 #define MPU6050_RA_INT_STATUS       0x3A
00102 #define MPU6050_RA_ACCEL_XOUT_H     0x3B
00103 #define MPU6050_RA_ACCEL_XOUT_L     0x3C
00104 #define MPU6050_RA_ACCEL_YOUT_H     0x3D
00105 #define MPU6050_RA_ACCEL_YOUT_L     0x3E
00106 #define MPU6050_RA_ACCEL_ZOUT_H     0x3F
00107 #define MPU6050_RA_ACCEL_ZOUT_L     0x40
00108 #define MPU6050_RA_TEMP_OUT_H       0x41
00109 #define MPU6050_RA_TEMP_OUT_L       0x42
00110 #define MPU6050_RA_GYRO_XOUT_H      0x43
00111 #define MPU6050_RA_GYRO_XOUT_L      0x44
00112 #define MPU6050_RA_GYRO_YOUT_H      0x45
00113 #define MPU6050_RA_GYRO_YOUT_L      0x46
00114 #define MPU6050_RA_GYRO_ZOUT_H      0x47
00115 #define MPU6050_RA_GYRO_ZOUT_L      0x48
00116 #define MPU6050_RA_EXT_SENS_DATA_00 0x49
00117 #define MPU6050_RA_EXT_SENS_DATA_01 0x4A
00118 #define MPU6050_RA_EXT_SENS_DATA_02 0x4B
00119 #define MPU6050_RA_EXT_SENS_DATA_03 0x4C
00120 #define MPU6050_RA_EXT_SENS_DATA_04 0x4D
00121 #define MPU6050_RA_EXT_SENS_DATA_05 0x4E
00122 #define MPU6050_RA_EXT_SENS_DATA_06 0x4F
00123 #define MPU6050_RA_EXT_SENS_DATA_07 0x50
00124 #define MPU6050_RA_EXT_SENS_DATA_08 0x51
00125 #define MPU6050_RA_EXT_SENS_DATA_09 0x52
00126 #define MPU6050_RA_EXT_SENS_DATA_10 0x53
00127 #define MPU6050_RA_EXT_SENS_DATA_11 0x54
00128 #define MPU6050_RA_EXT_SENS_DATA_12 0x55
00129 #define MPU6050_RA_EXT_SENS_DATA_13 0x56
00130 #define MPU6050_RA_EXT_SENS_DATA_14 0x57
```

```
00131 #define MPU6050_RA_EXT_SENS_DATA_15 0x58
00132 #define MPU6050_RA_EXT_SENS_DATA_16 0x59
00133 #define MPU6050_RA_EXT_SENS_DATA_17 0x5A
00134 #define MPU6050_RA_EXT_SENS_DATA_18 0x5B
00135 #define MPU6050_RA_EXT_SENS_DATA_19 0x5C
00136 #define MPU6050_RA_EXT_SENS_DATA_20 0x5D
00137 #define MPU6050_RA_EXT_SENS_DATA_21 0x5E
00138 #define MPU6050_RA_EXT_SENS_DATA_22 0x5F
00139 #define MPU6050_RA_EXT_SENS_DATA_23 0x60
00140 #define MPU6050_RA_MOT_DETECT_STATUS    0x61
00141 #define MPU6050_RA_I2C_SLV0_DO      0x63
00142 #define MPU6050_RA_I2C_SLV1_DO      0x64
00143 #define MPU6050_RA_I2C_SLV2_DO      0x65
00144 #define MPU6050_RA_I2C_SLV3_DO      0x66
00145 #define MPU6050_RA_I2C_MST_DELAY_CTRL   0x67
00146 #define MPU6050_RA_SIGNAL_PATH_RESET    0x68
00147 #define MPU6050_RA_MOT_DETECT_CTRL      0x69
00148 #define MPU6050_RA_USER_CTRL        0x6A
00149 #define MPU6050_RA_PWR_MGMT_1       0x6B
00150 #define MPU6050_RA_PWR_MGMT_2       0x6C
00151 #define MPU6050_RA_BANK_SEL         0x6D
00152 #define MPU6050_RA_MEM_START_ADDR   0x6E
00153 #define MPU6050_RA_MEM_R_W          0x6F
00154 #define MPU6050_RA_DMP_CFG_1        0x70
00155 #define MPU6050_RA_DMP_CFG_2        0x71
00156 #define MPU6050_RA_FIFO_COUNTH      0x72
00157 #define MPU6050_RA_FIFO_COUNTL      0x73
00158 #define MPU6050_RA_FIFO_R_W         0x74
00159 #define MPU6050_RA_WHO_AM_I         0x75
00160
00161 #define MPU6050_TC_PWR_MODE_BIT     7
00162 #define MPU6050_TC_OFFSET_BIT       6
00163 #define MPU6050_TC_OFFSET_LENGTH    6
00164 #define MPU6050_TC_OTP_BNK_VLD_BIT  0
00165
00166 #define MPU6050_VDDIO_LEVEL_VLOGIC  0
00167 #define MPU6050_VDDIO_LEVEL_VDD     1
00168
00169 #define MPU6050_CFG_EXT_SYNC_SET_BIT    5
00170 #define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
00171 #define MPU6050_CFG_DLPF_CFG_BIT    2
00172 #define MPU6050_CFG_DLPF_CFG_LENGTH 3
00173
00174 #define MPU6050_EXT_SYNC_DISABLED       0x0
00175 #define MPU6050_EXT_SYNC_TEMP_OUT_L     0x1
00176 #define MPU6050_EXT_SYNC_GYRO_XOUT_L    0x2
00177 #define MPU6050_EXT_SYNC_GYRO_YOUT_L    0x3
00178 #define MPU6050_EXT_SYNC_GYRO_ZOUT_L    0x4
00179 #define MPU6050_EXT_SYNC_ACCEL_XOUT_L   0x5
00180 #define MPU6050_EXT_SYNC_ACCEL_YOUT_L   0x6
00181 #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L   0x7
00182
00183 #define MPU6050_DLPF_BW_256         0x00
00184 #define MPU6050_DLPF_BW_188         0x01
00185 #define MPU6050_DLPF_BW_98          0x02
00186 #define MPU6050_DLPF_BW_42          0x03
00187 #define MPU6050_DLPF_BW_20          0x04
00188 #define MPU6050_DLPF_BW_10          0x05
00189 #define MPU6050_DLPF_BW_5           0x06
00190
00191 #define MPU6050_GCONFIG_FS_SEL_BIT      4
00192 #define MPU6050_GCONFIG_FS_SEL_LENGTH   2
00193
00194 #define MPU6050_GYRO_FS_250         0x00
00195 #define MPU6050_GYRO_FS_500         0x01
00196 #define MPU6050_GYRO_FS_1000        0x02
00197 #define MPU6050_GYRO_FS_2000        0x03
00198
00199 #define MPU6050_ACONFIG_XA_ST_BIT           7
00200 #define MPU6050_ACONFIG_YA_ST_BIT           6
00201 #define MPU6050_ACONFIG_ZA_ST_BIT           5
00202 #define MPU6050_ACONFIG_AFS_SEL_BIT         4
00203 #define MPU6050_ACONFIG_AFS_SEL_LENGTH      2
00204 #define MPU6050_ACONFIG_ACCEL_HPF_BIT       2
00205 #define MPU6050_ACONFIG_ACCEL_HPF_LENGTH    3
00206
00207 #define MPU6050_ACCEL_FS_2          0x00
00208 #define MPU6050_ACCEL_FS_4          0x01
00209 #define MPU6050_ACCEL_FS_8          0x02
00210 #define MPU6050_ACCEL_FS_16         0x03
00211
00212 #define MPU6050_DHPF_RESET          0x00
00213 #define MPU6050_DHPF_5              0x01
00214 #define MPU6050_DHPF_2P5            0x02
00215 #define MPU6050_DHPF_1P25           0x03
00216 #define MPU6050_DHPF_0P63           0x04
00217 #define MPU6050_DHPF_HOLD           0x07
```

```
00218
00219 #define MPU6050_TEMP_FIFO_EN_BIT    7
00220 #define MPU6050_XG_FIFO_EN_BIT      6
00221 #define MPU6050_YG_FIFO_EN_BIT      5
00222 #define MPU6050_ZG_FIFO_EN_BIT      4
00223 #define MPU6050_ACCEL_FIFO_EN_BIT   3
00224 #define MPU6050_SLV2_FIFO_EN_BIT    2
00225 #define MPU6050_SLV1_FIFO_EN_BIT    1
00226 #define MPU6050_SLV0_FIFO_EN_BIT    0
00227
00228 #define MPU6050_MULT_MST_EN_BIT     7
00229 #define MPU6050_WAIT_FOR_ES_BIT     6
00230 #define MPU6050_SLV_3_FIFO_EN_BIT   5
00231 #define MPU6050_I2C_MST_P_NSR_BIT   4
00232 #define MPU6050_I2C_MST_CLK_BIT     3
00233 #define MPU6050_I2C_MST_CLK_LENGTH  4
00234
00235 #define MPU6050_CLOCK_DIV_348       0x0
00236 #define MPU6050_CLOCK_DIV_333       0x1
00237 #define MPU6050_CLOCK_DIV_320       0x2
00238 #define MPU6050_CLOCK_DIV_308       0x3
00239 #define MPU6050_CLOCK_DIV_296       0x4
00240 #define MPU6050_CLOCK_DIV_286       0x5
00241 #define MPU6050_CLOCK_DIV_276       0x6
00242 #define MPU6050_CLOCK_DIV_267       0x7
00243 #define MPU6050_CLOCK_DIV_258       0x8
00244 #define MPU6050_CLOCK_DIV_500       0x9
00245 #define MPU6050_CLOCK_DIV_471       0xA
00246 #define MPU6050_CLOCK_DIV_444       0xB
00247 #define MPU6050_CLOCK_DIV_421       0xC
00248 #define MPU6050_CLOCK_DIV_400       0xD
00249 #define MPU6050_CLOCK_DIV_381       0xE
00250 #define MPU6050_CLOCK_DIV_364       0xF
00251
00252 #define MPU6050_I2C_SLV_RW_BIT      7
00253 #define MPU6050_I2C_SLV_ADDR_BIT    6
00254 #define MPU6050_I2C_SLV_ADDR_LENGTH 7
00255 #define MPU6050_I2C_SLV_EN_BIT      7
00256 #define MPU6050_I2C_SLV_BYTE_SW_BIT 6
00257 #define MPU6050_I2C_SLV_REG_DIS_BIT 5
00258 #define MPU6050_I2C_SLV_GRP_BIT     4
00259 #define MPU6050_I2C_SLV_LEN_BIT     3
00260 #define MPU6050_I2C_SLV_LEN_LENGTH  4
00261
00262 #define MPU6050_I2C_SLV4_RW_BIT         7
00263 #define MPU6050_I2C_SLV4_ADDR_BIT       6
00264 #define MPU6050_I2C_SLV4_ADDR_LENGTH    7
00265 #define MPU6050_I2C_SLV4_EN_BIT         7
00266 #define MPU6050_I2C_SLV4_INT_EN_BIT     6
00267 #define MPU6050_I2C_SLV4_REG_DIS_BIT    5
00268 #define MPU6050_I2C_SLV4_MST_DLY_BIT    4
00269 #define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5
00270
00271 #define MPU6050_MST_PASS_THROUGH_BIT    7
00272 #define MPU6050_MST_I2C_SLV4_DONE_BIT   6
00273 #define MPU6050_MST_I2C_LOST_ARB_BIT    5
00274 #define MPU6050_MST_I2C_SLV4_NACK_BIT   4
00275 #define MPU6050_MST_I2C_SLV3_NACK_BIT   3
00276 #define MPU6050_MST_I2C_SLV2_NACK_BIT   2
00277 #define MPU6050_MST_I2C_SLV1_NACK_BIT   1
00278 #define MPU6050_MST_I2C_SLV0_NACK_BIT   0
00279
00280 #define MPU6050_INTCFG_INT_LEVEL_BIT        7
00281 #define MPU6050_INTCFG_INT_OPEN_BIT         6
00282 #define MPU6050_INTCFG_LATCH_INT_EN_BIT     5
00283 #define MPU6050_INTCFG_INT_RD_CLEAR_BIT     4
00284 #define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT  3
00285 #define MPU6050_INTCFG_FSYNC_INT_EN_BIT     2
00286 #define MPU6050_INTCFG_I2C_BYPASS_EN_BIT    1
00287 #define MPU6050_INTCFG_CLKOUT_EN_BIT        0
00288
00289 #define MPU6050_INTMODE_ACTIVEHIGH  0x00
00290 #define MPU6050_INTMODE_ACTIVELOW   0x01
00291
00292 #define MPU6050_INTDRV_PUSHPULL     0x00
00293 #define MPU6050_INTDRV_OPENDRAIN    0x01
00294
00295 #define MPU6050_INTLATCH_50USPULSE  0x00
00296 #define MPU6050_INTLATCH_WAITCLEAR  0x01
00297
00298 #define MPU6050_INTCLEAR_STATUSREAD 0x00
00299 #define MPU6050_INTCLEAR_ANYREAD    0x01
00300
00301 #define MPU6050_INTERRUPT_FF_BIT            7
00302 #define MPU6050_INTERRUPT_MOT_BIT           6
00303 #define MPU6050_INTERRUPT_ZMOT_BIT          5
00304 #define MPU6050_INTERRUPT_FIFO_OFLOW_BIT    4
```

```
00305 #define MPU6050_INTERRUPT_I2C_MST_INT_BIT   3
00306 #define MPU6050_INTERRUPT_PLL_RDY_INT_BIT   2
00307 #define MPU6050_INTERRUPT_DMP_INT_BIT       1
00308 #define MPU6050_INTERRUPT_DATA_RDY_BIT      0
00309
00310 // TODO: figure out what these actually do
00311 // UMPL source code is not very obivous
00312 #define MPU6050_DMPINT_5_BIT            5
00313 #define MPU6050_DMPINT_4_BIT            4
00314 #define MPU6050_DMPINT_3_BIT            3
00315 #define MPU6050_DMPINT_2_BIT            2
00316 #define MPU6050_DMPINT_1_BIT            1
00317 #define MPU6050_DMPINT_0_BIT            0
00318
00319 #define MPU6050_MOTION_MOT_XNEG_BIT     7
00320 #define MPU6050_MOTION_MOT_XPOS_BIT     6
00321 #define MPU6050_MOTION_MOT_YNEG_BIT     5
00322 #define MPU6050_MOTION_MOT_YPOS_BIT     4
00323 #define MPU6050_MOTION_MOT_ZNEG_BIT     3
00324 #define MPU6050_MOTION_MOT_ZPOS_BIT     2
00325 #define MPU6050_MOTION_MOT_ZRMOT_BIT    0
00326
00327 #define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT   7
00328 #define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT   4
00329 #define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT   3
00330 #define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT   2
00331 #define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT   1
00332 #define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT   0
00333
00334 #define MPU6050_PATHRESET_GYRO_RESET_BIT    2
00335 #define MPU6050_PATHRESET_ACCEL_RESET_BIT   1
00336 #define MPU6050_PATHRESET_TEMP_RESET_BIT    0
00337
00338 #define MPU6050_DETECT_ACCEL_ON_DELAY_BIT       5
00339 #define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH    2
00340 #define MPU6050_DETECT_FF_COUNT_BIT             3
00341 #define MPU6050_DETECT_FF_COUNT_LENGTH          2
00342 #define MPU6050_DETECT_MOT_COUNT_BIT            1
00343 #define MPU6050_DETECT_MOT_COUNT_LENGTH         2
00344
00345 #define MPU6050_DETECT_DECREMENT_RESET  0x0
00346 #define MPU6050_DETECT_DECREMENT_1      0x1
00347 #define MPU6050_DETECT_DECREMENT_2      0x2
00348 #define MPU6050_DETECT_DECREMENT_4      0x3
00349
00350 #define MPU6050_USERCTRL_DMP_EN_BIT             7
00351 #define MPU6050_USERCTRL_FIFO_EN_BIT            6
00352 #define MPU6050_USERCTRL_I2C_MST_EN_BIT         5
00353 #define MPU6050_USERCTRL_I2C_IF_DIS_BIT         4
00354 #define MPU6050_USERCTRL_DMP_RESET_BIT          3
00355 #define MPU6050_USERCTRL_FIFO_RESET_BIT         2
00356 #define MPU6050_USERCTRL_I2C_MST_RESET_BIT      1
00357 #define MPU6050_USERCTRL_SIG_COND_RESET_BIT     0
00358
00359 #define MPU6050_PWR1_DEVICE_RESET_BIT   7
00360 #define MPU6050_PWR1_SLEEP_BIT          6
00361 #define MPU6050_PWR1_CYCLE_BIT          5
00362 #define MPU6050_PWR1_TEMP_DIS_BIT       3
00363 #define MPU6050_PWR1_CLKSEL_BIT         2
00364 #define MPU6050_PWR1_CLKSEL_LENGTH      3
00365
00366 #define MPU6050_CLOCK_INTERNAL          0x00
00367 #define MPU6050_CLOCK_PLL_XGYRO         0x01
00368 #define MPU6050_CLOCK_PLL_YGYRO         0x02
00369 #define MPU6050_CLOCK_PLL_ZGYRO         0x03
00370 #define MPU6050_CLOCK_PLL_EXT32K        0x04
00371 #define MPU6050_CLOCK_PLL_EXT19M        0x05
00372 #define MPU6050_CLOCK_KEEP_RESET        0x07
00373
00374 #define MPU6050_PWR2_LP_WAKE_CTRL_BIT       7
00375 #define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH    2
00376 #define MPU6050_PWR2_STBY_XA_BIT            5
00377 #define MPU6050_PWR2_STBY_YA_BIT            4
00378 #define MPU6050_PWR2_STBY_ZA_BIT            3
00379 #define MPU6050_PWR2_STBY_XG_BIT            2
00380 #define MPU6050_PWR2_STBY_YG_BIT            1
00381 #define MPU6050_PWR2_STBY_ZG_BIT            0
00382
00383 #define MPU6050_WAKE_FREQ_1P25      0x0
00384 #define MPU6050_WAKE_FREQ_2P5       0x1
00385 #define MPU6050_WAKE_FREQ_5         0x2
00386 #define MPU6050_WAKE_FREQ_10        0x3
00387
00388 #define MPU6050_BANKSEL_PRFTCH_EN_BIT       6
00389 #define MPU6050_BANKSEL_CFG_USER_BANK_BIT   5
00390 #define MPU6050_BANKSEL_MEM_SEL_BIT         4
00391 #define MPU6050_BANKSEL_MEM_SEL_LENGTH      5
```

```
00392
00393 #define MPU6050_WHO_AM_I_BIT        6
00394 #define MPU6050_WHO_AM_I_LENGTH     6
00395
00396 #define MPU6050_DMP_MEMORY_BANKS        8
00397 #define MPU6050_DMP_MEMORY_BANK_SIZE    256
00398 #define MPU6050_DMP_MEMORY_CHUNK_SIZE   16
00399
00400 // note: DMP code memory blocks defined at end of header file
00401
00402 class MPU6050 {
00403     public:
00404         MPU6050();
00405         MPU6050(uint8_t address);
00406
00407         void initialize();
00408         bool testConnection();
00409
00410         // AUX_VDDIO register
00411         uint8_t getAuxVDDIOLevel();
00412         void setAuxVDDIOLevel(uint8_t level);
00413
00414         // SMPLRT_DIV register
00415         uint8_t getRate();
00416         void setRate(uint8_t rate);
00417
00418         // CONFIG register
00419         uint8_t getExternalFrameSync();
00420         void setExternalFrameSync(uint8_t sync);
00421         uint8_t getDLPFMode();
00422         void setDLPFMode(uint8_t bandwidth);
00423
00424         // GYRO_CONFIG register
00425         uint8_t getFullScaleGyroRange();
00426         void setFullScaleGyroRange(uint8_t range);
00427
00428         // ACCEL_CONFIG register
00429         bool getAccelXSelfTest();
00430         void setAccelXSelfTest(bool enabled);
00431         bool getAccelYSelfTest();
00432         void setAccelYSelfTest(bool enabled);
00433         bool getAccelZSelfTest();
00434         void setAccelZSelfTest(bool enabled);
00435         uint8_t getFullScaleAccelRange();
00436         void setFullScaleAccelRange(uint8_t range);
00437         uint8_t getDHPFMode();
00438         void setDHPFMode(uint8_t mode);
00439
00440         // FF_THR register
00441         uint8_t getFreefallDetectionThreshold();
00442         void setFreefallDetectionThreshold(uint8_t threshold);
00443
00444         // FF_DUR register
00445         uint8_t getFreefallDetectionDuration();
00446         void setFreefallDetectionDuration(uint8_t duration);
00447
00448         // MOT_THR register
00449         uint8_t getMotionDetectionThreshold();
00450         void setMotionDetectionThreshold(uint8_t threshold);
00451
00452         // MOT_DUR register
00453         uint8_t getMotionDetectionDuration();
00454         void setMotionDetectionDuration(uint8_t duration);
00455
00456         // ZRMOT_THR register
00457         uint8_t getZeroMotionDetectionThreshold();
00458         void setZeroMotionDetectionThreshold(uint8_t threshold);
00459
00460         // ZRMOT_DUR register
00461         uint8_t getZeroMotionDetectionDuration();
00462         void setZeroMotionDetectionDuration(uint8_t duration);
00463
00464         // FIFO_EN register
00465         bool getTempFIFOEnabled();
00466         void setTempFIFOEnabled(bool enabled);
00467         bool getXGyroFIFOEnabled();
00468         void setXGyroFIFOEnabled(bool enabled);
00469         bool getYGyroFIFOEnabled();
00470         void setYGyroFIFOEnabled(bool enabled);
00471         bool getZGyroFIFOEnabled();
00472         void setZGyroFIFOEnabled(bool enabled);
00473         bool getAccelFIFOEnabled();
00474         void setAccelFIFOEnabled(bool enabled);
00475         bool getSlave2FIFOEnabled();
00476         void setSlave2FIFOEnabled(bool enabled);
00477         bool getSlave1FIFOEnabled();
00478         void setSlave1FIFOEnabled(bool enabled);
```

```
00479         bool getSlave0FIFOEnabled();
00480         void setSlave0FIFOEnabled(bool enabled);
00481
00482         // I2C_MST_CTRL register
00483         bool getMultiMasterEnabled();
00484         void setMultiMasterEnabled(bool enabled);
00485         bool getWaitForExternalSensorEnabled();
00486         void setWaitForExternalSensorEnabled(bool enabled);
00487         bool getSlave3FIFOEnabled();
00488         void setSlave3FIFOEnabled(bool enabled);
00489         bool getSlaveReadWriteTransitionEnabled();
00490         void setSlaveReadWriteTransitionEnabled(bool enabled);
00491         uint8_t getMasterClockSpeed();
00492         void setMasterClockSpeed(uint8_t speed);
00493
00494         // I2C_SLV* registers (Slave 0-3)
00495         uint8_t getSlaveAddress(uint8_t num);
00496         void setSlaveAddress(uint8_t num, uint8_t address);
00497         uint8_t getSlaveRegister(uint8_t num);
00498         void setSlaveRegister(uint8_t num, uint8_t reg);
00499         bool getSlaveEnabled(uint8_t num);
00500         void setSlaveEnabled(uint8_t num, bool enabled);
00501         bool getSlaveWordByteSwap(uint8_t num);
00502         void setSlaveWordByteSwap(uint8_t num, bool enabled);
00503         bool getSlaveWriteMode(uint8_t num);
00504         void setSlaveWriteMode(uint8_t num, bool mode);
00505         bool getSlaveWordGroupOffset(uint8_t num);
00506         void setSlaveWordGroupOffset(uint8_t num, bool enabled);
00507         uint8_t getSlaveDataLength(uint8_t num);
00508         void setSlaveDataLength(uint8_t num, uint8_t length);
00509
00510         // I2C_SLV* registers (Slave 4)
00511         uint8_t getSlave4Address();
00512         void setSlave4Address(uint8_t address);
00513         uint8_t getSlave4Register();
00514         void setSlave4Register(uint8_t reg);
00515         void setSlave4OutputByte(uint8_t data);
00516         bool getSlave4Enabled();
00517         void setSlave4Enabled(bool enabled);
00518         bool getSlave4InterruptEnabled();
00519         void setSlave4InterruptEnabled(bool enabled);
00520         bool getSlave4WriteMode();
00521         void setSlave4WriteMode(bool mode);
00522         uint8_t getSlave4MasterDelay();
00523         void setSlave4MasterDelay(uint8_t delay);
00524         uint8_t getSlate4InputByte();
00525
00526         // I2C_MST_STATUS register
00527         bool getPassthroughStatus();
00528         bool getSlave4IsDone();
00529         bool getLostArbitration();
00530         bool getSlave4Nack();
00531         bool getSlave3Nack();
00532         bool getSlave2Nack();
00533         bool getSlave1Nack();
00534         bool getSlave0Nack();
00535
00536         // INT_PIN_CFG register
00537         bool getInterruptMode();
00538         void setInterruptMode(bool mode);
00539         bool getInterruptDrive();
00540         void setInterruptDrive(bool drive);
00541         bool getInterruptLatch();
00542         void setInterruptLatch(bool latch);
00543         bool getInterruptLatchClear();
00544         void setInterruptLatchClear(bool clear);
00545         bool getFSyncInterruptLevel();
00546         void setFSyncInterruptLevel(bool level);
00547         bool getFSyncInterruptEnabled();
00548         void setFSyncInterruptEnabled(bool enabled);
00549         bool getI2CBypassEnabled();
00550         void setI2CBypassEnabled(bool enabled);
00551         bool getClockOutputEnabled();
00552         void setClockOutputEnabled(bool enabled);
00553
00554         // INT_ENABLE register
00555         uint8_t getIntEnabled();
00556         void setIntEnabled(uint8_t enabled);
00557         bool getIntFreefallEnabled();
00558         void setIntFreefallEnabled(bool enabled);
00559         bool getIntMotionEnabled();
00560         void setIntMotionEnabled(bool enabled);
00561         bool getIntZeroMotionEnabled();
00562         void setIntZeroMotionEnabled(bool enabled);
00563         bool getIntFIFOBufferOverflowEnabled();
00564         void setIntFIFOBufferOverflowEnabled(bool enabled);
00565         bool getIntI2CMasterEnabled();
```

```
00566         void setIntI2CMasterEnabled(bool enabled);
00567         bool getIntDataReadyEnabled();
00568         void setIntDataReadyEnabled(bool enabled);
00569 
00570         // INT_STATUS register
00571         uint8_t getIntStatus();
00572         bool getIntFreefallStatus();
00573         bool getIntMotionStatus();
00574         bool getIntZeroMotionStatus();
00575         bool getIntFIFOBufferOverflowStatus();
00576         bool getIntI2CMasterStatus();
00577         bool getIntDataReadyStatus();
00578 
00579         // ACCEL_*OUT_* registers
00580         void getMotion9(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy, int16_t*
      gz, int16_t* mx, int16_t* my, int16_t* mz);
00581         void getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy, int16_t*
      gz);
00582         void getAcceleration(int16_t* x, int16_t* y, int16_t* z);
00583         int16_t getAccelerationX();
00584         int16_t getAccelerationY();
00585         int16_t getAccelerationZ();
00586 
00587         // TEMP_OUT_* registers
00588         int16_t getTemperature();
00589 
00590         // GYRO_*OUT_* registers
00591         void getRotation(int16_t* x, int16_t* y, int16_t* z);
00592         void getRotationXY(int16_t* x, int16_t* y);
00593         int16_t getRotationX();
00594         int16_t getRotationY();
00595         int16_t getRotationZ();
00596 
00597         // EXT_SENS_DATA_* registers
00598         uint8_t getExternalSensorByte(int position);
00599         uint16_t getExternalSensorWord(int position);
00600         uint32_t getExternalSensorDWord(int position);
00601 
00602         // MOT_DETECT_STATUS register
00603         bool getXNegMotionDetected();
00604         bool getXPosMotionDetected();
00605         bool getYNegMotionDetected();
00606         bool getYPosMotionDetected();
00607         bool getZNegMotionDetected();
00608         bool getZPosMotionDetected();
00609         bool getZeroMotionDetected();
00610 
00611         // I2C_SLV*_DO register
00612         void setSlaveOutputByte(uint8_t num, uint8_t data);
00613 
00614         // I2C_MST_DELAY_CTRL register
00615         bool getExternalShadowDelayEnabled();
00616         void setExternalShadowDelayEnabled(bool enabled);
00617         bool getSlaveDelayEnabled(uint8_t num);
00618         void setSlaveDelayEnabled(uint8_t num, bool enabled);
00619 
00620         // SIGNAL_PATH_RESET register
00621         void resetGyroscopePath();
00622         void resetAccelerometerPath();
00623         void resetTemperaturePath();
00624 
00625         // MOT_DETECT_CTRL register
00626         uint8_t getAccelerometerPowerOnDelay();
00627         void setAccelerometerPowerOnDelay(uint8_t delay);
00628         uint8_t getFreefallDetectionCounterDecrement();
00629         void setFreefallDetectionCounterDecrement(uint8_t decrement);
00630         uint8_t getMotionDetectionCounterDecrement();
00631         void setMotionDetectionCounterDecrement(uint8_t decrement);
00632 
00633         // USER_CTRL register
00634         bool getFIFOEnabled();
00635         void setFIFOEnabled(bool enabled);
00636         bool getI2CMasterModeEnabled();
00637         void setI2CMasterModeEnabled(bool enabled);
00638         void switchSPIEnabled(bool enabled);
00639         void resetFIFO();
00640         void resetI2CMaster();
00641         void resetSensors();
00642 
00643         // PWR_MGMT_1 register
00644         void reset();
00645         bool getSleepEnabled();
00646         void setSleepEnabled(bool enabled);
00647         bool getWakeCycleEnabled();
00648         void setWakeCycleEnabled(bool enabled);
00649         bool getTempSensorEnabled();
00650         void setTempSensorEnabled(bool enabled);
```

```
00651         uint8_t getClockSource();
00652         void setClockSource(uint8_t source);
00653
00654         // PWR_MGMT_2 register
00655         uint8_t getWakeFrequency();
00656         void setWakeFrequency(uint8_t frequency);
00657         bool getStandbyXAccelEnabled();
00658         void setStandbyXAccelEnabled(bool enabled);
00659         bool getStandbyYAccelEnabled();
00660         void setStandbyYAccelEnabled(bool enabled);
00661         bool getStandbyZAccelEnabled();
00662         void setStandbyZAccelEnabled(bool enabled);
00663         bool getStandbyXGyroEnabled();
00664         void setStandbyXGyroEnabled(bool enabled);
00665         bool getStandbyYGyroEnabled();
00666         void setStandbyYGyroEnabled(bool enabled);
00667         bool getStandbyZGyroEnabled();
00668         void setStandbyZGyroEnabled(bool enabled);
00669
00670         // FIFO_COUNT_* registers
00671         uint16_t getFIFOCount();
00672
00673         // FIFO_R_W register
00674         uint8_t getFIFOByte();
00675         void setFIFOByte(uint8_t data);
00676         void getFIFOBytes(uint8_t *data, uint8_t length);
00677
00678         // WHO_AM_I register
00679         uint8_t getDeviceID();
00680         void setDeviceID(uint8_t id);
00681
00682         // ======== UNDOCUMENTED/DMP REGISTERS/METHODS ========
00683
00684         // XG_OFFS_TC register
00685         uint8_t getOTPBankValid();
00686         void setOTPBankValid(bool enabled);
00687         int8_t getXGyroOffset();
00688         void setXGyroOffset(int8_t offset);
00689
00690         // YG_OFFS_TC register
00691         int8_t getYGyroOffset();
00692         void setYGyroOffset(int8_t offset);
00693
00694         // ZG_OFFS_TC register
00695         int8_t getZGyroOffset();
00696         void setZGyroOffset(int8_t offset);
00697
00698         // X_FINE_GAIN register
00699         int8_t getXFineGain();
00700         void setXFineGain(int8_t gain);
00701
00702         // Y_FINE_GAIN register
00703         int8_t getYFineGain();
00704         void setYFineGain(int8_t gain);
00705
00706         // Z_FINE_GAIN register
00707         int8_t getZFineGain();
00708         void setZFineGain(int8_t gain);
00709
00710         // XA_OFFS_* registers
00711         int16_t getXAccelOffset();
00712         void setXAccelOffset(int16_t offset);
00713
00714         // YA_OFFS_* register
00715         int16_t getYAccelOffset();
00716         void setYAccelOffset(int16_t offset);
00717
00718         // ZA_OFFS_* register
00719         int16_t getZAccelOffset();
00720         void setZAccelOffset(int16_t offset);
00721
00722         // XG_OFFS_USR* registers
00723         int16_t getXGyroOffsetUser();
00724         void setXGyroOffsetUser(int16_t offset);
00725
00726         // YG_OFFS_USR* register
00727         int16_t getYGyroOffsetUser();
00728         void setYGyroOffsetUser(int16_t offset);
00729
00730         // ZG_OFFS_USR* register
00731         int16_t getZGyroOffsetUser();
00732         void setZGyroOffsetUser(int16_t offset);
00733
00734         // INT_ENABLE register (DMP functions)
00735         bool getIntPLLReadyEnabled();
00736         void setIntPLLReadyEnabled(bool enabled);
00737         bool getIntDMPEnabled();
```

```
00738         void setIntDMPEnabled(bool enabled);
00739
00740         // DMP_INT_STATUS
00741         bool getDMPInt5Status();
00742         bool getDMPInt4Status();
00743         bool getDMPInt3Status();
00744         bool getDMPInt2Status();
00745         bool getDMPInt1Status();
00746         bool getDMPInt0Status();
00747
00748         // INT_STATUS register (DMP functions)
00749         bool getIntPLLReadyStatus();
00750         bool getIntDMPStatus();
00751
00752         // USER_CTRL register (DMP functions)
00753         bool getDMPEnabled();
00754         void setDMPEnabled(bool enabled);
00755         void resetDMP();
00756
00757         // BANK_SEL register
00758         void setMemoryBank(uint8_t bank, bool prefetchEnabled=false, bool userBank=false);
00759
00760         // MEM_START_ADDR register
00761         void setMemoryStartAddress(uint8_t address);
00762
00763         // MEM_R_W register
00764         uint8_t readMemoryByte();
00765         void writeMemoryByte(uint8_t data);
00766         void readMemoryBlock(uint8_t *data, uint16_t dataSize, uint8_t bank=0, uint8_t
      address=0);
00767         bool writeMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank=0,
      uint8_t address=0, bool verify=true, bool useProgMem=false);
00768         bool writeProgMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank=
      0, uint8_t address=0, bool verify=true);
00769
00770         bool writeDMPConfigurationSet(const uint8_t *data, uint16_t dataSize, bool
      useProgMem=false);
00771         bool writeProgDMPConfigurationSet(const uint8_t *data, uint16_t
      dataSize);
00772
00773         // DMP_CFG_1 register
00774         uint8_t getDMPConfig1();
00775         void setDMPConfig1(uint8_t config);
00776
00777         // DMP_CFG_2 register
00778         uint8_t getDMPConfig2();
00779         void setDMPConfig2(uint8_t config);
00780
00781
00782         // special methods for MotionApps 2.0 implementation
00783         #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS20
00784             uint8_t *dmpPacketBuffer;
00785             uint16_t dmpPacketSize;
00786
00787             uint8_t dmpInitialize();
00788             bool dmpPacketAvailable();
00789
00790             uint8_t dmpSetFIFORate(uint8_t fifoRate);
00791             uint8_t dmpGetFIFORate();
00792             uint8_t dmpGetSampleStepSizeMS();
00793             uint8_t dmpGetSampleFrequency();
00794             int32_t dmpDecodeTemperature(int8_t tempReg);
00795
00796             // Register callbacks after a packet of FIFO data is processed
00797             //uint8_t dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
00798             //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func func);
00799             uint8_t dmpRunFIFORateProcesses();
00800
00801             // Setup FIFO for various output
00802             uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
00803             uint8_t dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
00804             uint8_t dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00805             uint8_t dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00806             uint8_t dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t accuracy);
00807             uint8_t dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
00808             uint8_t dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
00809             uint8_t dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
00810             uint8_t dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
00811             uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
00812             uint8_t dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00813             uint8_t dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);
00814
00815             // Get Fixed Point data from FIFO
00816             uint8_t dmpGetAccel(int32_t *data, const uint8_t* packet=0);
00817             uint8_t dmpGetAccel(int16_t *data, const uint8_t* packet=0);
00818             uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t* packet=0);
00819             uint8_t dmpGetQuaternion(int32_t *data, const uint8_t* packet=0);
```

```
00820             uint8_t dmpGetQuaternion(int16_t *data, const uint8_t* packet=0);
00821             uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t* packet=0);
00822             uint8_t dmpGet6AxisQuaternion(int32_t *data, const uint8_t* packet=0);
00823             uint8_t dmpGet6AxisQuaternion(int16_t *data, const uint8_t* packet=0);
00824             uint8_t dmpGet6AxisQuaternion(Quaternion *q, const uint8_t* packet=0);
00825             uint8_t dmpGetRelativeQuaternion(int32_t *data, const uint8_t* packet=0);
00826             uint8_t dmpGetRelativeQuaternion(int16_t *data, const uint8_t* packet=0);
00827             uint8_t dmpGetRelativeQuaternion(Quaternion *data, const uint8_t* packet=0);
00828             uint8_t dmpGetGyro(int32_t *data, const uint8_t* packet=0);
00829             uint8_t dmpGetGyro(int16_t *data, const uint8_t* packet=0);
00830             uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t* packet=0);
00831             uint8_t dmpSetLinearAccelFilterCoefficient(float coef);
00832             uint8_t dmpGetLinearAccel(int32_t *data, const uint8_t* packet=0);
00833             uint8_t dmpGetLinearAccel(int16_t *data, const uint8_t* packet=0);
00834             uint8_t dmpGetLinearAccel(VectorInt16 *v, const uint8_t* packet=0);
00835             uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw, VectorFloat *gravity);
00836             uint8_t dmpGetLinearAccelInWorld(int32_t *data, const uint8_t* packet=0);
00837             uint8_t dmpGetLinearAccelInWorld(int16_t *data, const uint8_t* packet=0);
00838             uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const uint8_t* packet=0);
00839             uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal, Quaternion *q);
00840             uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const uint8_t* packet=0);
00841             uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const uint8_t* packet=0);
00842             uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g, VectorInt16 *a, const uint8_t* packet=0);
00843             uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t* packet=0);
00844             uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t* packet=0);
00845             uint8_t dmpGetGyroSensor(VectorInt16 *v, const uint8_t* packet=0);
00846             uint8_t dmpGetControlData(int32_t *data, const uint8_t* packet=0);
00847             uint8_t dmpGetTemperature(int32_t *data, const uint8_t* packet=0);
00848             uint8_t dmpGetGravity(int32_t *data, const uint8_t* packet=0);
00849             uint8_t dmpGetGravity(int16_t *data, const uint8_t* packet=0);
00850             uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t* packet=0);
00851             uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
00852             uint8_t dmpGetUnquantizedAccel(int32_t *data, const uint8_t* packet=0);
00853             uint8_t dmpGetUnquantizedAccel(int16_t *data, const uint8_t* packet=0);
00854             uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
00855             uint8_t dmpGetQuantizedAccel(int32_t *data, const uint8_t* packet=0);
00856             uint8_t dmpGetQuantizedAccel(int16_t *data, const uint8_t* packet=0);
00857             uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
00858             uint8_t dmpGetExternalSensorData(int32_t *data, uint16_t size, const uint8_t* packet=0);
00859             uint8_t dmpGetEIS(int32_t *data, const uint8_t* packet=0);
00860
00861             uint8_t dmpGetEuler(float *data, Quaternion *q);
00862             uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat *gravity);
00863
00864             // Get Floating Point data from FIFO
00865             uint8_t dmpGetAccelFloat(float *data, const uint8_t* packet=0);
00866             uint8_t dmpGetQuaternionFloat(float *data, const uint8_t* packet=0);
00867
00868             uint8_t dmpProcessFIFOPacket(const unsigned char *dmpData);
00869             uint8_t dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t *processed=NULL);
00870
00871             uint8_t dmpSetFIFOProcessedCallback(void (*func) (void));
00872
00873             uint8_t dmpInitFIFOParam();
00874             uint8_t dmpCloseFIFO();
00875             uint8_t dmpSetGyroDataSource(uint8_t source);
00876             uint8_t dmpDecodeQuantizedAccel();
00877             uint32_t dmpGetGyroSumOfSquare();
00878             uint32_t dmpGetAccelSumOfSquare();
00879             void dmpOverrideQuaternion(long *q);
00880             uint16_t dmpGetFIFOPacketSize();
00881
00882         #endif
00883
00884         // special methods for MotionApps 4.1 implementation
00885         #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS41
00886             uint8_t *dmpPacketBuffer;
00887             uint16_t dmpPacketSize;
00888
00889             uint8_t dmpInitialize();
00890             bool dmpPacketAvailable();
00891
00892             uint8_t dmpSetFIFORate(uint8_t fifoRate);
00893             uint8_t dmpGetFIFORate();
00894             uint8_t dmpGetSampleStepSizeMS();
00895             uint8_t dmpGetSampleFrequency();
00896             int32_t dmpDecodeTemperature(int8_t tempReg);
00897
00898             // Register callbacks after a packet of FIFO data is processed
00899             //uint8_t dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
00900             //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func func);
00901             uint8_t dmpRunFIFORateProcesses();
00902
00903             // Setup FIFO for various output
00904             uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
00905             uint8_t dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
00906             uint8_t dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
```

```
00907             uint8_t dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00908             uint8_t dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t accuracy);
00909             uint8_t dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
00910             uint8_t dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
00911             uint8_t dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
00912             uint8_t dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
00913             uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
00914             uint8_t dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00915             uint8_t dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);
00916 
00917             // Get Fixed Point data from FIFO
00918             uint8_t dmpGetAccel(int32_t *data, const uint8_t* packet=0);
00919             uint8_t dmpGetAccel(int16_t *data, const uint8_t* packet=0);
00920             uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t* packet=0);
00921             uint8_t dmpGetQuaternion(int32_t *data, const uint8_t* packet=0);
00922             uint8_t dmpGetQuaternion(int16_t *data, const uint8_t* packet=0);
00923             uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t* packet=0);
00924             uint8_t dmpGet6AxisQuaternion(int32_t *data, const uint8_t* packet=0);
00925             uint8_t dmpGet6AxisQuaternion(int16_t *data, const uint8_t* packet=0);
00926             uint8_t dmpGet6AxisQuaternion(Quaternion *q, const uint8_t* packet=0);
00927             uint8_t dmpGetRelativeQuaternion(int32_t *data, const uint8_t* packet=0);
00928             uint8_t dmpGetRelativeQuaternion(int16_t *data, const uint8_t* packet=0);
00929             uint8_t dmpGetRelativeQuaternion(Quaternion *data, const uint8_t* packet=0);
00930             uint8_t dmpGetGyro(int32_t *data, const uint8_t* packet=0);
00931             uint8_t dmpGetGyro(int16_t *data, const uint8_t* packet=0);
00932             uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t* packet=0);
00933             uint8_t dmpGetMag(int16_t *data, const uint8_t* packet=0);
00934             uint8_t dmpSetLinearAccelFilterCoefficient(float coef);
00935             uint8_t dmpGetLinearAccel(int32_t *data, const uint8_t* packet=0);
00936             uint8_t dmpGetLinearAccel(int16_t *data, const uint8_t* packet=0);
00937             uint8_t dmpGetLinearAccel(VectorInt16 *v, const uint8_t* packet=0);
00938             uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw, VectorFloat *gravity);
00939             uint8_t dmpGetLinearAccelInWorld(int32_t *data, const uint8_t* packet=0);
00940             uint8_t dmpGetLinearAccelInWorld(int16_t *data, const uint8_t* packet=0);
00941             uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const uint8_t* packet=0);
00942             uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal, Quaternion *q);
00943             uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const uint8_t* packet=0);
00944             uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const uint8_t* packet=0);
00945             uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g, VectorInt16 *a, const uint8_t* packet=0);
00946             uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t* packet=0);
00947             uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t* packet=0);
00948             uint8_t dmpGetGyroSensor(VectorInt16 *v, const uint8_t* packet=0);
00949             uint8_t dmpGetControlData(int32_t *data, const uint8_t* packet=0);
00950             uint8_t dmpGetTemperature(int32_t *data, const uint8_t* packet=0);
00951             uint8_t dmpGetGravity(int32_t *data, const uint8_t* packet=0);
00952             uint8_t dmpGetGravity(int16_t *data, const uint8_t* packet=0);
00953             uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t* packet=0);
00954             uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
00955             uint8_t dmpGetUnquantizedAccel(int32_t *data, const uint8_t* packet=0);
00956             uint8_t dmpGetUnquantizedAccel(int16_t *data, const uint8_t* packet=0);
00957             uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
00958             uint8_t dmpGetQuantizedAccel(int32_t *data, const uint8_t* packet=0);
00959             uint8_t dmpGetQuantizedAccel(int16_t *data, const uint8_t* packet=0);
00960             uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
00961             uint8_t dmpGetExternalSensorData(int32_t *data, uint16_t size, const uint8_t* packet=0);
00962             uint8_t dmpGetEIS(int32_t *data, const uint8_t* packet=0);
00963 
00964             uint8_t dmpGetEuler(float *data, Quaternion *q);
00965             uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat *gravity);
00966 
00967             // Get Floating Point data from FIFO
00968             uint8_t dmpGetAccelFloat(float *data, const uint8_t* packet=0);
00969             uint8_t dmpGetQuaternionFloat(float *data, const uint8_t* packet=0);
00970 
00971             uint8_t dmpProcessFIFOPacket(const unsigned char *dmpData);
00972             uint8_t dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t *processed=NULL);
00973 
00974             uint8_t dmpSetFIFOProcessedCallback(void (*func) (void));
00975 
00976             uint8_t dmpInitFIFOParam();
00977             uint8_t dmpCloseFIFO();
00978             uint8_t dmpSetGyroDataSource(uint8_t source);
00979             uint8_t dmpDecodeQuantizedAccel();
00980             uint32_t dmpGetGyroSumOfSquare();
00981             uint32_t dmpGetAccelSumOfSquare();
00982             void dmpOverrideQuaternion(long *q);
00983             uint16_t dmpGetFIFOPacketSize();
00984         #endif
00985 
00986     private:
00987         uint8_t devAddr;
00988         uint8_t buffer[14];
00989 };
00990 
00991 #endif /* _MPU6050_H_ */
```

Index