

## Arduino Brushless Gimbal Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:51:26

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Class Index</b>  | <b>1</b> |
| 1.1      | Class List . . . . .  | 1        |
| <b>2</b> | <b>File Index</b>   | <b>1</b> |
| 2.1      | File List . . . . .   | 1        |
| <b>3</b> | <b>Class Documentation</b>                                      | <b>2</b> |
| 3.1      | I2Cdev Class Reference . . . . .                                | 2        |
| 3.1.1    | Detailed Description . . . . .                                  | 2        |
| 3.1.2    | Constructor & Destructor Documentation . . . . .                | 3        |
| 3.1.3    | Member Function Documentation . . . . .                         | 3        |
| 3.1.4    | Member Data Documentation . . . . .                             | 8        |
| 3.2      | MPU6050 Class Reference . . . . .                               | 8        |
| 3.2.1    | Detailed Description . . . . .                                  | 13       |
| 3.2.2    | Constructor & Destructor Documentation . . . . .                | 13       |
| 3.2.3    | Member Function Documentation . . . . .                         | 14       |
| 3.2.4    | Member Data Documentation . . . . .                             | 71       |
| 3.3      | PCintPort::PCintPin Class Reference . . . . .                   | 72       |
| 3.3.1    | Detailed Description . . . . .                                  | 72       |
| 3.3.2    | Constructor & Destructor Documentation . . . . .                | 72       |
| 3.3.3    | Member Data Documentation . . . . .                             | 72       |
| 3.4      | PCintPort Class Reference . . . . .                             | 73       |
| 3.4.1    | Detailed Description . . . . .                                  | 74       |
| 3.4.2    | Constructor & Destructor Documentation . . . . .                | 74       |
| 3.4.3    | Member Function Documentation . . . . .                         | 74       |
| 3.4.4    | Member Data Documentation . . . . .                             | 74       |
| 3.5      | Quaternion Class Reference . . . . .                            | 75       |
| 3.5.1    | Detailed Description . . . . .                                  | 76       |
| 3.5.2    | Constructor & Destructor Documentation . . . . .                | 76       |
| 3.5.3    | Member Function Documentation . . . . .                         | 76       |
| 3.5.4    | Member Data Documentation . . . . .                             | 76       |
| 3.6      | SerialCommand Class Reference . . . . .                         | 77       |
| 3.6.1    | Detailed Description . . . . .                                  | 77       |
| 3.6.2    | Constructor & Destructor Documentation . . . . .                | 78       |
| 3.6.3    | Member Function Documentation . . . . .                         | 78       |
| 3.6.4    | Member Data Documentation . . . . .                             | 78       |
| 3.7      | SerialCommand::SerialCommandCallback Struct Reference . . . . . | 79       |
| 3.7.1    | Detailed Description . . . . .                                  | 79       |
| 3.7.2    | Member Data Documentation . . . . .                             | 79       |

|          |  |           |
|----------|--|-----------|
| 3.8      | VectorFloat Class Reference            | 80        |
| 3.8.1    | Detailed Description                   | 80        |
| 3.8.2    | Constructor & Destructor Documentation | 80        |
| 3.8.3    | Member Function Documentation          | 80        |
| 3.8.4    | Member Data Documentation              | 81        |
| 3.9      | VectorInt16 Class Reference            | 81        |
| 3.9.1    | Detailed Description                   | 81        |
| 3.9.2    | Constructor & Destructor Documentation | 81        |
| 3.9.3    | Member Function Documentation          | 82        |
| 3.9.4    | Member Data Documentation              | 82        |
| <b>4</b> | <b>File Documentation</b>              | <b>82</b> |
| 4.1      | definitions.h File Reference           | 83        |
| 4.1.1    | Macro Definition Documentation         | 84        |
| 4.2      | definitions.h                          | 86        |
| 4.3      | EEPROMAnything.h File Reference        | 87        |
| 4.3.1    | Function Documentation                 | 87        |
| 4.4      | EEPROMAnything.h                       | 87        |
| 4.5      | helper_3dmath.h File Reference         | 88        |
| 4.6      | helper_3dmath.h                        | 88        |
| 4.7      | I2Cdev.cpp File Reference              | 91        |
| 4.8      | I2Cdev.cpp                             | 91        |
| 4.9      | I2Cdev.h File Reference                | 106       |
| 4.9.1    | Macro Definition Documentation         | 107       |
| 4.10     | I2Cdev.h                               | 107       |
| 4.11     | MPU6050.cpp File Reference             | 111       |
| 4.12     | MPU6050.cpp                            | 111       |
| 4.13     | MPU6050.h File Reference               | 130       |
| 4.13.1   | Macro Definition Documentation         | 137       |
| 4.14     | MPU6050.h                              | 157       |
| 4.15     | MPU6050_6Axis_DMP.h File Reference     | 169       |
| 4.15.1   | Macro Definition Documentation         | 170       |
| 4.15.2   | Variable Documentation                 | 170       |
| 4.16     | MPU6050_6Axis_DMP.h                    | 170       |
| 4.17     | PinChangeInt.h File Reference          | 178       |
| 4.17.1   | Macro Definition Documentation         | 179       |
| 4.17.2   | Typedef Documentation                  | 180       |
| 4.17.3   | Function Documentation                 | 180       |
| 4.17.4   | Variable Documentation                 | 180       |
| 4.18     | PinChangeInt.h                         | 181       |

|   |            |
|---|------------|
| 4.19 SerialCommand.cpp File Reference . . . . . | 189        |
| 4.20 SerialCommand.cpp . . . . .                | 189        |
| 4.21 SerialCommand.h File Reference . . . . .   | 190        |
| 4.21.1 Macro Definition Documentation . . . . . | 191        |
| 4.22 SerialCommand.h . . . . .                  | 192        |
| <b>Index</b>                                    | <b>193</b> |

## 1 Class Index

### 1.1 Class List

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

|  |                    |
|--|--------------------|
| <a href="#">I2Cdev</a>                               | <a href="#">2</a>  |
| <a href="#">MPU6050</a>                              | <a href="#">8</a>  |
| <a href="#">PCintPort::PCintPin</a>                  | <a href="#">72</a> |
| <a href="#">PCintPort</a>                            | <a href="#">73</a> |
| <a href="#">Quaternion</a>                           | <a href="#">75</a> |
| <a href="#">SerialCommand</a>                        | <a href="#">77</a> |
| <a href="#">SerialCommand::SerialCommandCallback</a> | <a href="#">79</a> |
| <a href="#">VectorFloat</a>                          | <a href="#">80</a> |
| <a href="#">VectorInt16</a>                          | <a href="#">81</a> |

## 2 File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

|                                     |                     |
|-------------------------------------|---------------------|
| <a href="#">definitions.h</a>       | <a href="#">83</a>  |
| <a href="#">EEPROMAnything.h</a>    | <a href="#">87</a>  |
| <a href="#">helper_3dmath.h</a>     | <a href="#">88</a>  |
| <a href="#">I2Cdev.cpp</a>          | <a href="#">91</a>  |
| <a href="#">I2Cdev.h</a>            | <a href="#">106</a> |
| <a href="#">MPU6050.cpp</a>         | <a href="#">111</a> |
| <a href="#">MPU6050.h</a>           | <a href="#">130</a> |
| <a href="#">MPU6050_6Axis_DMP.h</a> | <a href="#">169</a> |
| <a href="#">PinChangeInt.h</a>      | <a href="#">178</a> |

|                                   |     |
|-----------------------------------|-----|
| <a href="#">SerialCommand.cpp</a> | 189 |
| <a href="#">SerialCommand.h</a>   | 190 |

## 3 Class Documentation

### 3.1 I2Cdev Class Reference

```
#include <I2Cdev.h>
```

#### Public Member Functions

- [I2Cdev](#) ()

#### Static Public Member Functions

- static int8\_t [readBit](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitNum, uint8\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readBitW](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitNum, uint16\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readBits](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitStart, uint8\_t length, uint8\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readBitsW](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitStart, uint8\_t length, uint16\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readByte](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readWord](#) (uint8\_t devAddr, uint8\_t regAddr, uint16\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readBytes](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t length, uint8\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static int8\_t [readWords](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t length, uint16\_t \*data, uint16\_t timeout=[I2Cdev::readTimeout](#))
- static bool [writeBit](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitNum, uint8\_t data)
- static bool [writeBitW](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitNum, uint16\_t data)
- static bool [writeBits](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitStart, uint8\_t length, uint8\_t data)
- static bool [writeBitsW](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t bitStart, uint8\_t length, uint16\_t data)
- static bool [writeByte](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t data)
- static bool [writeWord](#) (uint8\_t devAddr, uint8\_t regAddr, uint16\_t data)
- static bool [writeBytes](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t length, uint8\_t \*data)
- static bool [writeWords](#) (uint8\_t devAddr, uint8\_t regAddr, uint8\_t length, uint16\_t \*data)

#### Static Public Attributes

- static uint16\_t [readTimeout](#) = [I2CDEV\\_DEFAULT\\_READ\\_TIMEOUT](#)

#### 3.1.1 Detailed Description

Definition at line 86 of file [I2Cdev.h](#).

## 3.1.2 Constructor &amp; Destructor Documentation

## 3.1.2.1 I2Cdev::I2Cdev ( )

Default constructor.

Definition at line 92 of file [I2Cdev.cpp](#).

## 3.1.3 Member Function Documentation

3.1.3.1 `int8_t I2Cdev::readBit ( uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t * data, uint16_t timeout = I2Cdev::readTimeout ) [static]`

Read a single bit from an 8-bit device register.

## Parameters

|                |  |
|----------------|--|
| <i>devAddr</i> | I2C slave device address   |
| <i>regAddr</i> | Register regAddr to read from  |
| <i>bitNum</i>  | Bit position to read (0-7)   |
| <i>data</i>    | Container for single bit value   |
| <i>timeout</i> | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

## Returns

Status of read operation (true = success)

Definition at line 103 of file [I2Cdev.cpp](#).

3.1.3.2 `int8_t I2Cdev::readBits ( uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length, uint8_t * data, uint16_t timeout = I2Cdev::readTimeout ) [static]`

Read multiple bits from an 8-bit device register.

## Parameters

|                 |  |
|-----------------|--|
| <i>devAddr</i>  | I2C slave device address   |
| <i>regAddr</i>  | Register regAddr to read from  |
| <i>bitStart</i> | First bit position to read (0-7)   |
| <i>length</i>   | Number of bits to read (not more than 8)   |
| <i>data</i>     | Container for right-aligned value (i.e. '101' read from any bitStart position will equal 0x05)                                     |
| <i>timeout</i>  | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

## Returns

Status of read operation (true = success)

Definition at line 134 of file [I2Cdev.cpp](#).

3.1.3.3 `int8_t I2Cdev::readBitsW ( uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length, uint16_t * data, uint16_t timeout = I2Cdev::readTimeout ) [static]`

Read multiple bits from a 16-bit device register.

**Parameters**

|                 |  |
|-----------------|--|
| <i>devAddr</i>  | I2C slave device address   |
| <i>regAddr</i>  | Register regAddr to read from  |
| <i>bitStart</i> | First bit position to read (0-15)  |
| <i>length</i>   | Number of bits to read (not more than 16)  |
| <i>data</i>     | Container for right-aligned value (i.e. '101' read from any bitStart position will equal 0x05)                                     |
| <i>timeout</i>  | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

**Returns**

Status of read operation (1 = success, 0 = failure, -1 = timeout)

Definition at line 159 of file [I2Cdev.cpp](#).

```
3.1.3.4 int8_t I2Cdev::readBitW ( uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t * data, uint16_t timeout =
      I2Cdev::readTimeout ) [static]
```

Read a single bit from a 16-bit device register.

**Parameters**

|                |  |
|----------------|--|
| <i>devAddr</i> | I2C slave device address   |
| <i>regAddr</i> | Register regAddr to read from  |
| <i>bitNum</i>  | Bit position to read (0-15)  |
| <i>data</i>    | Container for single bit value   |
| <i>timeout</i> | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

**Returns**

Status of read operation (true = success)

Definition at line 118 of file [I2Cdev.cpp](#).

```
3.1.3.5 int8_t I2Cdev::readByte ( uint8_t devAddr, uint8_t regAddr, uint8_t * data, uint16_t timeout = I2Cdev::readTimeout
      ) [static]
```

Read single byte from an 8-bit device register.

**Parameters**

|                |  |
|----------------|--|
| <i>devAddr</i> | I2C slave device address   |
| <i>regAddr</i> | Register regAddr to read from  |
| <i>data</i>    | Container for byte value read from device  |
| <i>timeout</i> | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

**Returns**

Status of read operation (true = success)

Definition at line 183 of file [I2Cdev.cpp](#).

```
3.1.3.6 int8_t I2Cdev::readBytes ( uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t * data, uint16_t timeout =
      I2Cdev::readTimeout ) [static]
```

Read multiple bytes from an 8-bit device register.

## Parameters

|                |  |
|----------------|--|
| <i>devAddr</i> | I2C slave device address   |
| <i>regAddr</i> | First register regAddr to read from  |
| <i>length</i>  | Number of bytes to read  |
| <i>data</i>    | Buffer to store read data in   |
| <i>timeout</i> | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

## Returns

Number of bytes read (-1 indicates failure)

Definition at line 206 of file [I2Cdev.cpp](#).

3.1.3.7 `int8_t I2Cdev::readWord ( uint8_t devAddr, uint8_t regAddr, uint16_t * data, uint16_t timeout = I2Cdev::readTimeout ) [static]`

Read single word from a 16-bit device register.

## Parameters

|                |  |
|----------------|--|
| <i>devAddr</i> | I2C slave device address   |
| <i>regAddr</i> | Register regAddr to read from  |
| <i>data</i>    | Container for word value read from device  |
| <i>timeout</i> | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

## Returns

Status of read operation (true = success)

Definition at line 194 of file [I2Cdev.cpp](#).

3.1.3.8 `int8_t I2Cdev::readWords ( uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t * data, uint16_t timeout = I2Cdev::readTimeout ) [static]`

Read multiple words from a 16-bit device register.

## Parameters

|                |  |
|----------------|--|
| <i>devAddr</i> | I2C slave device address   |
| <i>regAddr</i> | First register regAddr to read from  |
| <i>length</i>  | Number of words to read  |
| <i>data</i>    | Buffer to store read data in   |
| <i>timeout</i> | Optional read timeout in milliseconds (0 to disable, leave off to use default class value in <a href="#">I2Cdev::readTimeout</a> ) |

## Returns

Number of words read (0 indicates failure)

Definition at line 328 of file [I2Cdev.cpp](#).

3.1.3.9 `bool I2Cdev::writeBit ( uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t data ) [static]`

write a single bit in an 8-bit device register.



**Parameters**

|                |                              |
|----------------|------------------------------|
| <i>devAddr</i> | I2C slave device address     |
| <i>regAddr</i> | Register regAddr to write to |
| <i>bitNum</i>  | Bit position to write (0-7)  |
| <i>value</i>   | New bit value to write       |

**Returns**

Status of operation (true = success)

Definition at line 479 of file [I2Cdev.cpp](#).

```
3.1.3.10 bool I2Cdev::writeBits ( uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length, uint8_t data )
        [static]
```

Write multiple bits in an 8-bit device register.

**Parameters**

|                 |   |
|-----------------|---|
| <i>devAddr</i>  | I2C slave device address                  |
| <i>regAddr</i>  | Register regAddr to write to              |
| <i>bitStart</i> | First bit position to write (0-7)         |
| <i>length</i>   | Number of bits to write (not more than 8) |
| <i>data</i>     | Right-aligned value to write              |

**Returns**

Status of operation (true = success)

Definition at line 508 of file [I2Cdev.cpp](#).

```
3.1.3.11 bool I2Cdev::writeBitsW ( uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length, uint16_t data )
        [static]
```

Write multiple bits in a 16-bit device register.

**Parameters**

|                 |  |
|-----------------|--|
| <i>devAddr</i>  | I2C slave device address                   |
| <i>regAddr</i>  | Register regAddr to write to               |
| <i>bitStart</i> | First bit position to write (0-15)         |
| <i>length</i>   | Number of bits to write (not more than 16) |
| <i>data</i>     | Right-aligned value to write               |

**Returns**

Status of operation (true = success)

Definition at line 537 of file [I2Cdev.cpp](#).

```
3.1.3.12 bool I2Cdev::writeBitW ( uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t data ) [static]
```

write a single bit in a 16-bit device register.

**Parameters**

|                |                                     |
|----------------|-------------------------------------|
| <i>devAddr</i> | I2C slave device address            |
| <i>regAddr</i> | Register <i>regAddr</i> to write to |
| <i>bitNum</i>  | Bit position to write (0-15)        |
| <i>value</i>   | New bit value to write              |

**Returns**

Status of operation (true = success)

Definition at line 493 of file [I2Cdev.cpp](#).

3.1.3.13 `bool I2Cdev::writeByte ( uint8_t devAddr, uint8_t regAddr, uint8_t data ) [static]`

Write single byte to an 8-bit device register.

**Parameters**

|                |                              |
|----------------|------------------------------|
| <i>devAddr</i> | I2C slave device address     |
| <i>regAddr</i> | Register address to write to |
| <i>data</i>    | New byte value to write      |

**Returns**

Status of operation (true = success)

Definition at line 564 of file [I2Cdev.cpp](#).

3.1.3.14 `bool I2Cdev::writeBytes ( uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t* data ) [static]`

Write multiple bytes to an 8-bit device register.

**Parameters**

|                |                                    |
|----------------|------------------------------------|
| <i>devAddr</i> | I2C slave device address           |
| <i>regAddr</i> | First register address to write to |
| <i>length</i>  | Number of bytes to write           |
| <i>data</i>    | Buffer to copy new data from       |

**Returns**

Status of operation (true = success)

Definition at line 585 of file [I2Cdev.cpp](#).

3.1.3.15 `bool I2Cdev::writeWord ( uint8_t devAddr, uint8_t regAddr, uint16_t data ) [static]`

Write single word to a 16-bit device register.

**Parameters**

|                |                              |
|----------------|------------------------------|
| <i>devAddr</i> | I2C slave device address     |
| <i>regAddr</i> | Register address to write to |
| <i>data</i>    | New word value to write      |

**Returns**

Status of operation (true = success)

Definition at line 574 of file [I2Cdev.cpp](#).

3.1.3.16 `bool I2Cdev::writeWords ( uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t* data ) [static]`

Write multiple words to a 16-bit device register.

**Parameters**

|                |                                    |
|----------------|------------------------------------|
| <i>devAddr</i> | I2C slave device address           |
| <i>regAddr</i> | First register address to write to |
| <i>length</i>  | Number of words to write           |
| <i>data</i>    | Buffer to copy new data from       |

**Returns**

Status of operation (true = success)

Definition at line 635 of file [I2Cdev.cpp](#).

**3.1.4 Member Data Documentation****3.1.4.1 uint16\_t I2Cdev::readTimeout = I2CDEV\_DEFAULT\_READ\_TIMEOUT [static]**

Default timeout value for read operations.

Set this to 0 to disable timeout detection.

Definition at line 108 of file [I2Cdev.h](#).

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

- [I2Cdev.h](#)
- [I2Cdev.cpp](#)

**3.2 MPU6050 Class Reference**

```
#include <MPU6050.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 ()`
- `void setMultiMasterEnabled (bool enabled)`
- `bool getWaitForExternalSensorEnabled ()`
- `void setWaitForExternalSensorEnabled (bool enabled)`
- `bool getSlave3FIFOEnabled ()`
- `void setSlave3FIFOEnabled (bool enabled)`
- `bool getSlaveReadWriteTransitionEnabled ()`
- `void setSlaveReadWriteTransitionEnabled (bool enabled)`
- `uint8_t getMasterClockSpeed ()`
- `void setMasterClockSpeed (uint8_t speed)`
- `uint8_t getSlaveAddress (uint8_t num)`
- `void setSlaveAddress (uint8_t num, uint8_t address)`
- `uint8_t getSlaveRegister (uint8_t num)`
- `void setSlaveRegister (uint8_t num, uint8_t reg)`
- `bool getSlaveEnabled (uint8_t num)`
- `void setSlaveEnabled (uint8_t num, bool enabled)`
- `bool getSlaveWordByteSwap (uint8_t num)`
- `void setSlaveWordByteSwap (uint8_t num, bool enabled)`
- `bool getSlaveWriteMode (uint8_t num)`
- `void setSlaveWriteMode (uint8_t num, bool mode)`
- `bool getSlaveWordGroupOffset (uint8_t num)`
- `void setSlaveWordGroupOffset (uint8_t num, bool enabled)`
- `uint8_t getSlaveDataLength (uint8_t num)`
- `void setSlaveDataLength (uint8_t num, uint8_t length)`
- `uint8_t getSlave4Address ()`

- void [setSlave4Address](#) (uint8\_t address)
- uint8\_t [getSlave4Register](#) ()
- void [setSlave4Register](#) (uint8\_t reg)
- void [setSlave4OutputByte](#) (uint8\_t data)
- bool [getSlave4Enabled](#) ()
- void [setSlave4Enabled](#) (bool enabled)
- bool [getSlave4InterruptEnabled](#) ()
- void [setSlave4InterruptEnabled](#) (bool enabled)
- bool [getSlave4WriteMode](#) ()
- void [setSlave4WriteMode](#) (bool mode)
- uint8\_t [getSlave4MasterDelay](#) ()
- void [setSlave4MasterDelay](#) (uint8\_t delay)
- uint8\_t [getSlave4InputByte](#) ()
- bool [getPassthroughStatus](#) ()
- bool [getSlave4IsDone](#) ()
- bool [getLostArbitration](#) ()
- bool [getSlave4Nack](#) ()
- bool [getSlave3Nack](#) ()
- bool [getSlave2Nack](#) ()
- bool [getSlave1Nack](#) ()
- bool [getSlave0Nack](#) ()
- bool [getInterruptMode](#) ()
- void [setInterruptMode](#) (bool mode)
- bool [getInterruptDrive](#) ()
- void [setInterruptDrive](#) (bool drive)
- bool [getInterruptLatch](#) ()
- void [setInterruptLatch](#) (bool latch)
- bool [getInterruptLatchClear](#) ()
- void [setInterruptLatchClear](#) (bool clear)
- bool [getFSyncInterruptLevel](#) ()
- void [setFSyncInterruptLevel](#) (bool level)
- 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 [resetGyroscopePath](#) ()
- void [resetAccelerometerPath](#) ()
- void [resetTemperaturePath](#) ()
- uint8\_t [getAccelerometerPowerOnDelay](#) ()
- void [setAccelerometerPowerOnDelay](#) (uint8\_t delay)
- uint8\_t [getFreefallDetectionCounterDecrement](#) ()
- void [setFreefallDetectionCounterDecrement](#) (uint8\_t decrement)
- uint8\_t [getMotionDetectionCounterDecrement](#) ()
- void [setMotionDetectionCounterDecrement](#) (uint8\_t decrement)
- bool [getFIFOEnabled](#) ()
- void [setFIFOEnabled](#) (bool enabled)
- bool [getI2CMasterModeEnabled](#) ()
- void [setI2CMasterModeEnabled](#) (bool enabled)
- void [switchSPIEnabled](#) (bool enabled)
- void [resetFIFO](#) ()
- void [resetI2CMaster](#) ()
- void [resetSensors](#) ()
- void [reset](#) ()
- bool [getSleepEnabled](#) ()
- void [setSleepEnabled](#) (bool enabled)
- bool [getWakeCycleEnabled](#) ()
- void [setWakeCycleEnabled](#) (bool enabled)
- bool [getTempSensorEnabled](#) ()
- void [setTempSensorEnabled](#) (bool enabled)

- [uint8\\_t getClockSource \(\)](#)
- [void setClockSource \(uint8\\_t source\)](#)
- [uint8\\_t getWakeFrequency \(\)](#)
- [void setWakeFrequency \(uint8\\_t frequency\)](#)
- [bool getStandbyXAccelEnabled \(\)](#)
- [void setStandbyXAccelEnabled \(bool enabled\)](#)
- [bool getStandbyYAccelEnabled \(\)](#)
- [void setStandbyYAccelEnabled \(bool enabled\)](#)
- [bool getStandbyZAccelEnabled \(\)](#)
- [void setStandbyZAccelEnabled \(bool enabled\)](#)
- [bool getStandbyXGyroEnabled \(\)](#)
- [void setStandbyXGyroEnabled \(bool enabled\)](#)
- [bool getStandbyYGyroEnabled \(\)](#)
- [void setStandbyYGyroEnabled \(bool enabled\)](#)
- [bool getStandbyZGyroEnabled \(\)](#)
- [void setStandbyZGyroEnabled \(bool enabled\)](#)
- [uint16\\_t getFIFOCount \(\)](#)
- [uint8\\_t getFIFOByte \(\)](#)
- [void setFIFOByte \(uint8\\_t data\)](#)
- [void getFIFOBytes \(uint8\\_t \\*data, uint8\\_t length\)](#)
- [uint8\\_t getDeviceID \(\)](#)
- [void setDeviceID \(uint8\\_t id\)](#)
- [uint8\\_t getOTPBANKValid \(\)](#)
- [void setOTPBANKValid \(bool enabled\)](#)
- [int8\\_t getXGyroOffset \(\)](#)
- [void setXGyroOffset \(int8\\_t offset\)](#)
- [int8\\_t getYGyroOffset \(\)](#)
- [void setYGyroOffset \(int8\\_t offset\)](#)
- [int8\\_t getZGyroOffset \(\)](#)
- [void setZGyroOffset \(int8\\_t offset\)](#)
- [int8\\_t getXFineGain \(\)](#)
- [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 [MPU6050.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 [MPU6050.cpp](#).

##### 3.2.2.2 MPU6050::MPU6050 ( uint8\_t address )

Specific address constructor.

Parameters

|                |             |
|----------------|-------------|
| <i>address</i> | I2C address |
|----------------|-------------|



See also

[MPU6050\\_DEFAULT\\_ADDRESS](#)  
[MPU6050\\_ADDRESS\\_AD0\\_LOW](#)  
[MPU6050\\_ADDRESS\\_AD0\\_HIGH](#)

Definition at line 52 of file [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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`, `ACCEL_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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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.

| DLPF_CFG | ACCELEROMETER  |        | GYROSCOPE      |        |             |
|----------|----------------|--------|----------------|--------|-------------|
|          | 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**

DLPF configuration

**See also**

[MPU6050\\_RA\\_CONFIG](#)  
[MPU6050\\_CFG\\_DLPF\\_CFG\\_BIT](#)  
[MPU6050\\_CFG\\_DLPF\\_CFG\\_LENGTH](#)

Definition at line 205 of file [MPU6050.cpp](#).

**3.2.3.16 uint8\_t MPU6050::getDMPConfig1 ( )****3.2.3.17 uint8\_t MPU6050::getDMPConfig2 ( )****3.2.3.18 bool MPU6050::getDMPEntered ( )****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 [MPU6050.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\_SLV2\_CTRL, and I2C\_SLV3\_CTRL (Registers 39, 42, 45, and 48). When more than zero bytes are read (I2C\_SLVx\_CTRL\_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\_DATA\_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

|                 |                          |
|-----------------|--------------------------|
| <i>position</i> | Starting position (0-23) |
|-----------------|--------------------------|

#### Returns

Byte read from register

Definition at line 1975 of file MPU6050.cpp.

#### 3.2.3.27 uint32\_t MPU6050::getExternalSensorDWord ( int *position* )

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

#### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>position</i> | Starting position (0-20) |
|-----------------|--------------------------|

#### Returns

Double word read from registers

#### See also

[getExternalSensorByte\(\)](#)

Definition at line 1993 of file MPU6050.cpp.

#### 3.2.3.28 uint16\_t MPU6050::getExternalSensorWord ( int *position* )

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

## Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>position</i> | Starting position (0-21) |
|-----------------|--------------------------|

## Returns

Word read from register

## See also

[getExternalSensorByte\(\)](#)

Definition at line 1984 of file [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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**

|           |  |
|-----------|--|
| <i>ax</i> | 16-bit signed integer container for accelerometer X-axis value |
| <i>ay</i> | 16-bit signed integer container for accelerometer Y-axis value |
| <i>az</i> | 16-bit signed integer container for accelerometer Z-axis value |
| <i>gx</i> | 16-bit signed integer container for gyroscope X-axis value     |
| <i>gy</i> | 16-bit signed integer container for gyroscope Y-axis value     |
| <i>gz</i> | 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 [MPU6050.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**

|           |  |
|-----------|--|
| <i>ax</i> | 16-bit signed integer container for accelerometer X-axis value |
| <i>ay</i> | 16-bit signed integer container for accelerometer Y-axis value |
| <i>az</i> | 16-bit signed integer container for accelerometer Z-axis value |

|           |   |
|-----------|---|
| <i>gx</i> | 16-bit signed integer container for gyroscope X-axis value    |
| <i>gy</i> | 16-bit signed integer container for gyroscope Y-axis value    |
| <i>gz</i> | 16-bit signed integer container for gyroscope Z-axis value    |
| <i>mx</i> | 16-bit signed integer container for magnetometer X-axis value |
| <i>my</i> | 16-bit signed integer container for magnetometer Y-axis value |
| <i>mz</i> | 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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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`:

$$\text{Sample Rate} = \text{Gyroscope Output Rate} / (1 + \text{SMPLRT\_DIV})$$

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 [MPU6050.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:

| <code>FS_SEL</code> | 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

|                |   |
|----------------|---|
| <code>x</code> | 16-bit signed integer container for X-axis rotation |
| <code>y</code> | 16-bit signed integer container for Y-axis rotation |
| <code>z</code> | 16-bit signed integer container for Z-axis rotation |

#### See also

[getMotion6\(\)](#)  
[MPU6050\\_RA\\_GYRO\\_XOUT\\_H](#)

Definition at line 1858 of file [MPU6050.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 [MPU6050.cpp](#).

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

Definition at line 1865 of file [MPU6050.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 [MPU6050.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 [MPU6050.cpp](#).

### 3.2.3.81 `uint8_t MPU6050::getSlave4InputByte ( )`

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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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_INT_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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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

| <i>num</i> | Slave number (0-3) |
|------------|--------------------|
|------------|--------------------|

#### Returns

Current address for specified slave

#### See also

[MPU6050\\_RA\\_I2C\\_SLV0\\_ADDR](#)

Definition at line 901 of file [MPU6050.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**

|            |                    |
|------------|--------------------|
| <i>num</i> | 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 [MPU6050.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:

$1 / (1 + I2C\_MST\_DLY) \text{ Samples}$

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**

|            |                    |
|------------|--------------------|
| <i>num</i> | 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 [MPU6050.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**

|            |                    |
|------------|--------------------|
| <i>num</i> | Slave number (0-3) |
|------------|--------------------|

**Returns**

Current enabled value for specified slave

**See also**

[MPU6050\\_RA\\_I2C\\_SLV0\\_CTRL](#)

Definition at line 949 of file [MPU6050.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 [MPU6050.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**

|            |                    |
|------------|--------------------|
| <i>num</i> | Slave number (0-3) |
|------------|--------------------|

**Returns**

Current active register for specified slave

**See also**

[MPU6050\\_RA\\_I2C\\_SLV0\\_REG](#)

Definition at line 927 of file [MPU6050.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

|            |                    |
|------------|--------------------|
| <i>num</i> | 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 [MPU6050.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

|            |                    |
|------------|--------------------|
| <i>num</i> | 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 [MPU6050.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

|            |                    |
|------------|--------------------|
| <i>num</i> | 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 [MPU6050.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 [MPU6050.cpp](#).

### 3.2.3.108 `bool MPU6050::getStandbyXAccelEnabled ( )`

### 3.2.3.109 `bool MPU6050::getStandbyYGyroEnabled ( )`

### 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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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\_STATUS 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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.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\_RESET.

See also

[MPU6050\\_RA\\_USER\\_CTRL](#)  
[MPU6050\\_USERCTRL\\_SIG\\_COND\\_RESET\\_BIT](#)

Definition at line 2355 of file [MPU6050.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 [MPU6050.cpp](#).

**3.2.3.155 void MPU6050::setAccelerometerPowerOnDelay ( uint8\_t delay )**

Set accelerometer power-on delay.

Parameters

|              |  |
|--------------|--|
| <i>delay</i> | 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 [MPU6050.cpp](#).

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

Set accelerometer FIFO enabled value.

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>enabled</i> | New accelerometer FIFO enabled value |
|----------------|--------------------------------------|

See also

[getAccelFIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 667 of file [MPU6050.cpp](#).

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

Get self-test enabled setting for accelerometer X axis.

## Parameters

|                |                         |
|----------------|-------------------------|
| <i>enabled</i> | Self-test enabled value |
|----------------|-------------------------|

## See also

[MPU6050\\_RA\\_ACCEL\\_CONFIG](#)

Definition at line 270 of file [MPU6050.cpp](#).

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

Get self-test enabled value for accelerometer Y axis.

## Parameters

|                |                         |
|----------------|-------------------------|
| <i>enabled</i> | Self-test enabled value |
|----------------|-------------------------|

## See also

[MPU6050\\_RA\\_ACCEL\\_CONFIG](#)

Definition at line 285 of file [MPU6050.cpp](#).

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

Set self-test enabled value for accelerometer Z axis.

## Parameters

|                |                         |
|----------------|-------------------------|
| <i>enabled</i> | Self-test enabled value |
|----------------|-------------------------|

## See also

[MPU6050\\_RA\\_ACCEL\\_CONFIG](#)

Definition at line 300 of file [MPU6050.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

|              |  |
|--------------|--|
| <i>level</i> | I2C supply voltage level (0=VLOGIC, 1=VDD) |
|--------------|--|

Definition at line 96 of file [MPU6050.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

|                |   |
|----------------|---|
| <i>enabled</i> | New reference clock output enabled status |
|----------------|---|

## See also

[MPU6050\\_RA\\_INT\\_PIN\\_CFG](#)  
[MPU6050\\_INTCFG\\_CLKOUT\\_EN\\_BIT](#)

Definition at line 1474 of file [MPU6050.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

|               |                          |
|---------------|--------------------------|
| <i>source</i> | 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 [MPU6050.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

|                  |                                    |
|------------------|------------------------------------|
| <i>bandwidth</i> | New high-pass filter configuration |
|------------------|------------------------------------|

## See also

[setDHPFMode\(\)](#)  
[MPU6050\\_DHPF\\_RESET](#)  
[MPU6050\\_RA\\_ACCEL\\_CONFIG](#)

Definition at line 376 of file [MPU6050.cpp](#).



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

Set digital low-pass filter configuration.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>mode</i> | 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 [MPU6050.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

|             |                               |
|-------------|-------------------------------|
| <i>sync</i> | New FSYNC configuration value |
|-------------|-------------------------------|

Definition at line 174 of file [MPU6050.cpp](#).

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

Set external data shadow delay enabled status.

## Parameters

|                |  |
|----------------|--|
| <i>enabled</i> | 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 [MPU6050.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

|                |                         |
|----------------|-------------------------|
| <i>enabled</i> | New FIFO enabled status |
|----------------|-------------------------|

## See also

[getFIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_USER\\_CTRL](#)  
[MPU6050\\_USERCTRL\\_FIFO\\_EN\\_BIT](#)

Definition at line 2291 of file [MPU6050.cpp](#).

3.2.3.173 void MPU6050::setFreefallDetectionCounterDecrement ( uint8\_t *decrement* )

Set Free Fall detection counter decrement configuration.

## Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>decrement</i> | New decrement configuration value |
|------------------|-----------------------------------|

## See also

[getFreefallDetectionCounterDecrement\(\)](#)  
[MPU6050\\_RA\\_MOT\\_DETECT\\_CTRL](#)  
[MPU6050\\_DETECT\\_FF\\_COUNT\\_BIT](#)

Definition at line 2231 of file [MPU6050.cpp](#).

3.2.3.174 void MPU6050::setFreefallDetectionDuration ( uint8\_t *duration* )

Get free-fall event duration threshold.

## Parameters

|                 |  |
|-----------------|--|
| <i>duration</i> | New free-fall duration threshold value (LSB = 1ms) |
|-----------------|--|

## See also

[getFreefallDetectionDuration\(\)](#)  
[MPU6050\\_RA\\_FF\\_DUR](#)

Definition at line 438 of file [MPU6050.cpp](#).

3.2.3.175 void MPU6050::setFreefallDetectionThreshold ( uint8\_t *threshold* )

Get free-fall event acceleration threshold.

## Parameters

|                  |  |
|------------------|--|
| <i>threshold</i> | New free-fall acceleration threshold value (LSB = 2mg) |
|------------------|--|

## See also

[getFreefallDetectionThreshold\(\)](#)  
[MPU6050\\_RA\\_FF\\_THR](#)

Definition at line 406 of file [MPU6050.cpp](#).

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

Set FSYNC pin interrupt enabled setting.

## Parameters

|                |   |
|----------------|---|
| <i>enabled</i> | 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 [MPU6050.cpp](#).

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

Set FSYNC interrupt logic level mode.

## Parameters

|             |  |
|-------------|--|
| <i>mode</i> | 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 [MPU6050.cpp](#).

3.2.3.178 void MPU6050::setFullScaleAccelRange ( uint8\_t *range* )

Set full-scale accelerometer range.

## Parameters

|              |  |
|--------------|--|
| <i>range</i> | New full-scale accelerometer range setting |
|--------------|--|

## See also

[getFullScaleAccelRange\(\)](#)

Definition at line 328 of file [MPU6050.cpp](#).

3.2.3.179 void MPU6050::setFullScaleGyroRange ( uint8\_t *range* )

Set full-scale gyroscope range.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>range</i> | 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 [MPU6050.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

|                |                               |
|----------------|-------------------------------|
| <i>enabled</i> | New I2C bypass enabled status |
|----------------|-------------------------------|

See also

[MPU6050\\_RA\\_INT\\_PIN\\_CFG](#)  
[MPU6050\\_INTCFG\\_I2C\\_BYPASS\\_EN\\_BIT](#)

Definition at line 1449 of file [MPU6050.cpp](#).

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

Set I2C Master Mode enabled status.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>enabled</i> | 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 [MPU6050.cpp](#).

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

Set Data Ready interrupt enabled status.

Parameters

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

See also

[getIntDataReadyEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_CFG](#)  
[MPU6050\\_INTERRUPT\\_DATA\\_RDY\\_BIT](#)

Definition at line 1615 of file [MPU6050.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

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

## See also

[getIntFreefallEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_ENABLE](#)  
[MPU6050\\_INTERRUPT\\_FF\\_BIT](#)

Definition at line 1499 of file [MPU6050.cpp](#).

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

Set interrupt drive mode.

## Parameters

|              |  |
|--------------|--|
| <i>drive</i> | 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 [MPU6050.cpp](#).

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

Set interrupt latch mode.

## Parameters

|              |  |
|--------------|--|
| <i>latch</i> | 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 [MPU6050.cpp](#).

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

Set interrupt latch clear mode.

## Parameters

|              |  |
|--------------|--|
| <i>clear</i> | 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 [MPU6050.cpp](#).

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

Set interrupt logic level mode.

## Parameters

|             |  |
|-------------|--|
| <i>mode</i> | 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 [MPU6050.cpp](#).

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

Set FIFO Buffer Overflow interrupt enabled status.

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

## See also

[getIntFIFOBufferOverflowEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_ENABLE](#)  
[MPU6050\\_INTERRUPT\\_FIFO\\_OFLOW\\_BIT](#)

Definition at line 1575 of file [MPU6050.cpp](#).

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

Set Free Fall interrupt enabled status.

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

## See also

[getIntFreefallEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_ENABLE](#)  
[MPU6050\\_INTERRUPT\\_FF\\_BIT](#)

Definition at line 1518 of file [MPU6050.cpp](#).

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

Set I2C Master interrupt enabled status.

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

## See also

[getIntI2CMasterEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_ENABLE](#)  
[MPU6050\\_INTERRUPT\\_I2C\\_MST\\_INT\\_BIT](#)

Definition at line 1595 of file [MPU6050.cpp](#).

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

Set Motion Detection interrupt enabled status.

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

## See also

[getIntMotionEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_ENABLE](#)  
[MPU6050\\_INTERRUPT\\_MOT\\_BIT](#)

Definition at line 1537 of file [MPU6050.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

|                |                              |
|----------------|------------------------------|
| <i>enabled</i> | New interrupt enabled status |
|----------------|------------------------------|

## See also

[getIntZeroMotionEnabled\(\)](#)  
[MPU6050\\_RA\\_INT\\_ENABLE](#)  
[MPU6050\\_INTERRUPT\\_ZMOT\\_BIT](#)

Definition at line 1556 of file [MPU6050.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 [MPU6050.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

|                  |                                   |
|------------------|-----------------------------------|
| <i>decrement</i> | New decrement configuration value |
|------------------|-----------------------------------|

## See also

[getMotionDetectionCounterDecrement\(\)](#)  
[MPU6050\\_RA\\_MOT\\_DETECT\\_CTRL](#)  
[MPU6050\\_DETECT\\_MOT\\_COUNT\\_BIT](#)

Definition at line 2267 of file [MPU6050.cpp](#).

3.2.3.199 void MPU6050::setMotionDetectionDuration ( uint8\_t *duration* )

Set motion detection event duration threshold.



## Parameters

|                 |   |
|-----------------|---|
| <i>duration</i> | New motion detection duration threshold value (LSB = 1ms) |
|-----------------|---|

## See also

[getMotionDetectionDuration\(\)](#)  
[MPU6050\\_RA\\_MOT\\_DUR](#)

Definition at line 502 of file [MPU6050.cpp](#).

3.2.3.200 void MPU6050::setMotionDetectionThreshold ( uint8\_t *threshold* )

Set free-fall event acceleration threshold.

## Parameters

|                  |   |
|------------------|---|
| <i>threshold</i> | New motion detection acceleration threshold value (LSB = 2mg) |
|------------------|---|

## See also

[getMotionDetectionThreshold\(\)](#)  
[MPU6050\\_RA\\_MOT\\_THR](#)

Definition at line 472 of file [MPU6050.cpp](#).

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

Set multi-master enabled value.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>enabled</i> | New multi-master enabled value |
|----------------|--------------------------------|

## See also

[getMultiMasterEnabled\(\)](#)  
[MPU6050\\_RA\\_I2C\\_MST\\_CTRL](#)

Definition at line 751 of file [MPU6050.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

|             |                         |
|-------------|-------------------------|
| <i>rate</i> | New sample rate divider |
|-------------|-------------------------|

## See also

[getRate\(\)](#)  
[MPU6050\\_RA\\_SMPLRT\\_DIV](#)

Definition at line 132 of file [MPU6050.cpp](#).

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

Set Slave 0 FIFO enabled value.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>enabled</i> | New Slave 0 FIFO enabled value |
|----------------|--------------------------------|

## See also

[getSlave0FIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 721 of file [MPU6050.cpp](#).

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

Set Slave 1 FIFO enabled value.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>enabled</i> | New Slave 1 FIFO enabled value |
|----------------|--------------------------------|

## See also

[getSlave1FIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 703 of file [MPU6050.cpp](#).

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

Set Slave 2 FIFO enabled value.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>enabled</i> | New Slave 2 FIFO enabled value |
|----------------|--------------------------------|

## See also

[getSlave2FIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 685 of file [MPU6050.cpp](#).

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

Set Slave 3 FIFO enabled value.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>enabled</i> | New Slave 3 FIFO enabled value |
|----------------|--------------------------------|

## See also

[getSlave3FIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_MST\\_CTRL](#)

Definition at line 792 of file [MPU6050.cpp](#).

3.2.3.208 void MPU6050::setSlave4Address ( uint8\_t *address* )

Set the I2C address of Slave 4.

## Parameters

|                |                         |
|----------------|-------------------------|
| <i>address</i> | New address for Slave 4 |
|----------------|-------------------------|

## See also

[getSlave4Address\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV4\\_ADDR](#)

Definition at line 1084 of file [MPU6050.cpp](#).

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

Set the enabled value for Slave 4.

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>enabled</i> | New enabled value for Slave 4 |
|----------------|-------------------------------|

## See also

[getSlave4Enabled\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV4\\_CTRL](#)

Definition at line 1130 of file [MPU6050.cpp](#).

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

Set the enabled value for Slave 4 transaction interrupts.

## Parameters

|                |   |
|----------------|---|
| <i>enabled</i> | New enabled value for Slave 4 transaction interrupts. |
|----------------|---|

## See also

[getSlave4InterruptEnabled\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV4\\_CTRL](#)

Definition at line 1151 of file [MPU6050.cpp](#).

3.2.3.211 void MPU6050::setSlave4MasterDelay ( uint8\_t *delay* )

Set Slave 4 master delay value.

## Parameters

|              |                                |
|--------------|--------------------------------|
| <i>delay</i> | New Slave 4 master delay value |
|--------------|--------------------------------|

## See also

[getSlave4MasterDelay\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV4\\_CTRL](#)

Definition at line 1199 of file [MPU6050.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

|             |                              |
|-------------|------------------------------|
| <i>data</i> | New byte to write to Slave 4 |
|-------------|------------------------------|

## See also

[MPU6050\\_RA\\_I2C\\_SLV4\\_DO](#)

Definition at line 1112 of file [MPU6050.cpp](#).

3.2.3.213 void MPU6050::setSlave4Register ( uint8\_t *reg* )

Set the active internal register for Slave 4.

## Parameters

|            |                                 |
|------------|---------------------------------|
| <i>reg</i> | New active register for Slave 4 |
|------------|---------------------------------|

## See also

[getSlave4Register\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV4\\_REG](#)

Definition at line 1103 of file [MPU6050.cpp](#).

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

Set write mode for the Slave 4.

## Parameters

|             |   |
|-------------|---|
| <i>mode</i> | 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 [MPU6050.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

|                |                                 |
|----------------|---------------------------------|
| <i>num</i>     | Slave number (0-3)              |
| <i>address</i> | New address for specified slave |

## See also

[getSlaveAddress\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV0\\_ADDR](#)

Definition at line 912 of file [MPU6050.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

|               |   |
|---------------|---|
| <i>num</i>    | Slave number (0-3)                          |
| <i>length</i> | Number of bytes to read for specified slave |

## See also

[getSlaveDataLength\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV0\\_CTRL](#)

Definition at line 1059 of file [MPU6050.cpp](#).

3.2.3.217 void MPU6050::setSlaveDelayEnabled ( uint8\_t *num*, bool *enabled* )

Set slave delay enabled status.

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>num</i>     | Slave number (0-4)              |
| <i>enabled</i> | 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 [MPU6050.cpp](#).

3.2.3.218 void MPU6050::setSlaveEnabled ( uint8\_t *num*, bool *enabled* )

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

## Parameters

|                |                                       |
|----------------|---------------------------------------|
| <i>num</i>     | Slave number (0-3)                    |
| <i>enabled</i> | New enabled value for specified slave |

## See also

[getSlaveEnabled\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV0\\_CTRL](#)

Definition at line 960 of file [MPU6050.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

|             |                    |
|-------------|--------------------|
| <i>num</i>  | Slave number (0-3) |
| <i>data</i> | Byte to write      |

## See also

[MPU6050\\_RA\\_I2C\\_SLV0\\_DO](#)

Definition at line 2074 of file [MPU6050.cpp](#).

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

Set slave read/write transition enabled value.

## Parameters

|                |   |
|----------------|---|
| <i>enabled</i> | New slave read/write transition enabled value |
|----------------|---|

## See also

[getSlaveReadWriteTransitionEnabled\(\)](#)  
[MPU6050\\_RA\\_I2C\\_MST\\_CTRL](#)

Definition at line 814 of file [MPU6050.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

|            |   |
|------------|---|
| <i>num</i> | Slave number (0-3)                      |
| <i>reg</i> | New active register for specified slave |

## See also

[getSlaveRegister\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV0\\_REG](#)

Definition at line 938 of file [MPU6050.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

|                |   |
|----------------|---|
| <i>num</i>     | Slave number (0-3)  |
| <i>enabled</i> | New word pair byte-swapping enabled value for specified slave |

## See also

[getSlaveWordByteSwap\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV0\\_CTRL](#)

Definition at line 986 of file [MPU6050.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

|                |   |
|----------------|---|
| <i>num</i>     | Slave number (0-3)                                      |
| <i>enabled</i> | New word pair grouping order offset for specified slave |

## See also

[getSlaveWordGroupOffset\(\)](#)  
[MPU6050\\_RA\\_I2C\\_SLV0\\_CTRL](#)

Definition at line 1037 of file [MPU6050.cpp](#).

3.2.3.224 void MPU6050::setSlaveWriteMode ( uint8\_t *num*, bool *mode* )

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

## Parameters

|             |   |
|-------------|---|
| <i>num</i>  | Slave number (0-3)  |
| <i>mode</i> | 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 [MPU6050.cpp](#).

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

Set sleep mode status.

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>enabled</i> | New sleep mode enabled status |
|----------------|-------------------------------|

## See also

[getSleepEnabled\(\)](#)  
[MPU6050\\_RA\\_PWR\\_MGMT\\_1](#)  
[MPU6050\\_PWR1\\_SLEEP\\_BIT](#)

Definition at line 2390 of file [MPU6050.cpp](#).

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

3.2.3.227 void MPU6050::setStandbyYGyroEnabled ( 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

|                |                                    |
|----------------|------------------------------------|
| <i>enabled</i> | New temperature FIFO enabled value |
|----------------|------------------------------------|

## See also

[getTempFIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 594 of file [MPU6050.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

|                |                                       |
|----------------|---------------------------------------|
| <i>enabled</i> | New temperature sensor enabled status |
|----------------|---------------------------------------|

## See also

[getTempSensorEnabled\(\)](#)  
[MPU6050\\_RA\\_PWR\\_MGMT\\_1](#)  
[MPU6050\\_PWR1\\_TEMP\\_DIS\\_BIT](#)

Definition at line 2439 of file [MPU6050.cpp](#).

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

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

## Parameters

|                |   |
|----------------|---|
| <i>enabled</i> | New wait-for-external-sensor-data enabled value |
|----------------|---|

## See also

[getWaitForExternalSensorEnabled\(\)](#)  
[MPU6050\\_RA\\_I2C\\_MST\\_CTRL](#)

Definition at line 774 of file [MPU6050.cpp](#).

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

Set wake cycle enabled status.

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>enabled</i> | New sleep mode enabled status |
|----------------|-------------------------------|

## See also

[getWakeCycleEnabled\(\)](#)  
[MPU6050\\_RA\\_PWR\\_MGMT\\_1](#)  
[MPU6050\\_PWR1\\_CYCLE\\_BIT](#)

Definition at line 2411 of file [MPU6050.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

|                |   |
|----------------|---|
| <i>enabled</i> | New gyroscope X-axis FIFO enabled value |
|----------------|---|

## See also

[getXGyroFIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 612 of file [MPU6050.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

|                |   |
|----------------|---|
| <i>enabled</i> | New gyroscope Y-axis FIFO enabled value |
|----------------|---|

See also

[getYGyroFIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 630 of file [MPU6050.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

|                 |  |
|-----------------|--|
| <i>duration</i> | New zero motion detection duration threshold value (LSB = 1ms) |
|-----------------|--|

See also

[getZeroMotionDetectionDuration\(\)](#)  
[MPU6050\\_RA\\_ZRMOT\\_DUR](#)

Definition at line 573 of file [MPU6050.cpp](#).

- 3.2.3.249 void MPU6050::setZeroMotionDetectionThreshold ( uint8\_t *threshold* )

Set zero motion detection event acceleration threshold.

Parameters

|                  |  |
|------------------|--|
| <i>threshold</i> | New zero motion detection acceleration threshold value (LSB = 2mg) |
|------------------|--|

See also

[getZeroMotionDetectionThreshold\(\)](#)  
[MPU6050\\_RA\\_ZRMOT\\_THR](#)

Definition at line 542 of file [MPU6050.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

|                |   |
|----------------|---|
| <i>enabled</i> | New gyroscope Z-axis FIFO enabled value |
|----------------|---|

## See also

[getZGyroFIFOEnabled\(\)](#)  
[MPU6050\\_RA\\_FIFO\\_EN](#)

Definition at line 648 of file [MPU6050.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 [MPU6050.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 [MPU6050.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 [MPU6050.h](#).

3.2.4.2 uint8\_t MPU6050::devAddr [private]

Definition at line 987 of file [MPU6050.h](#).

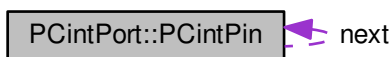
The documentation for this class was generated from the following files:

- [MPU6050.h](#)
- [MPU6050.cpp](#)

### 3.3 PCintPort::PCintPin Class Reference

```
#include <PinChangeInt.h>
```

Collaboration diagram for PCintPort::PCintPin:



#### Public Member Functions

- [PCintPin \(\)](#)

#### Public Attributes

- [PCIntvoidFuncPtr PCintFunc](#)
- [uint8\\_t mode](#)
- [uint8\\_t mask](#)
- [uint8\\_t arduinoPin](#)
- [PCintPin \\* next](#)

#### 3.3.1 Detailed Description

Definition at line 218 of file [PinChangeInt.h](#).

#### 3.3.2 Constructor & Destructor Documentation

##### 3.3.2.1 PCintPort::PCintPin::PCintPin ( ) [inline]

Definition at line 220 of file [PinChangeInt.h](#).

#### 3.3.3 Member Data Documentation

##### 3.3.3.1 uint8\_t PCintPort::PCintPin::arduinoPin

Definition at line 226 of file [PinChangeInt.h](#).

##### 3.3.3.2 uint8\_t PCintPort::PCintPin::mask

Definition at line 225 of file [PinChangeInt.h](#).

##### 3.3.3.3 uint8\_t PCintPort::PCintPin::mode

Definition at line 224 of file [PinChangeInt.h](#).

##### 3.3.3.4 PCintPin\* PCintPort::PCintPin::next

Definition at line 227 of file [PinChangeInt.h](#).

## 3.3.3.5 PCIntvoidFuncPtr PCintPort::PCintPin::PCintFunc

Definition at line 223 of file [PinChangeInt.h](#).

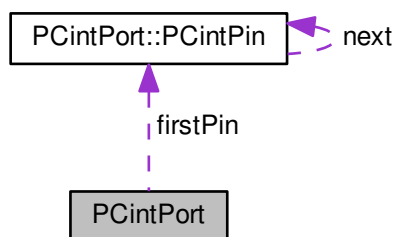
The documentation for this class was generated from the following file:

- [PinChangeInt.h](#)

## 3.4 PCintPort Class Reference

```
#include <PinChangeInt.h>
```

Collaboration diagram for PCintPort:



## Classes

- class [PCintPin](#)

## Public Member Functions

- [PCintPort](#) (int index, int pcindex, volatile uint8\_t &maskReg)
- [INLINE\\_PCINT](#) void [PCint](#) ()

## Static Public Member Functions

- static int8\_t [attachInterrupt](#) (uint8\_t pin, [PCIntvoidFuncPtr](#) userFunc, int mode)
- static void [detachInterrupt](#) (uint8\_t pin)

## Public Attributes

- volatile uint8\_t & [portInputReg](#)

## Static Public Attributes

- static volatile uint8\_t [curr](#) =0
- static volatile uint8\_t [arduinoPin](#) =0
- static volatile uint8\_t [pinState](#) =0

### Protected Member Functions

- void [enable](#) ([PCintPin](#) \*pin, [PCIntvoidFuncPtr](#) userFunc, [uint8\\_t](#) mode)
- [int8\\_t](#) [addPin](#) ([uint8\\_t](#) [arduinoPin](#), [PCIntvoidFuncPtr](#) userFunc, [uint8\\_t](#) mode)

### Protected Attributes

- volatile [uint8\\_t](#) & [portPCMask](#)
- const [uint8\\_t](#) [PCICRbit](#)
- volatile [uint8\\_t](#) [portRisingPins](#)
- volatile [uint8\\_t](#) [portFallingPins](#)
- volatile [uint8\\_t](#) [lastPinView](#)
- [PCintPin](#) \* [firstPin](#)

### 3.4.1 Detailed Description

Definition at line 169 of file [PinChangeInt.h](#).

### 3.4.2 Constructor & Destructor Documentation

3.4.2.1 [PCintPort::PCintPort](#) ( [int](#) *index*, [int](#) *pcindex*, volatile [uint8\\_t](#) & *maskReg* ) [inline]

Definition at line 171 of file [PinChangeInt.h](#).

### 3.4.3 Member Function Documentation

3.4.3.1 [int8\\_t](#) [PCintPort::addPin](#) ( [uint8\\_t](#) *arduinoPin*, [PCIntvoidFuncPtr](#) *userFunc*, [uint8\\_t](#) *mode* ) [protected]

Definition at line 377 of file [PinChangeInt.h](#).

3.4.3.2 [int8\\_t](#) [PCintPort::attachInterrupt](#) ( [uint8\\_t](#) *pin*, [PCIntvoidFuncPtr](#) *userFunc*, [int](#) *mode* ) [static]

Definition at line 421 of file [PinChangeInt.h](#).

3.4.3.3 void [PCintPort::detachInterrupt](#) ( [uint8\\_t](#) *pin* ) [static]

Definition at line 439 of file [PinChangeInt.h](#).

3.4.3.4 void [PCintPort::enable](#) ( [PCintPin](#) \* *pin*, [PCIntvoidFuncPtr](#) *userFunc*, [uint8\\_t](#) *mode* ) [protected]

Definition at line 366 of file [PinChangeInt.h](#).

3.4.3.5 void [PCintPort::PCint](#) ( )

Definition at line 473 of file [PinChangeInt.h](#).

### 3.4.4 Member Data Documentation

3.4.4.1 volatile [uint8\\_t](#) [PCintPort::arduinoPin](#) = 0 [static]

Definition at line 192 of file [PinChangeInt.h](#).

3.4.4.2 volatile [uint8\\_t](#) [PCintPort::curr](#) = 0 [static]

Definition at line 190 of file [PinChangeInt.h](#).

**3.4.4.3 PCintPin\* PCintPort::firstPin** [protected]

Definition at line 236 of file [PinChangeInt.h](#).

**3.4.4.4 volatile uint8\_t PCintPort::lastPinView** [protected]

Definition at line 235 of file [PinChangeInt.h](#).

**3.4.4.5 const uint8\_t PCintPort::PCICRbit** [protected]

Definition at line 232 of file [PinChangeInt.h](#).

**3.4.4.6 volatile uint8\_t PCintPort::pinState =0** [static]

Definition at line 195 of file [PinChangeInt.h](#).

**3.4.4.7 volatile uint8\_t PCintPort::portFallingPins** [protected]

Definition at line 234 of file [PinChangeInt.h](#).

**3.4.4.8 volatile uint8\_t& PCintPort::portInputReg**

Definition at line 186 of file [PinChangeInt.h](#).

**3.4.4.9 volatile uint8\_t& PCintPort::portPCMask** [protected]

Definition at line 231 of file [PinChangeInt.h](#).

**3.4.4.10 volatile uint8\_t PCintPort::portRisingPins** [protected]

Definition at line 233 of file [PinChangeInt.h](#).

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

- [PinChangeInt.h](#)

**3.5 Quaternion Class Reference**

```
#include <helper_3dmath.h>
```

**Public Member Functions**

- [Quaternion](#) ()
- [Quaternion](#) (float nw, float nx, float ny, float nz)
- [Quaternion getProduct](#) ([Quaternion](#) q)
- [Quaternion getConjugate](#) ()
- float [getMagnitude](#) ()
- void [normalize](#) ()
- [Quaternion getNormalized](#) ()

**Public Attributes**

- float [w](#)
- float [x](#)
- float [y](#)
- float [z](#)

### 3.5.1 Detailed Description

Definition at line 35 of file [helper\\_3dmath.h](#).

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 Quaternion::Quaternion ( ) [inline]

Definition at line 42 of file [helper\\_3dmath.h](#).

#### 3.5.2.2 Quaternion::Quaternion ( float *nw*, float *nx*, float *ny*, float *nz* ) [inline]

Definition at line 49 of file [helper\\_3dmath.h](#).

### 3.5.3 Member Function Documentation

#### 3.5.3.1 Quaternion Quaternion::getConjugate ( ) [inline]

Definition at line 69 of file [helper\\_3dmath.h](#).

#### 3.5.3.2 float Quaternion::getMagnitude ( ) [inline]

Definition at line 73 of file [helper\\_3dmath.h](#).

#### 3.5.3.3 Quaternion Quaternion::getNormalized ( ) [inline]

Definition at line 85 of file [helper\\_3dmath.h](#).

#### 3.5.3.4 Quaternion Quaternion::getProduct ( Quaternion *q* ) [inline]

Definition at line 56 of file [helper\\_3dmath.h](#).

#### 3.5.3.5 void Quaternion::normalize ( ) [inline]

Definition at line 77 of file [helper\\_3dmath.h](#).

### 3.5.4 Member Data Documentation

#### 3.5.4.1 float Quaternion::w

Definition at line 37 of file [helper\\_3dmath.h](#).

#### 3.5.4.2 float Quaternion::x

Definition at line 38 of file [helper\\_3dmath.h](#).

#### 3.5.4.3 float Quaternion::y

Definition at line 39 of file [helper\\_3dmath.h](#).

#### 3.5.4.4 float Quaternion::z

Definition at line 40 of file [helper\\_3dmath.h](#).

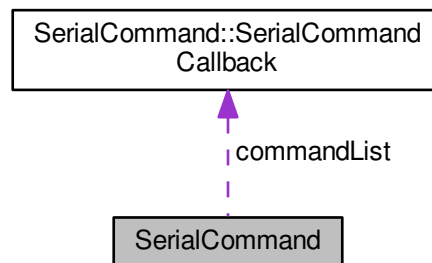
The documentation for this class was generated from the following file:

- [helper\\_3dmath.h](#)

### 3.6 SerialCommand Class Reference

```
#include <SerialCommand.h>
```

Collaboration diagram for SerialCommand:



#### Classes

- struct [SerialCommandCallback](#)

#### Public Member Functions

- [SerialCommand](#) ()
- void [addCommand](#) (const char \*command, void(\*function)())
- void [setDefaultHandler](#) (void(\*function)(const char \*))
- void [readSerial](#) ()
- void [clearBuffer](#) ()
- char \* [next](#) ()

#### Private Attributes

- [SerialCommandCallback](#) \* [commandList](#)
- uint8\_t [commandCount](#)
- void(\* [defaultHandler](#) )(const char \*)
- char [delim](#) [2]
- char [term](#)
- char [buffer](#) [SERIALCOMMAND\_BUFFER+1]
- uint8\_t [bufPos](#)
- char \* [last](#)

#### 3.6.1 Detailed Description

Definition at line 47 of file [SerialCommand.h](#).



### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 SerialCommand::SerialCommand ( )

**SerialCommand** - A Wiring/Arduino library to tokenize and parse commands received over a serial port.

Copyright (C) 2012 Stefan Rado Copyright (C) 2011 Steven Cogswell [steven.cogswell@gmail.com](mailto:steven.cogswell@gmail.com)  
<http://husks.wordpress.com>

Version 20120522

This library is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU General Public License along with this library. If not, see <http://www.gnu.org/licenses/>. Constructor makes sure some things are set.

Definition at line 29 of file [SerialCommand.cpp](#).

### 3.6.3 Member Function Documentation

#### 3.6.3.1 void SerialCommand::addCommand ( const char \* *command*, void(\*)() *function* )

Adds a "command" and a handler function to the list of available commands.

This is used for matching a found token in the buffer, and gives the pointer to the handler function to deal with it.

Definition at line 45 of file [SerialCommand.cpp](#).

#### 3.6.3.2 void SerialCommand::clearBuffer ( )

Definition at line 133 of file [SerialCommand.cpp](#).

#### 3.6.3.3 char \* SerialCommand::next ( )

Retrieve the next token ("word" or "argument") from the command buffer.

Returns NULL if no more tokens exist.

Definition at line 142 of file [SerialCommand.cpp](#).

#### 3.6.3.4 void SerialCommand::readSerial ( )

This checks the Serial stream for characters, and assembles them into a buffer.

When the terminator character (default ' ') is seen, it starts parsing the buffer for a prefix command, and calls handlers setup by [addCommand\(\)](#) member

Definition at line 73 of file [SerialCommand.cpp](#).

#### 3.6.3.5 void SerialCommand::setDefaultHandler ( void(\*) (const char \*) *function* )

This sets up a handler to be called in the event that the received command string isn't in the list of commands.

Definition at line 63 of file [SerialCommand.cpp](#).

### 3.6.4 Member Data Documentation

3.6.4.1 `char SerialCommand::buffer[SERIALCOMMAND_BUFFER+1]` [private]

Definition at line 72 of file [SerialCommand.h](#).

3.6.4.2 `uint8_t SerialCommand::bufPos` [private]

Definition at line 73 of file [SerialCommand.h](#).

3.6.4.3 `uint8_t SerialCommand::commandCount` [private]

Definition at line 64 of file [SerialCommand.h](#).

3.6.4.4 `SerialCommandCallback* SerialCommand::commandList` [private]

Definition at line 63 of file [SerialCommand.h](#).

3.6.4.5 `void(* SerialCommand::defaultHandler)(const char *)` [private]

Definition at line 67 of file [SerialCommand.h](#).

3.6.4.6 `char SerialCommand::delim[2]` [private]

Definition at line 69 of file [SerialCommand.h](#).

3.6.4.7 `char* SerialCommand::last` [private]

Definition at line 74 of file [SerialCommand.h](#).

3.6.4.8 `char SerialCommand::term` [private]

Definition at line 70 of file [SerialCommand.h](#).

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

- [SerialCommand.h](#)
- [SerialCommand.cpp](#)

## 3.7 SerialCommand::SerialCommandCallback Struct Reference

### Public Attributes

- `char command` [SERIALCOMMAND\_MAXCOMMANDELENGTH+1]
- `void(* function)()`

### 3.7.1 Detailed Description

Definition at line 59 of file [SerialCommand.h](#).

### 3.7.2 Member Data Documentation

3.7.2.1 `char SerialCommand::SerialCommandCallback::command[SERIALCOMMAND_MAXCOMMANDELENGTH+1]`

Definition at line 60 of file [SerialCommand.h](#).

3.7.2.2 `void(* SerialCommand::SerialCommandCallback::function)()`

Definition at line 61 of file [SerialCommand.h](#).

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

- [SerialCommand.h](#)

### 3.8 VectorFloat Class Reference

```
#include <helper_3dmath.h>
```

#### Public Member Functions

- [VectorFloat](#) ()
- [VectorFloat](#) (float nx, float ny, float nz)
- float [getMagnitude](#) ()
- void [normalize](#) ()
- [VectorFloat](#) [getNormalized](#) ()
- void [rotate](#) ([Quaternion](#) \*q)
- [VectorFloat](#) [getRotated](#) ([Quaternion](#) \*q)

#### Public Attributes

- float [x](#)
- float [y](#)
- float [z](#)

#### 3.8.1 Detailed Description

Definition at line 159 of file [helper\\_3dmath.h](#).

#### 3.8.2 Constructor & Destructor Documentation

##### 3.8.2.1 [VectorFloat::VectorFloat](#) ( ) [inline]

Definition at line 165 of file [helper\\_3dmath.h](#).

##### 3.8.2.2 [VectorFloat::VectorFloat](#) ( float nx, float ny, float nz ) [inline]

Definition at line 171 of file [helper\\_3dmath.h](#).

#### 3.8.3 Member Function Documentation

##### 3.8.3.1 float [VectorFloat::getMagnitude](#) ( ) [inline]

Definition at line 177 of file [helper\\_3dmath.h](#).

##### 3.8.3.2 [VectorFloat](#) [VectorFloat::getNormalized](#) ( ) [inline]

Definition at line 188 of file [helper\\_3dmath.h](#).

##### 3.8.3.3 [VectorFloat](#) [VectorFloat::getRotated](#) ( [Quaternion](#) \* q ) [inline]

Definition at line 209 of file [helper\\_3dmath.h](#).

##### 3.8.3.4 void [VectorFloat::normalize](#) ( ) [inline]

Definition at line 181 of file [helper\\_3dmath.h](#).

3.8.3.5 void VectorFloat::rotate ( Quaternion \* *q* ) [inline]

Definition at line 194 of file [helper\\_3dmath.h](#).

### 3.8.4 Member Data Documentation

3.8.4.1 float VectorFloat::x

Definition at line 161 of file [helper\\_3dmath.h](#).

3.8.4.2 float VectorFloat::y

Definition at line 162 of file [helper\\_3dmath.h](#).

3.8.4.3 float VectorFloat::z

Definition at line 163 of file [helper\\_3dmath.h](#).

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

- [helper\\_3dmath.h](#)

## 3.9 VectorInt16 Class Reference

```
#include <helper_3dmath.h>
```

### Public Member Functions

- [VectorInt16](#) ()
- [VectorInt16](#) (int16\_t nx, int16\_t ny, int16\_t nz)
- float [getMagnitude](#) ()
- void [normalize](#) ()
- [VectorInt16](#) [getNormalized](#) ()
- void [rotate](#) (Quaternion \*q)
- [VectorInt16](#) [getRotated](#) (Quaternion \*q)

### Public Attributes

- int16\_t [x](#)
- int16\_t [y](#)
- int16\_t [z](#)

### 3.9.1 Detailed Description

Definition at line 92 of file [helper\\_3dmath.h](#).

### 3.9.2 Constructor & Destructor Documentation

3.9.2.1 VectorInt16::VectorInt16 ( ) [inline]

Definition at line 98 of file [helper\\_3dmath.h](#).

3.9.2.2 VectorInt16::VectorInt16 ( int16\_t nx, int16\_t ny, int16\_t nz ) [inline]

Definition at line 104 of file [helper\\_3dmath.h](#).

### 3.9.3 Member Function Documentation

#### 3.9.3.1 float VectorInt16::getMagnitude ( ) [inline]

Definition at line 110 of file [helper\\_3dmath.h](#).

#### 3.9.3.2 VectorInt16 VectorInt16::getNormalized ( ) [inline]

Definition at line 121 of file [helper\\_3dmath.h](#).

#### 3.9.3.3 VectorInt16 VectorInt16::getRotated ( Quaternion \* q ) [inline]

Definition at line 152 of file [helper\\_3dmath.h](#).

#### 3.9.3.4 void VectorInt16::normalize ( ) [inline]

Definition at line 114 of file [helper\\_3dmath.h](#).

#### 3.9.3.5 void VectorInt16::rotate ( Quaternion \* q ) [inline]

Definition at line 127 of file [helper\\_3dmath.h](#).

### 3.9.4 Member Data Documentation

#### 3.9.4.1 int16\_t VectorInt16::x

Definition at line 94 of file [helper\\_3dmath.h](#).

#### 3.9.4.2 int16\_t VectorInt16::y

Definition at line 95 of file [helper\\_3dmath.h](#).

#### 3.9.4.3 int16\_t VectorInt16::z

Definition at line 96 of file [helper\\_3dmath.h](#).

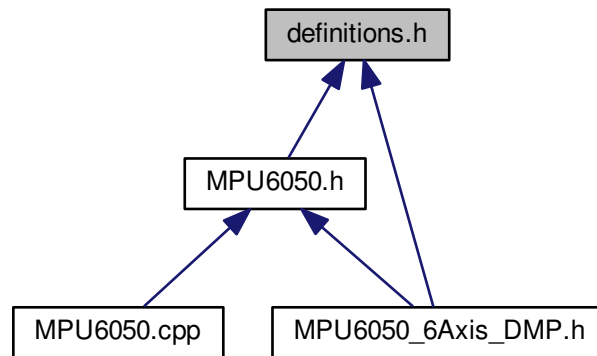
The documentation for this class was generated from the following file:

- [helper\\_3dmath.h](#)

## 4 File Documentation

## 4.1 definitions.h File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- `#define MPU6050_ADDRESS_AD0_LOW 0x68`
- `#define MPU6050_ADDRESS_AD0_HIGH 0x69`
- `#define MPU6050_DEFAULT_ADDRESS MPU6050_ADDRESS_AD0_LOW`
- `#define PWM_32KHZ_PHASE`
- `#define MOTORUPDATE_FREQ 32`
- `#define MPU6050_GYRO_FS MPU6050_GYRO_FS_250`
- `#define MPU6050_DLPF_BW MPU6050_DLPF_BW_256`
- `#define N_SIN 256`
- `#define SCALE_ACC 10000.0`
- `#define SCALE_PID_PARAMS 100.0`
- `#define RC_PIN_ROLL A2`
- `#define RC_PIN_PITCH A1`
- `#define MID_RC 1500`
- `#define MIN_RC 1000`
- `#define MAX_RC 2000`
- `#define RC_DEADBAND 50`
- `#define DMP_50HZ`
- `#define I2C_SPEED 800000L`
- `#define PWM_A_MOTOR1 OCR2A`
- `#define PWM_B_MOTOR1 OCR1B`
- `#define PWM_C_MOTOR1 OCR1A`
- `#define PWM_A_MOTOR0 OCR0A`
- `#define PWM_B_MOTOR0 OCR0B`
- `#define PWM_C_MOTOR0 OCR2B`
- `#define DEBUG_PRINT(x)`
- `#define DEBUG_PRINTF(x, y)`
- `#define DEBUG_PRINTLN(x)`
- `#define DEBUG_PRINTLNf(x, y)`
- `#define CC_FACTOR 32`
- `#define LEDPIN_PINMODE pinMode (8, OUTPUT);`

- `#define LEDPIN_SWITCH` `digitalWrite(8,!bitRead(PORTB,0));`
- `#define LEDPIN_OFF` `digitalWrite(8, LOW);`
- `#define LEDPIN_ON` `digitalWrite(8, HIGH);`

#### 4.1.1 Macro Definition Documentation

##### 4.1.1.1 `#define CC_FACTOR 32`

Definition at line 76 of file [definitions.h](#).

##### 4.1.1.2 `#define DEBUG_PRINT( x )`

Definition at line 68 of file [definitions.h](#).

##### 4.1.1.3 `#define DEBUG_PRINTF( x, y )`

Definition at line 69 of file [definitions.h](#).

##### 4.1.1.4 `#define DEBUG_PRINTLN( x )`

Definition at line 70 of file [definitions.h](#).

##### 4.1.1.5 `#define DEBUG_PRINTLNf( x, y )`

Definition at line 71 of file [definitions.h](#).

##### 4.1.1.6 `#define DMP_50HZ`

Definition at line 40 of file [definitions.h](#).

##### 4.1.1.7 `#define I2C_SPEED 800000L`

Definition at line 47 of file [definitions.h](#).

##### 4.1.1.8 `#define LEDPIN_OFF` `digitalWrite(8, LOW);`

Definition at line 91 of file [definitions.h](#).

##### 4.1.1.9 `#define LEDPIN_ON` `digitalWrite(8, HIGH);`

Definition at line 92 of file [definitions.h](#).

##### 4.1.1.10 `#define LEDPIN_PINMODE` `pinMode (8, OUTPUT);`

Definition at line 89 of file [definitions.h](#).

##### 4.1.1.11 `#define LEDPIN_SWITCH` `digitalWrite(8,!bitRead(PORTB,0));`

Definition at line 90 of file [definitions.h](#).

##### 4.1.1.12 `#define MAX_RC 2000`

Definition at line 36 of file [definitions.h](#).

##### 4.1.1.13 `#define MID_RC 1500`

Definition at line 34 of file [definitions.h](#).

##### 4.1.1.14 `#define MIN_RC 1000`

Definition at line 35 of file [definitions.h](#).

4.1.1.15 `#define MOTORUPDATE_FREQ 32`

Definition at line 16 of file [definitions.h](#).

4.1.1.16 `#define MPU6050_ADDRESS_AD0_HIGH 0x69`

Definition at line 6 of file [definitions.h](#).

4.1.1.17 `#define MPU6050_ADDRESS_AD0_LOW 0x68`

Definition at line 5 of file [definitions.h](#).

4.1.1.18 `#define MPU6050_DEFAULT_ADDRESS MPU6050_ADDRESS_AD0_LOW`

Definition at line 7 of file [definitions.h](#).

4.1.1.19 `#define MPU6050_DLPF_BW MPU6050_DLPF_BW_256`

Definition at line 20 of file [definitions.h](#).

4.1.1.20 `#define MPU6050_GYRO_FS MPU6050_GYRO_FS_250`

Definition at line 19 of file [definitions.h](#).

4.1.1.21 `#define N_SIN 256`

Definition at line 26 of file [definitions.h](#).

4.1.1.22 `#define PWM_32KHZ_PHASE`

Definition at line 11 of file [definitions.h](#).

4.1.1.23 `#define PWM_A_MOTOR0 OCR0A`

Definition at line 57 of file [definitions.h](#).

4.1.1.24 `#define PWM_A_MOTOR1 OCR2A`

Definition at line 53 of file [definitions.h](#).

4.1.1.25 `#define PWM_B_MOTOR0 OCR0B`

Definition at line 58 of file [definitions.h](#).

4.1.1.26 `#define PWM_B_MOTOR1 OCR1B`

Definition at line 54 of file [definitions.h](#).

4.1.1.27 `#define PWM_C_MOTOR0 OCR2B`

Definition at line 59 of file [definitions.h](#).

4.1.1.28 `#define PWM_C_MOTOR1 OCR1A`

Definition at line 55 of file [definitions.h](#).

4.1.1.29 `#define RC_DEADBAND 50`

Definition at line 37 of file [definitions.h](#).



#### 4.1.1.30 #define RC\_PIN\_PITCH A1

Definition at line 33 of file [definitions.h](#).

#### 4.1.1.31 #define RC\_PIN\_ROLL A2

Definition at line 32 of file [definitions.h](#).

#### 4.1.1.32 #define SCALE\_ACC 10000.0

Definition at line 28 of file [definitions.h](#).

#### 4.1.1.33 #define SCALE\_PID\_PARAMS 100.0

Definition at line 29 of file [definitions.h](#).

## 4.2 definitions.h

```

00001 /*****
00002  * Definitions
00003  *****/
00004 // MPU Address Settings
00005 #define MPU6050_ADDRESS_AD0_LOW      0x68 // default for InvenSense evaluation board
00006 #define MPU6050_ADDRESS_AD0_HIGH    0x69 // Drotek MPU breakout board
00007 #define MPU6050_DEFAULT_ADDRESS      MPU6050_ADDRESS_AD0_LOW
00008
00009
00010 // Define Brushless PWM Mode, uncomment ONE setting
00011 #define PWM_32KHZ_PHASE // Resolution 8 bit for PWM
00012 // #define PWM_8KHZ_FAST // Resolution 8 bit for PWM
00013 // #define PWM_4KHZ_PHASE // Resolution 8 bit for PWM
00014 // #define NO_PWM_LOOP
00015
00016 #define MOTORUPDATE_FREQ 32 //in kHz 1,2,4,8 for 32kHz, 1,2,4 for 4kHz
00017
00018 // Do not change for now
00019 #define MPU6050_GYRO_FS MPU6050_GYRO_FS_250 // +-250,500,1000,2000 deg/s
00020 #define MPU6050_DLPF_BW MPU6050_DLPF_BW_256 //0x07 //MPU6050_DLPF_BW_256 //256 //
    5,10,20,42,98,188,256 Hz
00021
00022
00023 // Number of sinus values for full 360 deg.
00024 // NOW FIXED TO 256 !!!
00025 // Reason: Fast Motor Routine using uint8_t overflow for stepping
00026 #define N_SIN 256
00027
00028 #define SCALE_ACC 10000.0
00029 #define SCALE_PID_PARAMS 100.0
00030
00031 // RC Pins
00032 #define RC_PIN_ROLL A2
00033 #define RC_PIN_PITCH A1
00034 #define MID_RC 1500
00035 #define MIN_RC 1000
00036 #define MAX_RC 2000
00037 #define RC_DEADBAND 50
00038
00039 // DMP Update frequency, 100Hz should be enough for repositioning
00040 #define DMP_50HZ // is actually 100Hz due to high gyro read rate
00041 // #define DMP_100HZ
00042 // #define DMP_200HZ
00043
00044 // I2C Frequency
00045 // #define I2C_SPEED 100000L //100kHz normal mode
00046 // #define I2C_SPEED 400000L //400kHz fast mode
00047 #define I2C_SPEED 800000L //800kHz ultra fast mode
00048
00049
00050
00051 // Hardware Abstraction for Motor connectors,
00052 // DO NOT CHANGE UNLESS YOU KNOW WHAT YOU ARE DOING !!!
00053 #define PWM_A_MOTOR1 OCR2A
00054 #define PWM_B_MOTOR1 OCR1B
00055 #define PWM_C_MOTOR1 OCR1A
00056
00057 #define PWM_A_MOTOR0 OCR0A
00058 #define PWM_B_MOTOR0 OCR0B
00059 #define PWM_C_MOTOR0 OCR2B
00060

```

```

00061
00062 #ifndef DEBUG
00063     #define DEBUG_PRINT(x) Serial.print(x)
00064     #define DEBUG_PRINTF(x, y) Serial.print(x, y)
00065     #define DEBUG_PRINTLN(x) Serial.println(x)
00066     #define DEBUG_PRINTLNf(x, y) Serial.println(x, y)
00067 #else
00068     #define DEBUG_PRINT(x)
00069     #define DEBUG_PRINTF(x, y)
00070     #define DEBUG_PRINTLN(x)
00071     #define DEBUG_PRINTLNf(x, y)
00072 #endif
00073
00074
00075 #ifndef PWM_32KHZ_PHASE
00076     #define CC_FACTOR 32
00077 #endif
00078 #ifndef PWM_4KHZ_PHASE
00079     #define CC_FACTOR 4
00080 #endif
00081 #ifndef PWM_8KHZ_FAST
00082     #define CC_FACTOR 8
00083 #endif
00084 #ifndef NO_PWM_LOOP
00085     #define CC_FACTOR 1
00086 #endif
00087
00088
00089 #define LEDPIN_PINMODE      pinMode (8, OUTPUT);
00090 #define LEDPIN_SWITCH      digitalWrite(8,!bitRead(PORTB,0));
00091 #define LEDPIN_OFF         digitalWrite(8, LOW);
00092 #define LEDPIN_ON          digitalWrite(8, HIGH);
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104

```

## 4.3 EEPROMAnything.h File Reference

### Functions

- `template<class T >`  
`int EEPROM_writeAnything (int ee, const T &value)`
- `template<class T >`  
`int EEPROM_readAnything (int ee, T &value)`

#### 4.3.1 Function Documentation

##### 4.3.1.1 `template<class T > int EEPROM_readAnything ( int ee, T & value )`

Definition at line 10 of file [EEPROMAnything.h](#).

##### 4.3.1.2 `template<class T > int EEPROM_writeAnything ( int ee, const T & value )`

Definition at line 1 of file [EEPROMAnything.h](#).

## 4.4 EEPROMAnything.h

```

00001 template <class T> int EEPROM_writeAnything(int ee, const T& value)
00002 {
00003     const byte* p = (const byte*)(const void*)&value;
00004     unsigned int i;
00005     for (i = 0; i < sizeof(value); i++)
00006         EEPROM.write(ee++, *p++);
00007     return i;

```

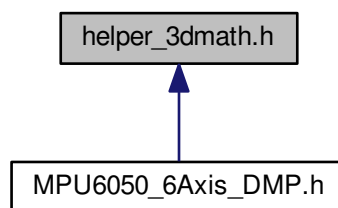
```

00008 }
00009
00010 template <class T> int EEPROM_readAnything(int ee, T& value)
00011 {
00012     byte* p = (byte*)(void*)&value;
00013     unsigned int i;
00014     for (i = 0; i < sizeof(value); i++)
00015         *p++ = EEPROM.read(ee++);
00016     return i;
00017 }

```

## 4.5 helper\_3dmath.h File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- class [Quaternion](#)
- class [VectorInt16](#)
- class [VectorFloat](#)

## 4.6 helper\_3dmath.h

```

00001 // I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class, 3D math helper
00002 // 6/5/2012 by Jeff Rowberg <jeff@rowberg.net>
00003 // Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
00004 //
00005 // Changelog:
00006 // 2012-06-05 - add 3D math helper file to DMP6 example sketch
00007
00008 /* =====
00009 I2Cdev device library code is placed under the MIT license
00010 Copyright (c) 2012 Jeff Rowberg
00011
00012 Permission is hereby granted, free of charge, to any person obtaining a copy
00013 of this software and associated documentation files (the "Software"), to deal
00014 in the Software without restriction, including without limitation the rights
00015 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00016 copies of the Software, and to permit persons to whom the Software is
00017 furnished to do so, subject to the following conditions:
00018
00019 The above copyright notice and this permission notice shall be included in
00020 all copies or substantial portions of the Software.
00021
00022 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00023 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00024 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00025 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00026 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00027 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00028 THE SOFTWARE.
00029 =====
00030 */
00031

```

```

00032 #ifndef _HELPER_3DMATH_H_
00033 #define _HELPER_3DMATH_H_
00034
00035 class Quaternion {
00036     public:
00037         float w;
00038         float x;
00039         float y;
00040         float z;
00041
00042     Quaternion() {
00043         w = 1.0f;
00044         x = 0.0f;
00045         y = 0.0f;
00046         z = 0.0f;
00047     }
00048
00049     Quaternion(float nw, float nx, float ny, float nz) {
00050         w = nw;
00051         x = nx;
00052         y = ny;
00053         z = nz;
00054     }
00055
00056     Quaternion getProduct(Quaternion q) {
00057         // Quaternion multiplication is defined by:
00058         //      (Q1 * Q2).w = (w1w2 - x1x2 - y1y2 - z1z2)
00059         //      (Q1 * Q2).x = (w1x2 + x1w2 + y1z2 - z1y2)
00060         //      (Q1 * Q2).y = (w1y2 - x1z2 + y1w2 + z1x2)
00061         //      (Q1 * Q2).z = (w1z2 + x1y2 - y1x2 + z1w2)
00062         return Quaternion(
00063             w*q.w - x*q.x - y*q.y - z*q.z, // new w
00064             w*q.x + x*q.w + y*q.z - z*q.y, // new x
00065             w*q.y - x*q.z + y*q.w + z*q.x, // new y
00066             w*q.z + x*q.y - y*q.x + z*q.w); // new z
00067     }
00068
00069     Quaternion getConjugate() {
00070         return Quaternion(w, -x, -y, -z);
00071     }
00072
00073     float getMagnitude() {
00074         return sqrt(w*w + x*x + y*y + z*z);
00075     }
00076
00077     void normalize() {
00078         float m = getMagnitude();
00079         w /= m;
00080         x /= m;
00081         y /= m;
00082         z /= m;
00083     }
00084
00085     Quaternion getNormalized() {
00086         Quaternion r(w, x, y, z);
00087         r.normalize();
00088         return r;
00089     }
00090 };
00091
00092 class VectorInt16 {
00093     public:
00094         int16_t x;
00095         int16_t y;
00096         int16_t z;
00097
00098     VectorInt16() {
00099         x = 0;
00100         y = 0;
00101         z = 0;
00102     }
00103
00104     VectorInt16(int16_t nx, int16_t ny, int16_t nz) {
00105         x = nx;
00106         y = ny;
00107         z = nz;
00108     }
00109
00110     float getMagnitude() {
00111         return sqrt(x*x + y*y + z*z);
00112     }
00113
00114     void normalize() {
00115         float m = getMagnitude();
00116         x /= m;
00117         y /= m;
00118         z /= m;

```

```

00119     }
00120
00121     VectorInt16 getNormalized() {
00122         VectorInt16 r(x, y, z);
00123         r.normalize();
00124         return r;
00125     }
00126
00127     void rotate(Quaternion *q) {
00128         // http://www.cprogramming.com/tutorial/3d/quaternions.html
00129         //
00130         http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/transforms/index.htm
00131         // http://content.gpwiki.org/index.php/OpenGL:Tutorials:Using_Quaternions_to_represent_rotation
00132         // ^ or:
00133         http://webcache.googleusercontent.com/search?q=cache:xgJAp3bDNhQJ:content.gpwiki.org/index.php/OpenGL:Tutorials:Using_
00134
00135         // P_out = q * P_in * conj(q)
00136         // - P_out is the output vector
00137         // - q is the orientation quaternion
00138         // - P_in is the input vector (a*aReal)
00139         // - conj(q) is the conjugate of the orientation quaternion (q=[w,x,y,z], q*=[w,-x,-y,-z])
00140         Quaternion p(0, x, y, z);
00141
00142         // quaternion multiplication: q * p, stored back in p
00143         p = q -> getProduct(p);
00144
00145         // quaternion multiplication: p * conj(q), stored back in p
00146         p = p.getProduct(q -> getConjugate());
00147
00148         // p quaternion is now [0, x', y', z']
00149         x = p.x;
00150         y = p.y;
00151         z = p.z;
00152     }
00153
00154     VectorInt16 getRotated(Quaternion *q) {
00155         VectorInt16 r(x, y, z);
00156         r.rotate(q);
00157         return r;
00158     }
00159 };
00160
00161 class VectorFloat {
00162 public:
00163     float x;
00164     float y;
00165     float z;
00166
00167     VectorFloat() {
00168         x = 0;
00169         y = 0;
00170         z = 0;
00171     }
00172
00173     VectorFloat(float nx, float ny, float nz) {
00174         x = nx;
00175         y = ny;
00176         z = nz;
00177     }
00178
00179     float getMagnitude() {
00180         return sqrt(x*x + y*y + z*z);
00181     }
00182
00183     void normalize() {
00184         float m = getMagnitude();
00185         x /= m;
00186         y /= m;
00187         z /= m;
00188     }
00189
00190     VectorFloat getNormalized() {
00191         VectorFloat r(x, y, z);
00192         r.normalize();
00193         return r;
00194     }
00195
00196     void rotate(Quaternion *q) {
00197         Quaternion p(0, x, y, z);
00198
00199         // quaternion multiplication: q * p, stored back in p
00200         p = q -> getProduct(p);
00201
00202         // quaternion multiplication: p * conj(q), stored back in p
00203         p = p.getProduct(q -> getConjugate());
00204
00205         // p quaternion is now [0, x', y', z']

```

```

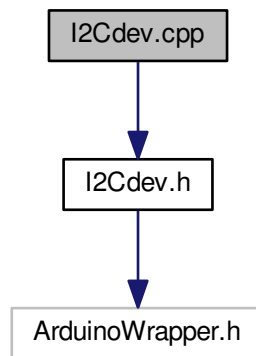
00204         x = p.x;
00205         y = p.y;
00206         z = p.z;
00207     }
00208
00209     VectorFloat getRotated(Quaternion *q) {
00210         VectorFloat r(x, y, z);
00211         r.rotate(q);
00212         return r;
00213     }
00214 };
00215
00216 #endif /* _HELPER_3DMATH_H_ */

```

## 4.7 I2Cdev.cpp File Reference

```
#include "I2Cdev.h"
```

Include dependency graph for I2Cdev.cpp:



## 4.8 I2Cdev.cpp

```

00001 // I2Cdev library collection - Main I2C device class
00002 // Abstracts bit and byte I2C R/W functions into a convenient class
00003 // 6/9/2012 by Jeff Rowberg <jeff@rowberg.net>
00004 //
00005 // Changelog:
00006 //     2012-06-09 - fix major issue with reading > 32 bytes at a time with Arduino Wire
00007 //     - add compiler warnings when using outdated or IDE or limited I2Cdev implementation
00008 //     2011-11-01 - fix write*Bits mask calculation (thanks sasquatch @ Arduino forums)
00009 //     2011-10-03 - added automatic Arduino version detection for ease of use
00010 //     2011-10-02 - added Gene Knight's NBWire TwoWire class implementation with small modifications
00011 //     2011-08-31 - added support for Arduino 1.0 Wire library (methods are different from 0.x)
00012 //     2011-08-03 - added optional timeout parameter to read* methods to easily change from default
00013 //     2011-08-02 - added support for 16-bit registers
00014 //     - fixed incorrect Doxygen comments on some methods
00015 //     - added timeout value for read operations (thanks mem @ Arduino forums)
00016 //     2011-07-30 - changed read/write function structures to return success or byte counts
00017 //     - made all methods static for multi-device memory savings
00018 //     2011-07-28 - initial release
00019
00020 /* =====
00021 I2Cdev device library code is placed under the MIT license
00022 Copyright (c) 2012 Jeff Rowberg
00023
00024 Permission is hereby granted, free of charge, to any person obtaining a copy
00025 of this software and associated documentation files (the "Software"), to deal
00026 in the Software without restriction, including without limitation the rights
00027 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00028 copies of the Software, and to permit persons to whom the Software is
00029 furnished to do so, subject to the following conditions:
00030

```

```

00031 The above copyright notice and this permission notice shall be included in
00032 all copies or substantial portions of the Software.
00033
00034 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00035 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00036 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00037 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00038 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00039 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00040 THE SOFTWARE.
00041 =====
00042 */
00043
00044 #include "I2Cdev.h"
00045
00046 #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
00047
00048     #ifdef I2CDEV_IMPLEMENTATION_WARNINGS
00049         #if ARDUINO < 100
00050             #warning Using outdated Arduino IDE with Wire library is functionally limiting.
00051             #warning Arduino IDE v1.0.1+ with I2Cdev Fastwire implementation is recommended.
00052             #warning This I2Cdev implementation does not support:
00053             #warning - Repeated starts conditions
00054             #warning - Timeout detection (some Wire requests block forever)
00055         #elif ARDUINO == 100
00056             #warning Using outdated Arduino IDE with Wire library is functionally limiting.
00057             #warning Arduino IDE v1.0.1+ with I2Cdev Fastwire implementation is recommended.
00058             #warning This I2Cdev implementation does not support:
00059             #warning - Repeated starts conditions
00060             #warning - Timeout detection (some Wire requests block forever)
00061         #elif ARDUINO > 100
00062             /*
00063             #warning Using current Arduino IDE with Wire library is functionally limiting.
00064             #warning Arduino IDE v1.0.1+ with I2CDEV_BUILTIN_FASTWIRE implementation is recommended.
00065             #warning This I2Cdev implementation does not support:
00066             #warning - Timeout detection (some Wire requests block forever)
00067             */
00068         #endif
00069     #endif
00070
00071 #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
00072
00073     #error The I2CDEV_BUILTIN_FASTWIRE implementation is known to be broken right now. Patience, Iago!
00074
00075 #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
00076
00077     #ifdef I2CDEV_IMPLEMENTATION_WARNINGS
00078         #warning Using I2CDEV_BUILTIN_NBWIRE implementation may adversely affect interrupt detection.
00079         #warning This I2Cdev implementation does not support:
00080         #warning - Repeated starts conditions
00081     #endif
00082
00083     // NBWire implementation based heavily on code by Gene Knight <Gene@Telobot.com>
00084     // Originally posted on the Arduino forum at http://arduino.cc/forum/index.php/topic,70705.0.html
00085     // Originally offered to the i2cdevlib project at http://arduino.cc/forum/index.php/topic,68210.30.html
00086     TwoWire Wire;
00087
00088 #endif
00089
00092 I2Cdev::I2Cdev() {
00093 }
00094
00103 int8_t I2Cdev::readBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t *data,
    uint16_t timeout) {
00104     uint8_t b;
00105     uint8_t count = readByte(devAddr, regAddr, &b, timeout);
00106     *data = b & (1 << bitNum);
00107     return count;
00108 }
00109
00118 int8_t I2Cdev::readBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t *data,
    uint16_t timeout) {
00119     uint16_t b;
00120     uint8_t count = readWord(devAddr, regAddr, &b, timeout);
00121     *data = b & (1 << bitNum);
00122     return count;
00123 }
00124
00134 int8_t I2Cdev::readBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
    uint8_t *data, uint16_t timeout) {
00135     // 01101001 read byte
00136     // 76543210 bit numbers
00137     //   xxx   args: bitStart=4, length=3
00138     //   010   masked
00139     //   -> 010 shifted
00140     uint8_t count, b;
00141     if ((count = readByte(devAddr, regAddr, &b, timeout)) != 0) {

```

```

00142         uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
00143         b &= mask;
00144         b >>= (bitStart - length + 1);
00145         *data = b;
00146     }
00147     return count;
00148 }
00149
00159 int8_t I2Cdev::readBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint16_t *data, uint16_t timeout) {
00160     // 1101011001101001 read byte
00161     // fedcba9876543210 bit numbers
00162     //   xxx           args: bitStart=12, length=3
00163     //   010           masked
00164     //               -> 010 shifted
00165     uint8_t count;
00166     uint16_t w;
00167     if ((count = readWord(devAddr, regAddr, &w, timeout)) != 0) {
00168         uint16_t mask = ((1 << length) - 1) << (bitStart - length + 1);
00169         w &= mask;
00170         w >>= (bitStart - length + 1);
00171         *data = w;
00172     }
00173     return count;
00174 }
00175
00183 int8_t I2Cdev::readByte(uint8_t devAddr, uint8_t regAddr, uint8_t *data, uint16_t timeout)
{
00184     return readBytes(devAddr, regAddr, 1, data, timeout);
00185 }
00186
00194 int8_t I2Cdev::readWord(uint8_t devAddr, uint8_t regAddr, uint16_t *data, uint16_t timeout)
{
00195     return readWords(devAddr, regAddr, 1, data, timeout);
00196 }
00197
00206 int8_t I2Cdev::readBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *data,
uint16_t timeout) {
00207     #ifdef I2CDEV_SERIAL_DEBUG
00208         Serial.print("I2C (0x");
00209         Serial.print(devAddr, HEX);
00210         Serial.print(") reading ");
00211         Serial.print(length, DEC);
00212         Serial.print(" bytes from 0x");
00213         Serial.print(regAddr, HEX);
00214         Serial.print("...\n");
00215     #endif
00216
00217     int8_t count = 0;
00218     uint32_t t1 = millis();
00219
00220     #if (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE)
00221
00222         #if (ARDUINO < 100)
00223             // Arduino v00xx (before v1.0), Wire library
00224
00225             // I2C/TWI subsystem uses internal buffer that breaks with large data requests
00226             // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
00227             // smaller chunks instead of all at once
00228             for (uint8_t k = 0; k < length; k += min(length, BUFFER_LENGTH)) {
00229                 Wire.beginTransmission(devAddr);
00230                 Wire.send(regAddr);
00231                 Wire.endTransmission();
00232                 Wire.beginTransmission(devAddr);
00233                 Wire.requestFrom(devAddr, (uint8_t)min(length - k, BUFFER_LENGTH));
00234
00235                 for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
00236                     data[count] = Wire.receive();
00237                     #ifdef I2CDEV_SERIAL_DEBUG
00238                         Serial.print(data[count], HEX);
00239                         if (count + 1 < length) Serial.print(" ");
00240                     #endif
00241                 }
00242
00243                 Wire.endTransmission();
00244             }
00245         #elif (ARDUINO == 100)
00246             // Arduino v1.0.0, Wire library
00247             // Adds standardized write() and read() stream methods instead of send() and receive()
00248
00249             // I2C/TWI subsystem uses internal buffer that breaks with large data requests
00250             // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
00251             // smaller chunks instead of all at once
00252             for (uint8_t k = 0; k < length; k += min(length, BUFFER_LENGTH)) {
00253                 Wire.beginTransmission(devAddr);
00254                 Wire.write(regAddr);
00255                 Wire.endTransmission();

```



```

00256         Wire.beginTransmission(devAddr);
00257         Wire.requestFrom(devAddr, (uint8_t)min(length - k, BUFFER_LENGTH));
00258
00259         for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
00260             data[count] = Wire.read();
00261             #ifdef I2CDEV_SERIAL_DEBUG
00262                 Serial.print(data[count], HEX);
00263                 if (count + 1 < length) Serial.print(" ");
00264             #endif
00265         }
00266
00267         Wire.endTransmission();
00268     }
00269 #elif (ARDUINO > 100)
00270     // Arduino v1.0.1+, Wire library
00271     // Adds official support for repeated start condition, yay!
00272
00273     // I2C/TWI subsystem uses internal buffer that breaks with large data requests
00274     // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
00275     // smaller chunks instead of all at once
00276     for (uint8_t k = 0; k < length; k += min(length, BUFFER_LENGTH)) {
00277         Wire.beginTransmission(devAddr);
00278         Wire.write(regAddr);
00279         Wire.endTransmission();
00280         Wire.beginTransmission(devAddr);
00281         Wire.requestFrom(devAddr, (uint8_t)min(length - k, BUFFER_LENGTH));
00282
00283         for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
00284             data[count] = Wire.read();
00285             #ifdef I2CDEV_SERIAL_DEBUG
00286                 Serial.print(data[count], HEX);
00287                 if (count + 1 < length) Serial.print(" ");
00288             #endif
00289         }
00290
00291         Wire.endTransmission();
00292     }
00293 #endif
00294
00295 #elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
00296     // Fastwire library (STILL UNDER DEVELOPMENT, NON-FUNCTIONAL!)
00297
00298     // no loop required for fastwire
00299     uint8_t status = Fastwire::readBuf(devAddr, regAddr, data, length);
00300     if (status == 0) {
00301         count = length; // success
00302     } else {
00303         count = -1; // error
00304     }
00305 #endif
00306
00307 // check for timeout
00308 if (timeout > 0 && millis() - t1 >= timeout && count < length) count = -1; // timeout
00309
00310 #ifdef I2CDEV_SERIAL_DEBUG
00311     Serial.print(". Done (");
00312     Serial.print(count, DEC);
00313     Serial.println(" read).");
00314 #endif
00315
00316 return count;
00317 }
00318 }
00319
00320 int8_t I2Cdev::readWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t *data,
00321                          uint16_t timeout) {
00322     #ifdef I2CDEV_SERIAL_DEBUG
00323         Serial.print("I2C (0x");
00324         Serial.print(devAddr, HEX);
00325         Serial.print(") reading ");
00326         Serial.print(length, DEC);
00327         Serial.print(" words from 0x");
00328         Serial.print(regAddr, HEX);
00329         Serial.print("...\n");
00330     #endif
00331
00332     int8_t count = 0;
00333     uint32_t t1 = millis();
00334
00335     #if (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE)
00336         #if (ARDUINO < 100)
00337             // Arduino v00xx (before v1.0), Wire library
00338
00339             // I2C/TWI subsystem uses internal buffer that breaks with large data requests
00340             // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
00341             // smaller chunks instead of all at once

```

```

00350         for (uint8_t k = 0; k < length * 2; k += min(length * 2, BUFFER_LENGTH)) {
00351             Wire.beginTransaction(devAddr);
00352             Wire.send(regAddr);
00353             Wire.endTransmission();
00354             Wire.beginTransaction(devAddr);
00355             Wire.requestFrom(devAddr, (uint8_t)(length * 2)); // length=words, this wants bytes
00356
00357             bool msb = true; // starts with MSB, then LSB
00358             for (; Wire.available() && count < length && (timeout == 0 || millis() - t1 < timeout);) {
00359                 if (msb) {
00360                     // first byte is bits 15-8 (MSb=15)
00361                     data[count] = Wire.receive() << 8;
00362                 } else {
00363                     // second byte is bits 7-0 (LSb=0)
00364                     data[count] |= Wire.receive();
00365                     #ifdef I2CDEV_SERIAL_DEBUG
00366                         Serial.print(data[count], HEX);
00367                         if (count + 1 < length) Serial.print(" ");
00368                     #endif
00369                     count++;
00370                 }
00371                 msb = !msb;
00372             }
00373
00374             Wire.endTransmission();
00375         }
00376     #elif (ARDUINO == 100)
00377         // Arduino v1.0.0, Wire library
00378         // Adds standardized write() and read() stream methods instead of send() and receive()
00379
00380         // I2C/TWI subsystem uses internal buffer that breaks with large data requests
00381         // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
00382         // smaller chunks instead of all at once
00383         for (uint8_t k = 0; k < length * 2; k += min(length * 2, BUFFER_LENGTH)) {
00384             Wire.beginTransaction(devAddr);
00385             Wire.write(regAddr);
00386             Wire.endTransmission();
00387             Wire.beginTransaction(devAddr);
00388             Wire.requestFrom(devAddr, (uint8_t)(length * 2)); // length=words, this wants bytes
00389
00390             bool msb = true; // starts with MSB, then LSB
00391             for (; Wire.available() && count < length && (timeout == 0 || millis() - t1 < timeout);) {
00392                 if (msb) {
00393                     // first byte is bits 15-8 (MSb=15)
00394                     data[count] = Wire.read() << 8;
00395                 } else {
00396                     // second byte is bits 7-0 (LSb=0)
00397                     data[count] |= Wire.read();
00398                     #ifdef I2CDEV_SERIAL_DEBUG
00399                         Serial.print(data[count], HEX);
00400                         if (count + 1 < length) Serial.print(" ");
00401                     #endif
00402                     count++;
00403                 }
00404                 msb = !msb;
00405             }
00406
00407             Wire.endTransmission();
00408         }
00409     #elif (ARDUINO > 100)
00410         // Arduino v1.0.1+, Wire library
00411         // Adds official support for repeated start condition, yay!
00412
00413         // I2C/TWI subsystem uses internal buffer that breaks with large data requests
00414         // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
00415         // smaller chunks instead of all at once
00416         for (uint8_t k = 0; k < length * 2; k += min(length * 2, BUFFER_LENGTH)) {
00417             Wire.beginTransaction(devAddr);
00418             Wire.write(regAddr);
00419             Wire.endTransmission();
00420             Wire.beginTransaction(devAddr);
00421             Wire.requestFrom(devAddr, (uint8_t)(length * 2)); // length=words, this wants bytes
00422
00423             bool msb = true; // starts with MSB, then LSB
00424             for (; Wire.available() && count < length && (timeout == 0 || millis() - t1 < timeout);) {
00425                 if (msb) {
00426                     // first byte is bits 15-8 (MSb=15)
00427                     data[count] = Wire.read() << 8;
00428                 } else {
00429                     // second byte is bits 7-0 (LSb=0)
00430                     data[count] |= Wire.read();
00431                     #ifdef I2CDEV_SERIAL_DEBUG
00432                         Serial.print(data[count], HEX);
00433                         if (count + 1 < length) Serial.print(" ");
00434                     #endif
00435                     count++;
00436                 }

```

```

00437         msb = !msb;
00438     }
00439
00440     Wire.endTransmission();
00441 }
00442 #endif
00443
00444 #elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
00445     // Fastwire library (STILL UNDER DEVELOPMENT, NON-FUNCTIONAL!)
00446
00447     // no loop required for fastwire
00448     uint16_t intermediate[(uint8_t)length];
00449     uint8_t status = Fastwire::readBuf(devAddr, regAddr, (uint8_t *)intermediate, (uint8_t)(length * 2)
);
00450     if (status == 0) {
00451         count = length; // success
00452         for (uint8_t i = 0; i < length; i++) {
00453             data[i] = (intermediate[2*i] << 8) | intermediate[2*i + 1];
00454         }
00455     } else {
00456         count = -1; // error
00457     }
00458 #endif
00459
00460 if (timeout > 0 && millis() - t1 >= timeout && count < length) count = -1; // timeout
00462 #ifdef I2CDEV_SERIAL_DEBUG
00463     Serial.print(". Done (");
00464     Serial.print(count, DEC);
00465     Serial.println(" read).");
00466 #endif
00467 return count;
00468 }
00469
00470 }
00471
00479 bool I2Cdev::writeBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t data) {
00480     uint8_t b;
00481     readByte(devAddr, regAddr, &b);
00482     b = (data != 0) ? (b | (1 << bitNum)) : (b & ~(1 << bitNum));
00483     return writeByte(devAddr, regAddr, b);
00484 }
00485
00493 bool I2Cdev::writeBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t data) {
00494     uint16_t w;
00495     readWord(devAddr, regAddr, &w);
00496     w = (data != 0) ? (w | (1 << bitNum)) : (w & ~(1 << bitNum));
00497     return writeWord(devAddr, regAddr, w);
00498 }
00499
00508 bool I2Cdev::writeBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
uint8_t data) {
00509     // 010 value to write
00510     // 76543210 bit numbers
00511     //   xxx   args: bitStart=4, length=3
00512     // 00011100 mask byte
00513     // 10101111 original value (sample)
00514     // 10100011 original & ~mask
00515     // 10101011 masked | value
00516     uint8_t b;
00517     if (readByte(devAddr, regAddr, &b) != 0) {
00518         uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
00519         data <= (bitStart - length + 1); // shift data into correct position
00520         data &= mask; // zero all non-important bits in data
00521         b &= ~(mask); // zero all important bits in existing byte
00522         b |= data; // combine data with existing byte
00523         return writeByte(devAddr, regAddr, b);
00524     } else {
00525         return false;
00526     }
00527 }
00528
00537 bool I2Cdev::writeBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint16_t data) {
00538     // 010 value to write
00539     // fedcba9876543210 bit numbers
00540     //   xxx   args: bitStart=12, length=3
00541     // 0001110000000000 mask byte
00542     // 1010111110010110 original value (sample)
00543     // 1010001110010110 original & ~mask
00544     // 1010101110010110 masked | value
00545     uint16_t w;
00546     if (readWord(devAddr, regAddr, &w) != 0) {
00547         uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
00548         data <= (bitStart - length + 1); // shift data into correct position
00549         data &= mask; // zero all non-important bits in data
00550         w &= ~(mask); // zero all important bits in existing word

```

```

00551         w |= data; // combine data with existing word
00552         return writeWord(devAddr, regAddr, w);
00553     } else {
00554         return false;
00555     }
00556 }
00557
00564 bool I2Cdev::writeByte(uint8_t devAddr, uint8_t regAddr, uint8_t data) {
00565     return writeBytes(devAddr, regAddr, 1, &data);
00566 }
00567
00574 bool I2Cdev::writeWord(uint8_t devAddr, uint8_t regAddr, uint16_t data) {
00575     return writeWords(devAddr, regAddr, 1, &data);
00576 }
00577
00585 bool I2Cdev::writeBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t* data)
00586 {
00587     #ifdef I2CDEV_SERIAL_DEBUG
00588         Serial.print("I2C (0x");
00589         Serial.print(devAddr, HEX);
00590         Serial.print(") writing ");
00591         Serial.print(length, DEC);
00592         Serial.print(" bytes to 0x");
00593         Serial.print(regAddr, HEX);
00594         Serial.print("...");
00595     #endif
00596     uint8_t status = 0;
00597     #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO < 100) || I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_NBWIRE)
00598         Wire.beginTransmission(devAddr);
00599         Wire.send((uint8_t) regAddr); // send address
00600     #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >= 100)
00601         Wire.beginTransmission(devAddr);
00602         Wire.write((uint8_t) regAddr); // send address
00603     #endif
00604     for (uint8_t i = 0; i < length; i++) {
00605         #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO < 100) || I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_NBWIRE)
00606             Wire.send((uint8_t) data[i]);
00607         #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >= 100)
00608             Wire.write((uint8_t) data[i]);
00609         #elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
00610             status = Fastwire::write(devAddr, regAddr, data[i]);
00611             Serial.println(status);
00612         #endif
00613         #ifdef I2CDEV_SERIAL_DEBUG
00614             Serial.print(data[i], HEX);
00615             if (i + 1 < length) Serial.print(" ");
00616         #endif
00617     }
00618     #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO < 100) || I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_NBWIRE)
00619         Wire.endTransmission();
00620     #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >= 100)
00621         status = Wire.endTransmission();
00622     #endif
00623     #ifdef I2CDEV_SERIAL_DEBUG
00624         Serial.println(". Done.");
00625     #endif
00626     return status == 0;
00627 }
00635 bool I2Cdev::writeWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t* data)
00636 {
00637     #ifdef I2CDEV_SERIAL_DEBUG
00638         Serial.print("I2C (0x");
00639         Serial.print(devAddr, HEX);
00640         Serial.print(") writing ");
00641         Serial.print(length, DEC);
00642         Serial.print(" words to 0x");
00643         Serial.print(regAddr, HEX);
00644         Serial.print("...");
00645     #endif
00646     uint8_t status = 0;
00647     #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO < 100) || I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_NBWIRE)
00648         Wire.beginTransmission(devAddr);
00649         Wire.send(regAddr); // send address
00650     #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >= 100)
00651         Wire.beginTransmission(devAddr);
00652         Wire.write(regAddr); // send address
00653     #endif
00654     for (uint8_t i = 0; i < length * 2; i++) {
00655         #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO < 100) || I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_NBWIRE)
00656             Wire.send((uint8_t)(data[i++] >> 8)); // send MSB
00657             Wire.send((uint8_t)data[i]); // send LSB

```

```

00657         #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >= 100)
00658             Wire.write((uint8_t)(data[i++] >> 8)); // send MSB
00659             Wire.write((uint8_t)data[i]);           // send LSB
00660         #elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
00661             status = Fastwire::write(devAddr, regAddr, (uint8_t)(data[i++] >> 8));
00662             status = Fastwire::write(devAddr, regAddr + 1, (uint8_t)data[i]);
00663         #endif
00664         #ifdef I2CDEV_SERIAL_DEBUG
00665             Serial.print(data[i], HEX);
00666             if (i + 1 < length) Serial.print(" ");
00667         #endif
00668     }
00669     #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO < 100) || I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_NBWIRE)
00670         Wire.endTransmission();
00671     #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >= 100)
00672         status = Wire.endTransmission();
00673     #endif
00674     #ifdef I2CDEV_SERIAL_DEBUG
00675         Serial.println(". Done.");
00676     #endif
00677     return status == 0;
00678 }
00679
00683 uint16_t I2Cdev::readTimeout = I2CDEV_DEFAULT_READ_TIMEOUT;
00684
00685 #if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
00686     /*
00687     FastWire 0.2
00688     This is a library to help faster programs to read I2C devices.
00689     Copyright (C) 2011 Francesco Ferrara
00690     occhiobello at gmail dot com
00691     */
00692
00693     boolean Fastwire::waitInt() {
00694         int l = 250;
00695         while (!(TWCR & (1 << TWINT)) && l-- > 0);
00696         return l > 0;
00697     }
00698
00699     void Fastwire::setup(int khz, boolean pullup) {
00700         TWCR = 0;
00701         #if defined(__AVR_ATmega168__) || defined(__AVR_ATmega8__) || defined(__AVR_ATmega328P__)
00702             // activate internal pull-ups for twi (PORTC bits 4 & 5)
00703             // as per note from atmega8 manual pg167
00704             if (pullup) PORTC |= ((1 << 4) | (1 << 5));
00705             else PORTC &= ~((1 << 4) | (1 << 5));
00706         #elif defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644__)
00707             // activate internal pull-ups for twi (PORTC bits 0 & 1)
00708             if (pullup) PORTC |= ((1 << 0) | (1 << 1));
00709             else PORTC &= ~((1 << 0) | (1 << 1));
00710         #else
00711             // activate internal pull-ups for twi (PORTD bits 0 & 1)
00712             // as per note from atmega128 manual pg204
00713             if (pullup) PORTD |= ((1 << 0) | (1 << 1));
00714             else PORTD &= ~((1 << 0) | (1 << 1));
00715         #endif
00716
00717         TWSR = 0; // no prescaler => prescaler = 1
00718         TWBR = ((16000L / khz) - 16) / 2; // change the I2C clock rate
00719         TWCR = 1 << TWEN; // enable twi module, no interrupt
00720     }
00721
00722     byte Fastwire::write(byte device, byte address, byte value) {
00723         byte twst, retry;
00724
00725         retry = 2;
00726         do {
00727             TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO) | (1 << TWSTA);
00728             if (!waitInt()) return 1;
00729             twst = TWSR & 0xF8;
00730             if (twst != TW_START && twst != TW_REP_START) return 2;
00731
00732             TWDR = device & 0xFE; // send device address without read bit (1)
00733             TWCR = (1 << TWINT) | (1 << TWEN);
00734             if (!waitInt()) return 3;
00735             twst = TWSR & 0xF8;
00736         } while (twst == TW_MT_SLA_NACK && retry-- > 0);
00737         if (twst != TW_MT_SLA_ACK) return 4;
00738
00739         TWDR = address; // send data to the previously addressed device
00740         TWCR = (1 << TWINT) | (1 << TWEN);
00741         if (!waitInt()) return 5;
00742         twst = TWSR & 0xF8;
00743         if (twst != TW_MT_DATA_ACK) return 6;
00744
00745         TWDR = value; // send data to the previously addressed device

```

```

00746     TWCR = (1 << TWINT) | (1 << TWEN);
00747     if (!waitInt()) return 7;
00748     twst = TWSR & 0xF8;
00749     if (twst != TW_MT_DATA_ACK) return 8;
00750
00751     return 0;
00752 }
00753
00754 byte Fastwire::readBuf(byte device, byte address, byte *data, byte num) {
00755     byte twst, retry;
00756
00757     retry = 2;
00758     do {
00759         TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO) | (1 << TWSTA);
00760         if (!waitInt()) return 16;
00761         twst = TWSR & 0xF8;
00762         if (twst != TW_START && twst != TW_REP_START) return 17;
00763
00764         TWDR = device & 0xfe; // send device address to write
00765         TWCR = (1 << TWINT) | (1 << TWEN);
00766         if (!waitInt()) return 18;
00767         twst = TWSR & 0xF8;
00768     } while (twst == TW_MT_SLA_NACK && retry-- > 0);
00769     if (twst != TW_MT_SLA_ACK) return 19;
00770
00771     TWDR = address; // send data to the previously addressed device
00772     TWCR = (1 << TWINT) | (1 << TWEN);
00773     if (!waitInt()) return 20;
00774     twst = TWSR & 0xF8;
00775     if (twst != TW_MT_DATA_ACK) return 21;
00776
00777     /**/
00778
00779     retry = 2;
00780     do {
00781         TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO) | (1 << TWSTA);
00782         if (!waitInt()) return 22;
00783         twst = TWSR & 0xF8;
00784         if (twst != TW_START && twst != TW_REP_START) return 23;
00785
00786         TWDR = device | 0x01; // send device address with the read bit (1)
00787         TWCR = (1 << TWINT) | (1 << TWEN);
00788         if (!waitInt()) return 24;
00789         twst = TWSR & 0xF8;
00790     } while (twst == TW_MR_SLA_NACK && retry-- > 0);
00791     if (twst != TW_MR_SLA_ACK) return 25;
00792
00793     for(uint8_t i = 0; i < num; i++) {
00794         if (i == num - 1)
00795             TWCR = (1 << TWINT) | (1 << TWEN);
00796         else
00797             TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
00798         if (!waitInt()) return 26;
00799         twst = TWSR & 0xF8;
00800         if (twst != TW_MR_DATA_ACK && twst != TW_MR_DATA_NACK) return twst;
00801         data[i] = TWDR;
00802     }
00803
00804     return 0;
00805 }
00806 #endif
00807
00808 #if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
00809     // NBWire implementation based heavily on code by Gene Knight <Gene@Telobot.com>
00810     // Originally posted on the Arduino forum at http://arduino.cc/forum/index.php/topic,70705.0.html
00811     // Originally offered to the i2cdevlib project at http://arduino.cc/forum/index.php/topic,68210.30.html
00812
00813     /*
00814     call this version 1.0
00815
00816     Offhand, the only funky part that I can think of is in nbrequestFrom, where the buffer
00817     length and index are set *before* the data is actually read. The problem is that these
00818     are variables local to the TwoWire object, and by the time we actually have read the
00819     data, and know what the length actually is, we have no simple access to the object's
00820     variables. The actual bytes read *is* given to the callback function, though.
00821
00822     The ISR code for a slave receiver is commented out. I don't have that setup, and can't
00823     verify it at this time. Save it for 2.0!
00824
00825     The handling of the read and write processes here is much like in the demo sketch code:
00826     the process is broken down into sequential functions, where each registers the next as a
00827     callback, essentially.
00828
00829     For example, for the Read process, twi_read00 just returns if TWI is not yet in a
00830     ready state. When there's another interrupt, and the interface *is* ready, then it
00831     sets up the read, starts it, and registers twi_read01 as the function to call after
00832     the *next* interrupt. twi_read01, then, just returns if the interface is still in a

```

```

00833     "reading" state. When the reading is done, it copies the information to the buffer,
00834     cleans up, and calls the user-requested callback function with the actual number of
00835     bytes read.
00836
00837     The writing is similar.
00838
00839     Questions, comments and problems can go to Gene@Telobot.com.
00840
00841     Thumbs Up!
00842     Gene Knight
00843
00844     */
00845
00846     uint8_t TwoWire::rxBuffer[NBWIRE_BUFFER_LENGTH];
00847     uint8_t TwoWire::rxBufferIndex = 0;
00848     uint8_t TwoWire::rxBufferLength = 0;
00849
00850     uint8_t TwoWire::txAddress = 0;
00851     uint8_t TwoWire::txBuffer[NBWIRE_BUFFER_LENGTH];
00852     uint8_t TwoWire::txBufferIndex = 0;
00853     uint8_t TwoWire::txBufferLength = 0;
00854
00855     //uint8_t TwoWire::transmitting = 0;
00856     void (*TwoWire::user_onRequest)(void);
00857     void (*TwoWire::user_onReceive)(int);
00858
00859     static volatile uint8_t twi_transmitting;
00860     static volatile uint8_t twi_state;
00861     static uint8_t twi_slarw;
00862     static volatile uint8_t twi_error;
00863     static uint8_t twi_masterBuffer[TWI_BUFFER_LENGTH];
00864     static volatile uint8_t twi_masterBufferIndex;
00865     static uint8_t twi_masterBufferLength;
00866     static uint8_t twi_rxBuffer[TWI_BUFFER_LENGTH];
00867     static volatile uint8_t twi_rxBufferIndex;
00868     //static volatile uint8_t twi_interrupt_continue_command;
00869     static volatile uint8_t twi_return_value;
00870     static volatile uint8_t twi_done;
00871     void (*twi_cbendTransmissionDone)(int);
00872     void (*twi_cbreadFromDone)(int);
00873
00874     void twi_init() {
00875         // initialize state
00876         twi_state = TWI_READY;
00877
00878         // activate internal pull-ups for twi
00879         // as per note from atmega8 manual pgl67
00880         sbi(PORTC, 4);
00881         sbi(PORTC, 5);
00882
00883         // initialize twi prescaler and bit rate
00884         cbi(TWSR, TWPS0); // TWI Status Register - Prescaler bits
00885         cbi(TWSR, TWPS1);
00886
00887         /* twi bit rate formula from atmega128 manual pg 204
00888         SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
00889         note: TWBR should be 10 or higher for master mode
00890         It is 72 for a 16mhz Wiring board with 100kHz TWI */
00891
00892         TWBR = ((CPU_FREQ / TWI_FREQ) - 16) / 2; // bitrate register
00893         // enable twi module, acks, and twi interrupt
00894
00895         TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);
00896
00897         /* TWEN - TWI Enable Bit
00898         TWIE - TWI Interrupt Enable
00899         TWEA - TWI Enable Acknowledge Bit
00900         TWINT - TWI Interrupt Flag
00901         TWSTA - TWI Start Condition
00902         */
00903     }
00904
00905     typedef struct {
00906         uint8_t address;
00907         uint8_t* data;
00908         uint8_t length;
00909         uint8_t wait;
00910         uint8_t i;
00911     } twi_Write_Vars;
00912
00913     twi_Write_Vars *ptwv = 0;
00914     static void (*fNextInterruptFunction)(void) = 0;
00915
00916     void twi_Finish(byte bRetVal) {
00917         if (ptwv) {
00918             free(ptwv);
00919             ptwv = 0;

```

```

00920     }
00921     twi_Done = 0xFF;
00922     twi_Return_Value = bRetVal;
00923     fNextInterruptFunction = 0;
00924 }
00925
00926 uint8_t twii_WaitForDone(uint16_t timeout) {
00927     uint32_t endMillis = millis() + timeout;
00928     while (!twi_Done && (timeout == 0 || millis() < endMillis)) continue;
00929     return twi_Return_Value;
00930 }
00931
00932 void twii_SetState(uint8_t ucState) {
00933     twi_state = ucState;
00934 }
00935
00936 void twii_SetError(uint8_t ucError) {
00937     twi_error = ucError;
00938 }
00939
00940 void twii_InitBuffer(uint8_t ucPos, uint8_t ucLength) {
00941     twi_masterBufferIndex = 0;
00942     twi_masterBufferLength = ucLength;
00943 }
00944
00945 void twii_CopyToBuf(uint8_t* pData, uint8_t ucLength) {
00946     uint8_t i;
00947     for (i = 0; i < ucLength; ++i) {
00948         twi_masterBuffer[i] = pData[i];
00949     }
00950 }
00951
00952 void twii_CopyFromBuf(uint8_t *pData, uint8_t ucLength) {
00953     uint8_t i;
00954     for (i = 0; i < ucLength; ++i) {
00955         pData[i] = twi_masterBuffer[i];
00956     }
00957 }
00958
00959 void twii_SetSlaRW(uint8_t ucSlaRW) {
00960     twi_slarw = ucSlaRW;
00961 }
00962
00963 void twii_SetStart() {
00964     TWCN = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);
00965 }
00966
00967 void twi_write01() {
00968     if (TWI_MTX == twi_state) return; // blocking test
00969     twi_transmitting = 0;
00970     if (twi_error == 0xFF)
00971         twi_Finish(0); // success
00972     else if (twi_error == TW_MT_SLA_NACK)
00973         twi_Finish(2); // error: address send, nack received
00974     else if (twi_error == TW_MT_DATA_NACK)
00975         twi_Finish(3); // error: data send, nack received
00976     else
00977         twi_Finish(4); // other twi error
00978     if (twi_cbendTransmissionDone) return twi_cbendTransmissionDone(twi_Return_Value);
00979     return;
00980 }
00981
00982 void twi_write00() {
00983     if (TWI_READY != twi_state) return; // blocking test
00984     if (TWI_BUFFER_LENGTH < ptwv -> length) {
00985         twi_Finish(1); // end write with error 1
00986         return;
00987     }
00988     twi_Done = 0x00; // show as working
00989     twii_SetState(TWI_MTX); // to transmitting
00990     twii_SetError(0xFF); // to No Error
00991     twii_InitBuffer(0, ptwv -> length); // pointer and length
00992     twii_CopyToBuf(ptwv -> data, ptwv -> length); // get the data
00993     twii_SetSlaRW((ptwv -> address << 1) | TW_WRITE); // write command
00994     twii_SetStart(); // start the cycle
00995     fNextInterruptFunction = twi_write01; // next routine
00996     return twi_write01();
00997 }
00998
00999 void twi_writeTo(uint8_t address, uint8_t* data, uint8_t length, uint8_t wait) {
01000     uint8_t i;
01001     ptwv = (twi_Write_Vars *)malloc(sizeof(twi_Write_Vars));
01002     ptwv -> address = address;
01003     ptwv -> data = data;
01004     ptwv -> length = length;
01005     ptwv -> wait = wait;

```



```

01007         fNextInterruptFunction = twi_write00;
01008         return twi_write00();
01009     }
01010
01011     void twi_read01() {
01012         if (TWI_MR_X == twi_state) return; // blocking test
01013         if (twi_masterBufferIndex < ptwv -> length) ptwv -> length = twi_masterBufferIndex;
01014         twi_CopyFromBuf(ptwv -> data, ptwv -> length);
01015         twi_Finish(ptwv -> length);
01016         if (twi_cbreadFromDone) return twi_cbreadFromDone(twi_Return_Value);
01017         return;
01018     }
01019
01020     void twi_read00() {
01021         if (TWI_READY != twi_state) return; // blocking test
01022         if (TWI_BUFFER_LENGTH < ptwv -> length) twi_Finish(0); // error return
01023         twi_Done = 0x00; // show as working
01024         twi_SetState(TWI_MR_X); // reading
01025         twi_SetError(0xFF); // reset error
01026         twi_InitBuffer(0, ptwv -> length - 1); // init to one less than length
01027         twi_SetSlaRW((ptwv -> address << 1) | TW_READ); // read command
01028         twi_SetStart(); // start cycle
01029         fNextInterruptFunction = twi_read01;
01030         return twi_read01();
01031     }
01032
01033     void twi_readFrom(uint8_t address, uint8_t* data, uint8_t length) {
01034         uint8_t i;
01035
01036         ptwv = (twi_Write_Vars *)malloc(sizeof(twi_Write_Vars));
01037         ptwv -> address = address;
01038         ptwv -> data = data;
01039         ptwv -> length = length;
01040         fNextInterruptFunction = twi_read00;
01041         return twi_read00();
01042     }
01043
01044     void twi_reply(uint8_t ack) {
01045         // transmit master read ready signal, with or without ack
01046         if (ack) {
01047             TWC_R = _BV(TWEN) | _BV(TWIE) | _BV(TWINT) | _BV(TWEA);
01048         } else {
01049             TWC_R = _BV(TWEN) | _BV(TWIE) | _BV(TWINT);
01050         }
01051     }
01052
01053     void twi_stop(void) {
01054         // send stop condition
01055         TWC_R = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTO);
01056
01057         // wait for stop condition to be executed on bus
01058         // TWINT is not set after a stop condition!
01059         while (TWC_R & _BV(TWSTO)) {
01060             continue;
01061         }
01062
01063         // update twi state
01064         twi_state = TWI_READY;
01065     }
01066
01067     void twi_releaseBus(void) {
01068         // release bus
01069         TWC_R = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT);
01070
01071         // update twi state
01072         twi_state = TWI_READY;
01073     }
01074
01075     SIGNAL(TWI_vect) {
01076         switch (TW_STATUS) {
01077             // All Master
01078             case TW_START: // sent start condition
01079             case TW_REP_START: // sent repeated start condition
01080                 // copy device address and r/w bit to output register and ack
01081                 TWDR = twi_slarw;
01082                 twi_reply(1);
01083                 break;
01084
01085             // Master Transmitter
01086             case TW_MT_SLA_ACK: // slave receiver acked address
01087             case TW_MT_DATA_ACK: // slave receiver acked data
01088                 // if there is data to send, send it, otherwise stop
01089                 if (twi_masterBufferIndex < twi_masterBufferLength) {
01090                     // copy data to output register and ack
01091                     TWDR = twi_masterBuffer[twi_masterBufferIndex++];
01092                     twi_reply(1);
01093                 } else {

```

```

01094         twi_stop();
01095     }
01096     break;
01097
01098     case TW_MT_SLA_NACK: // address sent, nack received
01099         twi_error = TW_MT_SLA_NACK;
01100         twi_stop();
01101         break;
01102
01103     case TW_MT_DATA_NACK: // data sent, nack received
01104         twi_error = TW_MT_DATA_NACK;
01105         twi_stop();
01106         break;
01107
01108     case TW_MT_ARB_LOST: // lost bus arbitration
01109         twi_error = TW_MT_ARB_LOST;
01110         twi_releaseBus();
01111         break;
01112
01113     // Master Receiver
01114     case TW_MR_DATA_ACK: // data received, ack sent
01115         // put byte into buffer
01116         twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
01117
01118     case TW_MR_SLA_ACK: // address sent, ack received
01119         // ack if more bytes are expected, otherwise nack
01120         if (twi_masterBufferIndex < twi_masterBufferLength) {
01121             twi_reply(1);
01122         } else {
01123             twi_reply(0);
01124         }
01125         break;
01126
01127     case TW_MR_DATA_NACK: // data received, nack sent
01128         // put final byte into buffer
01129         twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
01130
01131     case TW_MR_SLA_NACK: // address sent, nack received
01132         twi_stop();
01133         break;
01134
01135     // TW_MR_ARB_LOST handled by TW_MT_ARB_LOST case
01136
01137     // Slave Receiver (NOT IMPLEMENTED YET)
01138     /*
01139     case TW_SR_SLA_ACK: // addressed, returned ack
01140     case TW_SR_GCALL_ACK: // addressed generally, returned ack
01141     case TW_SR_ARB_LOST_SLA_ACK: // lost arbitration, returned ack
01142     case TW_SR_ARB_LOST_GCALL_ACK: // lost arbitration, returned ack
01143         // enter slave receiver mode
01144         twi_state = TWI_SRX;
01145
01146         // indicate that rx buffer can be overwritten and ack
01147         twi_rxBufferIndex = 0;
01148         twi_reply(1);
01149         break;
01150
01151     case TW_SR_DATA_ACK: // data received, returned ack
01152     case TW_SR_GCALL_DATA_ACK: // data received generally, returned ack
01153         // if there is still room in the rx buffer
01154         if (twi_rxBufferIndex < TWI_BUFFER_LENGTH) {
01155             // put byte in buffer and ack
01156             twi_rxBuffer[twi_rxBufferIndex++] = TWDR;
01157             twi_reply(1);
01158         } else {
01159             // otherwise nack
01160             twi_reply(0);
01161         }
01162         break;
01163
01164     case TW_SR_STOP: // stop or repeated start condition received
01165         // put a null char after data if there's room
01166         if (twi_rxBufferIndex < TWI_BUFFER_LENGTH) {
01167             twi_rxBuffer[twi_rxBufferIndex] = 0;
01168         }
01169
01170         // sends ack and stops interface for clock stretching
01171         twi_stop();
01172
01173         // callback to user defined callback
01174         twi_onSlaveReceive(twi_rxBuffer, twi_rxBufferIndex);
01175
01176         // since we submit rx buffer to "wire" library, we can reset it
01177         twi_rxBufferIndex = 0;
01178
01179         // ack future responses and leave slave receiver state
01180         twi_releaseBus();

```

```

01181         break;
01182
01183     case TW_SR_DATA_NACK: // data received, returned nack
01184     case TW_SR_GCALL_DATA_NACK: // data received generally, returned nack
01185         // nack back at master
01186         twi_reply(0);
01187         break;
01188
01189     // Slave Transmitter
01190     case TW_ST_SLA_ACK: // addressed, returned ack
01191     case TW_ST_ARB_LOST_SLA_ACK: // arbitration lost, returned ack
01192         // enter slave transmitter mode
01193         twi_state = TWI_STX;
01194
01195         // ready the tx buffer index for iteration
01196         twi_txBufferIndex = 0;
01197
01198         // set tx buffer length to be zero, to verify if user changes it
01199         twi_txBufferLength = 0;
01200
01201         // request for txBuffer to be filled and length to be set
01202         // note: user must call twi_transmit(bytes, length) to do this
01203         twi_onSlaveTransmit();
01204
01205         // if they didn't change buffer & length, initialize it
01206         if (0 == twi_txBufferLength) {
01207             twi_txBufferLength = 1;
01208             twi_txBuffer[0] = 0x00;
01209         }
01210
01211         // transmit first byte from buffer, fall through
01212
01213     case TW_ST_DATA_ACK: // byte sent, ack returned
01214         // copy data to output register
01215         TWDR = twi_txBuffer[twi_txBufferIndex++];
01216
01217         // if there is more to send, ack, otherwise nack
01218         if (twi_txBufferIndex < twi_txBufferLength) {
01219             twi_reply(1);
01220         } else {
01221             twi_reply(0);
01222         }
01223         break;
01224
01225     case TW_ST_DATA_NACK: // received nack, we are done
01226     case TW_ST_LAST_DATA: // received ack, but we are done already!
01227         // ack future responses
01228         twi_reply(1);
01229         // leave slave receiver state
01230         twi_state = TWI_READY;
01231         break;
01232     */
01233
01234     // all
01235     case TW_NO_INFO: // no state information
01236         break;
01237
01238     case TW_BUS_ERROR: // bus error, illegal stop/start
01239         twi_error = TW_BUS_ERROR;
01240         twi_stop();
01241         break;
01242 }
01243
01244 if (fNextInterruptFunction) return fNextInterruptFunction();
01245 }
01246
01247 TwoWire::TwoWire() { }
01248
01249 void TwoWire::begin(void) {
01250     rxBufferIndex = 0;
01251     rxBufferLength = 0;
01252
01253     txBufferIndex = 0;
01254     txBufferLength = 0;
01255
01256     twi_init();
01257 }
01258
01259 void TwoWire::beginTransmission(uint8_t address) {
01260     //beginTransmission((uint8_t)address);
01261
01262     // indicate that we are transmitting
01263     twi_transmitting = 1;
01264
01265     // set address of targeted slave
01266     txAddress = address;
01267

```

```

01268         // reset tx buffer iterator vars
01269         txBufferIndex = 0;
01270         txBufferLength = 0;
01271     }
01272
01273     uint8_t TwoWire::endTransmission(uint16_t timeout) {
01274         // transmit buffer (blocking)
01275         //int8_t ret =
01276         twi_cbendTransmissionDone = NULL;
01277         twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
01278         int8_t ret = twii_WaitForDone(timeout);
01279
01280         // reset tx buffer iterator vars
01281         txBufferIndex = 0;
01282         txBufferLength = 0;
01283
01284         // indicate that we are done transmitting
01285         // twi_transmitting = 0;
01286         return ret;
01287     }
01288
01289     void TwoWire::nbendTransmission(void (*function)(int)) {
01290         twi_cbendTransmissionDone = function;
01291         twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
01292         return;
01293     }
01294
01295     void TwoWire::send(uint8_t data) {
01296         if (twi_transmitting) {
01297             // in master transmitter mode
01298             // don't bother if buffer is full
01299             if (txBufferLength >= NBWIRE_BUFFER_LENGTH) {
01300                 return;
01301             }
01302
01303             // put byte in tx buffer
01304             txBuffer[txBufferIndex] = data;
01305             ++txBufferIndex;
01306
01307             // update amount in buffer
01308             txBufferLength = txBufferIndex;
01309         } else {
01310             // in slave send mode
01311             // reply to master
01312             //twi_transmit(&data, 1);
01313         }
01314     }
01315
01316     uint8_t TwoWire::receive(void) {
01317         // default to returning null char
01318         // for people using with char strings
01319         uint8_t value = 0;
01320
01321         // get each successive byte on each call
01322         if (rxBufferIndex < rxBufferLength) {
01323             value = rxBuffer[rxBufferIndex];
01324             ++rxBufferIndex;
01325         }
01326
01327         return value;
01328     }
01329
01330     uint8_t TwoWire::requestFrom(uint8_t address, int quantity, uint16_t timeout) {
01331         // clamp to buffer length
01332         if (quantity > NBWIRE_BUFFER_LENGTH) {
01333             quantity = NBWIRE_BUFFER_LENGTH;
01334         }
01335
01336         // perform blocking read into buffer
01337         twi_cbreadFromDone = NULL;
01338         twi_readFrom(address, rxBuffer, quantity);
01339         uint8_t read = twii_WaitForDone(timeout);
01340
01341         // set rx buffer iterator vars
01342         rxBufferIndex = 0;
01343         rxBufferLength = read;
01344
01345         return read;
01346     }
01347
01348     void TwoWire::nbrequestFrom(uint8_t address, int quantity, void (*function)(int)) {
01349         // clamp to buffer length
01350         if (quantity > NBWIRE_BUFFER_LENGTH) {
01351             quantity = NBWIRE_BUFFER_LENGTH;
01352         }
01353
01354         // perform blocking read into buffer

```

```

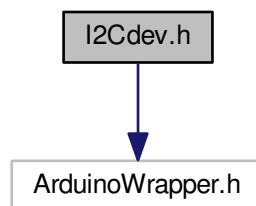
01355     twi_cbreadFromDone = function;
01356     twi_readFrom(address, rxBuffer, quantity);
01357     //uint8_t read = twii_WaitForDone();
01358
01359     // set rx buffer iterator vars
01360     //rxBufferIndex = 0;
01361     //rxBufferLength = read;
01362
01363     rxBufferIndex = 0;
01364     rxBufferLength = quantity; // this is a hack
01365
01366     return; //read;
01367 }
01368
01369 uint8_t TwoWire::available(void) {
01370     return rxBufferLength - rxBufferIndex;
01371 }
01372
01373 #endif

```

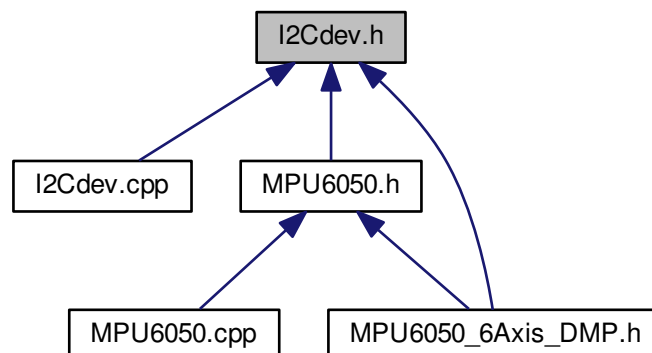
## 4.9 I2Cdev.h File Reference

#include "ArduinoWrapper.h"

Include dependency graph for I2Cdev.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [I2Cdev](#)

## Macros

- `#define I2CDEV_IMPLEMENTATION I2CDEV_ARDUINO_WIRE`
- `#define I2CDEV_IMPLEMENTATION_WARNINGS`
- `#define I2CDEV_ARDUINO_WIRE 1`
- `#define I2CDEV_BUILTIN_NBWIRE 2`
- `#define I2CDEV_BUILTIN_FASTWIRE 3`
- `#define I2CDEV_DEFAULT_READ_TIMEOUT 1000`

## 4.9.1 Macro Definition Documentation

4.9.1.1 `#define I2CDEV_ARDUINO_WIRE 1`

Definition at line 59 of file [I2Cdev.h](#).

4.9.1.2 `#define I2CDEV_BUILTIN_FASTWIRE 3`

Definition at line 62 of file [I2Cdev.h](#).

4.9.1.3 `#define I2CDEV_BUILTIN_NBWIRE 2`

Definition at line 60 of file [I2Cdev.h](#).

4.9.1.4 `#define I2CDEV_DEFAULT_READ_TIMEOUT 1000`

Definition at line 84 of file [I2Cdev.h](#).

4.9.1.5 `#define I2CDEV_IMPLEMENTATION I2CDEV_ARDUINO_WIRE`

Definition at line 50 of file [I2Cdev.h](#).

4.9.1.6 `#define I2CDEV_IMPLEMENTATION_WARNINGS`

Definition at line 54 of file [I2Cdev.h](#).

## 4.10 I2Cdev.h

```
00001 // I2Cdev library collection - Main I2C device class header file
00002 // Abstracts bit and byte I2C R/W functions into a convenient class
00003 // 6/9/2012 by Jeff Rowberg <jeff@rowberg.net>
00004 //
00005 // Changelog:
00006 // 2012-06-09 - fix major issue with reading > 32 bytes at a time with Arduino Wire
00007 // - add compiler warnings when using outdated or IDE or limited I2Cdev implementation
00008 // 2011-11-01 - fix write*Bits mask calculation (thanks sasquatch @ Arduino forums)
00009 // 2011-10-03 - added automatic Arduino version detection for ease of use
00010 // 2011-10-02 - added Gene Knight's NBWire TwoWire class implementation with small modifications
00011 // 2011-08-31 - added support for Arduino 1.0 Wire library (methods are different from 0.x)
00012 // 2011-08-03 - added optional timeout parameter to read* methods to easily change from default
00013 // 2011-08-02 - added support for 16-bit registers
00014 // - fixed incorrect Doxygen comments on some methods
00015 // - added timeout value for read operations (thanks mem @ Arduino forums)
00016 // 2011-07-30 - changed read/write function structures to return success or byte counts
00017 // - made all methods static for multi-device memory savings
00018 // 2011-07-28 - initial release
00019
00020 /* =====
00021 I2Cdev device library code is placed under the MIT license
00022 Copyright (c) 2012 Jeff Rowberg
00023
00024 Permission is hereby granted, free of charge, to any person obtaining a copy
```

```

00025 of this software and associated documentation files (the "Software"), to deal
00026 in the Software without restriction, including without limitation the rights
00027 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00028 copies of the Software, and to permit persons to whom the Software is
00029 furnished to do so, subject to the following conditions:
00030
00031 The above copyright notice and this permission notice shall be included in
00032 all copies or substantial portions of the Software.
00033
00034 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00035 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00036 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00037 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00038 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00039 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00040 THE SOFTWARE.
00041 =====
00042 */
00043
00044 #ifndef _I2CDEV_H_
00045 #define _I2CDEV_H_
00046
00047 // -----
00048 // I2C interface implementation setting
00049 // -----
00050 #define I2CDEV_IMPLEMENTATION          I2CDEV_ARDUINO_WIRE
00051
00052 // comment this out if you are using a non-optimal IDE/implementation setting
00053 // but want the compiler to shut up about it
00054 #define I2CDEV_IMPLEMENTATION_WARNINGS
00055
00056 // -----
00057 // I2C interface implementation options
00058 // -----
00059 #define I2CDEV_ARDUINO_WIRE            1 // Wire object from Arduino
00060 #define I2CDEV_BUILTIN_NBWIRE          2 // Tweaked Wire object from Gene Knight's NBWire project
00061 // ^^^ NBWire implementation is still buggy w/some interrupts!
00062 #define I2CDEV_BUILTIN_FASTWIRE        3 // FastWire object from Francesco Ferrara's project
00063 // ^^^ FastWire implementation in I2Cdev is INCOMPLETE!
00064
00065 // -----
00066 // Arduino-style "Serial.print" debug constant (uncomment to enable)
00067 // -----
00068 // #define I2CDEV_SERIAL_DEBUG
00069
00070 #ifdef ARDUINO
00071     #if ARDUINO < 100
00072         #include "WProgram.h"
00073     #else
00074         #include "Arduino.h"
00075     #endif
00076     #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
00077         #include <Wire.h>
00078     #endif
00079 #else
00080     #include "ArduinoWrapper.h"
00081 #endif
00082
00083 // 1000ms default read timeout (modify with "I2Cdev::readTimeout = [ms];")
00084 #define I2CDEV_DEFAULT_READ_TIMEOUT    1000
00085
00086 class I2Cdev {
00087     public:
00088         I2Cdev();
00089
00090         static int8_t readBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t *data,
00091                               uint16_t timeout=I2Cdev::readTimeout);
00092         static int8_t readBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t *data,
00093                                uint16_t timeout=I2Cdev::readTimeout);
00094         static int8_t readBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
00095                                uint8_t *data, uint16_t timeout=I2Cdev::readTimeout);
00096         static int8_t readBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
00097                                  , uint16_t *data, uint16_t timeout=I2Cdev::readTimeout);
00098         static int8_t readByte(uint8_t devAddr, uint8_t regAddr, uint8_t *data, uint16_t timeout=
00099                                I2Cdev::readTimeout);
00100         static int8_t readWord(uint8_t devAddr, uint8_t regAddr, uint16_t *data, uint16_t timeout=
00101                                I2Cdev::readTimeout);
00102         static int8_t readBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *data,
00103                                  uint16_t timeout=I2Cdev::readTimeout);
00104         static int8_t readWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t *data,
00105                                  uint16_t timeout=I2Cdev::readTimeout);
00106
00107         static bool writeBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t data);
00108         static bool writeBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t data);
00109         static bool writeBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
00110                                uint8_t data);
00111         static bool writeBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length

```

```

, uint16_t data);
00103     static bool writeByte(uint8_t devAddr, uint8_t regAddr, uint8_t data);
00104     static bool writeWord(uint8_t devAddr, uint8_t regAddr, uint16_t data);
00105     static bool writeBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *data);
00106     static bool writeWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t *data);
00107
00108     static uint16_t readTimeout;
00109 };
00110
00111 #if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
00112     // FastWire 0.2
00113     // This is a library to help faster programs to read I2C devices.
00114     // Copyright(C) 2011
00115     // Francesco Ferrara
00116
00117     /* Master */
00118     #define TW_START                0x08
00119     #define TW_REP_START            0x10
00120
00121     /* Master Transmitter */
00122     #define TW_MT_SLA_ACK           0x18
00123     #define TW_MT_SLA_NACK         0x20
00124     #define TW_MT_DATA_ACK         0x28
00125     #define TW_MT_DATA_NACK       0x30
00126     #define TW_MT_ARB_LOST         0x38
00127
00128     /* Master Receiver */
00129     #define TW_MR_ARB_LOST         0x38
00130     #define TW_MR_SLA_ACK         0x40
00131     #define TW_MR_SLA_NACK       0x48
00132     #define TW_MR_DATA_ACK       0x50
00133     #define TW_MR_DATA_NACK      0x58
00134
00135     #define TW_OK                   0
00136     #define TW_ERROR               1
00137
00138     class Fastwire {
00139     private:
00140         static boolean waitInt();
00141
00142     public:
00143         static void setup(int khz, boolean pullup);
00144         static byte write(byte device, byte address, byte value);
00145         static byte readBuf(byte device, byte address, byte *data, byte num);
00146     };
00147 #endif
00148
00149 #if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
00150     // NBWire implementation based heavily on code by Gene Knight <Gene@Telobot.com>
00151     // Originally posted on the Arduino forum at http://arduino.cc/forum/index.php/topic,70705.0.html
00152     // Originally offered to the i2cdevlib project at http://arduino.cc/forum/index.php/topic,68210.30.html
00153
00154     #define NBWIRE_BUFFER_LENGTH 32
00155
00156     class TwoWire {
00157     private:
00158         static uint8_t rxBuffer[];
00159         static uint8_t rxBufferIndex;
00160         static uint8_t rxBufferLength;
00161
00162         static uint8_t txAddress;
00163         static uint8_t txBuffer[];
00164         static uint8_t txBufferIndex;
00165         static uint8_t txBufferLength;
00166
00167         // static uint8_t transmitting;
00168         static void (*user_onRequest)(void);
00169         static void (*user_onReceive)(int);
00170         static void onRequestService(void);
00171         static void onReceiveService(uint8_t*, int);
00172
00173     public:
00174         TwoWire();
00175         void begin();
00176         void begin(uint8_t);
00177         void begin(int);
00178         void beginTransmission(uint8_t);
00179         //void beginTransmission(int);
00180         uint8_t endTransmission(uint16_t timeout=0);
00181         void nbendTransmission(void (*function)(int)) ;
00182         uint8_t requestFrom(uint8_t, int, uint16_t timeout=0);
00183         //uint8_t requestFrom(int, int);
00184         void nbrequestFrom(uint8_t, int, void (*function)(int));
00185         void send(uint8_t);
00186         void send(uint8_t*, uint8_t);
00187         //void send(int);
00188         void send(char*);
00189
00190

```



```

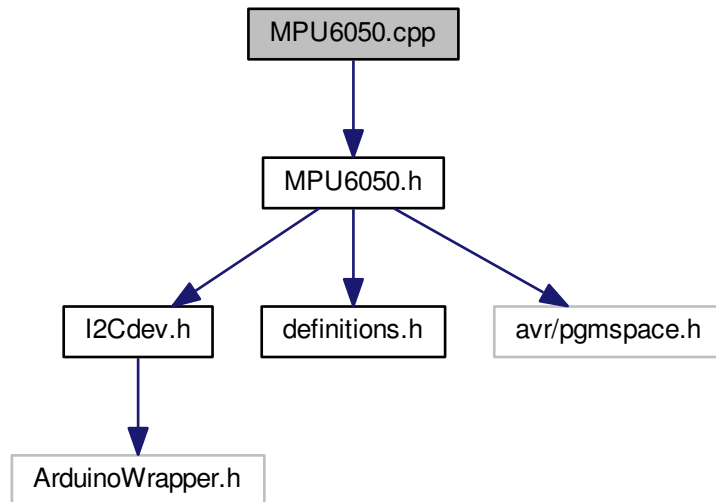
00191         uint8_t available(void);
00192         uint8_t receive(void);
00193         void onReceive(void (*)(int));
00194         void onRequest(void (*)(void));
00195     };
00196
00197     #define TWI_READY    0
00198     #define TWI_MRXL     1
00199     #define TWI_MTXL     2
00200     #define TWI_SRXL     3
00201     #define TWI_STXL     4
00202
00203     #define TW_WRITE     0
00204     #define TW_READ      1
00205
00206     #define TW_MT_SLA_NACK    0x20
00207     #define TW_MT_DATA_NACK   0x30
00208
00209     #define CPU_FREQ         16000000L
00210     #define TWI_FREQ         1000000L
00211     #define TWI_BUFFER_LENGTH 32
00212
00213     /* TWI Status is in TWSR, in the top 5 bits: TWS7 - TWS3 */
00214
00215     #define TW_STATUS_MASK    (_BV(TWS7) | _BV(TWS6) | _BV(TWS5) | _BV(TWS4) | _BV(TWS3))
00216     #define TW_STATUS        (TWSR & TW_STATUS_MASK)
00217     #define TW_START         0x08
00218     #define TW_REP_START     0x10
00219     #define TW_MT_SLA_ACK    0x18
00220     #define TW_MT_SLA_NACK   0x20
00221     #define TW_MT_DATA_ACK   0x28
00222     #define TW_MT_DATA_NACK  0x30
00223     #define TW_MT_ARB_LOST   0x38
00224     #define TW_MR_ARB_LOST   0x38
00225     #define TW_MR_SLA_ACK    0x40
00226     #define TW_MR_SLA_NACK   0x48
00227     #define TW_MR_DATA_ACK   0x50
00228     #define TW_MR_DATA_NACK  0x58
00229     #define TW_ST_SLA_ACK    0xA8
00230     #define TW_ST_ARB_LOST_SLA_ACK 0xB0
00231     #define TW_ST_DATA_ACK   0xB8
00232     #define TW_ST_DATA_NACK  0xC0
00233     #define TW_ST_LAST_DATA  0xC8
00234     #define TW_SR_SLA_ACK    0x60
00235     #define TW_SR_ARB_LOST_SLA_ACK 0x68
00236     #define TW_SR_GCALL_ACK  0x70
00237     #define TW_SR_ARB_LOST_GCALL_ACK 0x78
00238     #define TW_SR_DATA_ACK   0x80
00239     #define TW_SR_DATA_NACK  0x88
00240     #define TW_SR_GCALL_DATA_ACK 0x90
00241     #define TW_SR_GCALL_DATA_NACK 0x98
00242     #define TW_SR_STOP       0xA0
00243     #define TW_NO_INFO       0xF8
00244     #define TW_BUS_ERROR     0x00
00245
00246     // #define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *) (mem_addr))
00247     // #define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))
00248
00249     #ifndef sbi // set bit
00250         #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
00251     #endif // sbi
00252
00253     #ifndef cbi // clear bit
00254         #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
00255     #endif // cbi
00256
00257     extern TwoWire Wire;
00258
00259 #endif // I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
00260
00261 #endif /* _I2CDEV_H_ */

```

## 4.11 MPU6050.cpp File Reference

```
#include "MPU6050.h"
```

Include dependency graph for MPU6050.cpp:



## 4.12 MPU6050.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,
00088         MPU6050_TC_PWR_MODE_BIT, buffer);
00089     return buffer[0];
00090 }
00096 void MPU6050::setAuxVDDIOLevel(uint8_t level) {
00097     I2Cdev::writeBit(devAddr, MPU6050_RA_YG_OFFS_TC,
00098         MPU6050_TC_PWR_MODE_BIT, level);
00099 }
00100 // SMPLRT_DIV register
00101
00123 uint8_t MPU6050::getRate() {
00124     I2Cdev::readByte(devAddr, MPU6050_RA_SMPLRT_DIV,
00125         buffer);
00126     return buffer[0];
00127 }
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,
00167         MPU6050_CFG_EXT_SYNC_SET_BIT,
00168         MPU6050_CFG_EXT_SYNC_SET_LENGTH, buffer);
00169     return buffer[0];
00170 }
00174 void MPU6050::setExternalFrameSync(uint8_t sync) {
00175     I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG,
00176         MPU6050_CFG_EXT_SYNC_SET_BIT,
00177         MPU6050_CFG_EXT_SYNC_SET_LENGTH, sync);
00178 }
00205 uint8_t MPU6050::getDLPFMode() {
00206     I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG,
00207         MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH,
00208         buffer);
00209     return buffer[0];
00210 }
00217 void MPU6050::setDLPFMode(uint8_t mode) {
00218     I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG,
00219         MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH,
00220         mode);
00221 }
00222 // GYRO_CONFIG register
00223
00240 uint8_t MPU6050::getFullScaleGyroRange() {
00241     I2Cdev::readBits(devAddr, MPU6050_RA_GYRO_CONFIG,
00242         MPU6050_GCONFIG_FS_SEL_BIT,
00243         MPU6050_GCONFIG_FS_SEL_LENGTH, buffer);
00244     return buffer[0];
00245 }
00252 void MPU6050::setFullScaleGyroRange(uint8_t range) {
00253     I2Cdev::writeBits(devAddr, MPU6050_RA_GYRO_CONFIG,
00254         MPU6050_GCONFIG_FS_SEL_BIT,
00255         MPU6050_GCONFIG_FS_SEL_LENGTH, range);
00256 }
00257 // ACCEL_CONFIG register
00258
00262 bool MPU6050::getAccelXSelfTest() {
00263     I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
00264         MPU6050_ACONFIG_XA_ST_BIT, buffer);
00265     return buffer[0];
00266 }

```

```

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,
00535         buffer);
00536     return buffer[0];
00537 }
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,
00566         buffer);
00567     return buffer[0];
00568 }
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,
00587         MPU6050_TEMP_FIFO_EN_BIT, buffer);
00588     return buffer[0];
00589 }
00594 void MPU6050::setTempFIFOEnabled(bool enabled) {
00595     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
00596         MPU6050_TEMP_FIFO_EN_BIT, enabled);
00597 }
00603 bool MPU6050::getXGyroFIFOEnabled() {
00604     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
00605         MPU6050_XG_FIFO_EN_BIT, buffer);
00606     return buffer[0];
00607 }
00612 void MPU6050::setXGyroFIFOEnabled(bool enabled) {
00613     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
00614         MPU6050_XG_FIFO_EN_BIT, enabled);
00615 }
00621 bool MPU6050::getYGyroFIFOEnabled() {
00622     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
00623         MPU6050_YG_FIFO_EN_BIT, buffer);
00624     return buffer[0];
00625 }
00630 void MPU6050::setYGyroFIFOEnabled(bool enabled) {
00631     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
00632         MPU6050_YG_FIFO_EN_BIT, enabled);
00633 }
00639 bool MPU6050::getZGyroFIFOEnabled() {
00640     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
00641         MPU6050_ZG_FIFO_EN_BIT, buffer);
00642     return buffer[0];
00643 }
00648 void MPU6050::setZGyroFIFOEnabled(bool enabled) {
00649     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
00650         MPU6050_ZG_FIFO_EN_BIT, enabled);
00651 }
00658 bool MPU6050::getAccelFIFOEnabled() {
00659     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
00660         MPU6050_ACCEL_FIFO_EN_BIT, buffer);
00661     return buffer[0];
00662 }
00667 void MPU6050::setAccelFIFOEnabled(bool enabled) {
00668     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
00669         MPU6050_ACCEL_FIFO_EN_BIT, enabled);
00670 }
00676 bool MPU6050::getSlave2FIFOEnabled() {
00677     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
00678         MPU6050_SLV2_FIFO_EN_BIT, buffer);
00679     return buffer[0];
00680 }
00685 void MPU6050::setSlave2FIFOEnabled(bool enabled) {
00686     I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
00687         MPU6050_SLV2_FIFO_EN_BIT, enabled);
00688 }
00694 bool MPU6050::getSlave1FIFOEnabled() {
00695     I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
00696         MPU6050_SLV1_FIFO_EN_BIT, buffer);
00697     return buffer[0];
00698 }
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_SLV3_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_SLV3_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::getSlave4InputByte() {
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 }
01444 void MPU6050::setI2CBypassEnabled(bool enabled) {
01445     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_I2C_BYPASS_EN_BIT, enabled);
01446 }
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 }
02572 void MPU6050::setStandbyZAccelEnabled(bool enabled) {
02573     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02574         MPU6050_PWR2_STBY_ZA_BIT, enabled);
02575 }
02576 bool MPU6050::getStandbyXGyroEnabled() {
02577     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02578         MPU6050_PWR2_STBY_XG_BIT, buffer);
02579     return buffer[0];
02580 }
02581 void MPU6050::setStandbyXGyroEnabled(bool enabled) {
02582     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02583         MPU6050_PWR2_STBY_XG_BIT, enabled);
02584 }
02585 bool MPU6050::getStandbyYGyroEnabled() {
02586     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02587         MPU6050_PWR2_STBY_YG_BIT, buffer);
02588     return buffer[0];
02589 }
02590 void MPU6050::setStandbyYGyroEnabled(bool enabled) {
02591     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02592         MPU6050_PWR2_STBY_YG_BIT, enabled);
02593 }
02594 bool MPU6050::getStandbyZGyroEnabled() {
02595     I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02596         MPU6050_PWR2_STBY_ZG_BIT, buffer);
02597     return buffer[0];
02598 }
02599 void MPU6050::setStandbyZGyroEnabled(bool enabled) {
02600     I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
02601         MPU6050_PWR2_STBY_ZG_BIT, enabled);
02602 }
02603 // FIFO_COUNT* registers
02604 uint16_t MPU6050::getFIFOCount() {
02605     I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_COUNTH, 2,
02606         buffer);
02607     return ((uint16_t)buffer[0]) << 8 | buffer[1];
02608 }
02609 // FIFO_R_W register
02610 uint8_t MPU6050::getFIFOByte() {
02611     I2Cdev::readByte(devAddr, MPU6050_RA_FIFO_R_W,
02612         buffer);
02613     return buffer[0];
02614 }
02615 void MPU6050::getFIFOBytes(uint8_t *data, uint8_t length) {
02616     I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_R_W, length, data
02617 );
02618 }
02619 void MPU6050::setFIFOByte(uint8_t data) {
02620     I2Cdev::writeByte(devAddr, MPU6050_RA_FIFO_R_W, data);
02621 }
02622 // WHO_AM_I register
02623 uint8_t MPU6050::getDeviceID() {
02624     I2Cdev::readBits(devAddr, MPU6050_RA_WHO_AM_I,
02625         MPU6050_WHO_AM_I_BIT, MPU6050_WHO_AM_I_LENGTH,
02626         buffer);
02627     return buffer[0];
02628 }
02629 void MPU6050::setDeviceID(uint8_t id) {
02630     I2Cdev::writeBits(devAddr, MPU6050_RA_WHO_AM_I,
02631         MPU6050_WHO_AM_I_BIT, MPU6050_WHO_AM_I_LENGTH, id);
02632 }
02633 // ===== UNDOCUMENTED/DMP REGISTERS/METHODS =====
02634 // XG_OFFS_TC register
02635 uint8_t MPU6050::getOTPBankValid() {
02636     I2Cdev::readBit(devAddr, MPU6050_RA_XG_OFFS_TC,
02637         MPU6050_TC_OTP_BNK_VLD_BIT, buffer);
02638     return buffer[0];
02639 }
02640 void MPU6050::setOTPBankValid(bool enabled) {
02641     I2Cdev::writeBit(devAddr, MPU6050_RA_XG_OFFS_TC,
02642         MPU6050_TC_OTP_BNK_VLD_BIT, enabled);
02643 }
02644 int8_t MPU6050::getXGyroOffset() {
02645     I2Cdev::readBits(devAddr, MPU6050_RA_XG_OFFS_TC,
02646         MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH,
02647         buffer);

```

```

02734     return buffer[0];
02735 }
02736 void MPU6050::setXGyroOffset(int8_t offset) {
02737     I2Cdev::writeBits(devAddr, MPU6050_RA_XG_OFFS_TC,
02738         MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
02739 }
02740 // YG_OFFS_TC register
02741
02742 int8_t MPU6050::getYGyroOffset() {
02743     I2Cdev::readBits(devAddr, MPU6050_RA_YG_OFFS_TC,
02744         MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH,
02745         buffer);
02746     return buffer[0];
02747 }
02748 void MPU6050::setYGyroOffset(int8_t offset) {
02749     I2Cdev::writeBits(devAddr, MPU6050_RA_YG_OFFS_TC,
02750         MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
02751 }
02752 // ZG_OFFS_TC register
02753
02754 int8_t MPU6050::getZGyroOffset() {
02755     I2Cdev::readBits(devAddr, MPU6050_RA_ZG_OFFS_TC,
02756         MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH,
02757         buffer);
02758     return buffer[0];
02759 }
02760 void MPU6050::setZGyroOffset(int8_t offset) {
02761     I2Cdev::writeBits(devAddr, MPU6050_RA_ZG_OFFS_TC,
02762         MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
02763 }
02764 // X_FINE_GAIN register
02765
02766 int8_t MPU6050::getXFineGain() {
02767     I2Cdev::readByte(devAddr, MPU6050_RA_X_FINE_GAIN,
02768         buffer);
02769     return buffer[0];
02770 }
02771 void MPU6050::setXFineGain(int8_t gain) {
02772     I2Cdev::writeByte(devAddr, MPU6050_RA_X_FINE_GAIN, gain);
02773 }
02774 // Y_FINE_GAIN register
02775
02776 int8_t MPU6050::getYFineGain() {
02777     I2Cdev::readByte(devAddr, MPU6050_RA_Y_FINE_GAIN,
02778         buffer);
02779     return buffer[0];
02780 }
02781 void MPU6050::setYFineGain(int8_t gain) {
02782     I2Cdev::writeByte(devAddr, MPU6050_RA_Y_FINE_GAIN, gain);
02783 }
02784 // Z_FINE_GAIN register
02785
02786 int8_t MPU6050::getZFineGain() {
02787     I2Cdev::readByte(devAddr, MPU6050_RA_Z_FINE_GAIN,
02788         buffer);
02789     return buffer[0];
02790 }
02791 void MPU6050::setZFineGain(int8_t gain) {
02792     I2Cdev::writeByte(devAddr, MPU6050_RA_Z_FINE_GAIN, gain);
02793 }
02794 // XA_OFFS_* registers
02795
02796 int16_t MPU6050::getXAccelOffset() {
02797     I2Cdev::readBytes(devAddr, MPU6050_RA_XA_OFFS_H, 2,
02798         buffer);
02799     return (((int16_t)buffer[0]) << 8) | buffer[1];
02800 }
02801 void MPU6050::setXAccelOffset(int16_t offset) {
02802     I2Cdev::writeWord(devAddr, MPU6050_RA_XA_OFFS_H, offset);
02803 }
02804 // YA_OFFS_* register
02805
02806 int16_t MPU6050::getYAccelOffset() {
02807     I2Cdev::readBytes(devAddr, MPU6050_RA_YA_OFFS_H, 2,
02808         buffer);
02809     return (((int16_t)buffer[0]) << 8) | buffer[1];
02810 }
02811 void MPU6050::setYAccelOffset(int16_t offset) {
02812     I2Cdev::writeWord(devAddr, MPU6050_RA_YA_OFFS_H, offset);
02813 }
02814 }

```



```

02809
02810 // ZA_OFFS_* register
02811
02812 int16_t MPU6050::getZAccelOffset() {
02813     I2Cdev::readBytes(devAddr, MPU6050_RA_ZA_OFFS_H, 2,
02814         buffer);
02815     return (((int16_t)buffer[0]) << 8) | buffer[1];
02816 }
02817 void MPU6050::setZAccelOffset(int16_t offset) {
02818     I2Cdev::writeWord(devAddr, MPU6050_RA_ZA_OFFS_H, offset);
02819 }
02820 // XG_OFFS_USR* registers
02821
02822 int16_t MPU6050::getXGyroOffsetUser() {
02823     I2Cdev::readBytes(devAddr, MPU6050_RA_XG_OFFS_USRH, 2,
02824         buffer);
02825     return (((int16_t)buffer[0]) << 8) | buffer[1];
02826 }
02827 void MPU6050::setXGyroOffsetUser(int16_t offset) {
02828     I2Cdev::writeWord(devAddr, MPU6050_RA_XG_OFFS_USRH,
02829         offset);
02830 }
02831 // YG_OFFS_USR* register
02832
02833 int16_t MPU6050::getYGyroOffsetUser() {
02834     I2Cdev::readBytes(devAddr, MPU6050_RA_YG_OFFS_USRH, 2,
02835         buffer);
02836     return (((int16_t)buffer[0]) << 8) | buffer[1];
02837 }
02838 void MPU6050::setYGyroOffsetUser(int16_t offset) {
02839     I2Cdev::writeWord(devAddr, MPU6050_RA_YG_OFFS_USRH,
02840         offset);
02841 }
02842 // ZG_OFFS_USR* register
02843
02844 int16_t MPU6050::getZGyroOffsetUser() {
02845     I2Cdev::readBytes(devAddr, MPU6050_RA_ZG_OFFS_USRH, 2,
02846         buffer);
02847     return (((int16_t)buffer[0]) << 8) | buffer[1];
02848 }
02849 void MPU6050::setZGyroOffsetUser(int16_t offset) {
02850     I2Cdev::writeWord(devAddr, MPU6050_RA_ZG_OFFS_USRH,
02851         offset);
02852 }
02853 // INT_ENABLE register (DMP functions)
02854
02855 bool MPU6050::getIntPLLReadyEnabled() {
02856     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
02857         MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
02858     return buffer[0];
02859 }
02860 void MPU6050::setIntPLLReadyEnabled(bool enabled) {
02861     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
02862         MPU6050_INTERRUPT_PLL_RDY_INT_BIT, enabled);
02863 }
02864 bool MPU6050::getIntDMPEnabled() {
02865     I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
02866         MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
02867     return buffer[0];
02868 }
02869 void MPU6050::setIntDMPEnabled(bool enabled) {
02870     I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
02871         MPU6050_INTERRUPT_DMP_INT_BIT, enabled);
02872 }
02873 // DMP_INT_STATUS
02874
02875 bool MPU6050::getDMPInt5Status() {
02876     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
02877         MPU6050_DMPINT_5_BIT, buffer);
02878     return buffer[0];
02879 }
02880 bool MPU6050::getDMPInt4Status() {
02881     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
02882         MPU6050_DMPINT_4_BIT, buffer);
02883     return buffer[0];
02884 }
02885 bool MPU6050::getDMPInt3Status() {
02886     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
02887         MPU6050_DMPINT_3_BIT, buffer);
02888     return buffer[0];
02889 }
02890 bool MPU6050::getDMPInt2Status() {

```

```

02882     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
02883     MPU6050_DMPINT_2_BIT, buffer);
02884     return buffer[0];
02885 }
02886 bool MPU6050::getDMPInt1Status() {
02887     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
02888     MPU6050_DMPINT_1_BIT, buffer);
02889     return buffer[0];
02890 }
02891 bool MPU6050::getDMPInt0Status() {
02892     I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
02893     MPU6050_DMPINT_0_BIT, buffer);
02894     return buffer[0];
02895 }
02896 // INT_STATUS register (DMP functions)
02897 bool MPU6050::getIntPLLReadyStatus() {
02898     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
02899     MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
02900     return buffer[0];
02901 }
02902 bool MPU6050::getIntDMPStatus() {
02903     I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
02904     MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
02905     return buffer[0];
02906 }
02907 // USER_CTRL register (DMP functions)
02908 bool MPU6050::getDMPEnabled() {
02909     I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
02910     MPU6050_USERCTRL_DMP_EN_BIT, buffer);
02911     return buffer[0];
02912 }
02913 void MPU6050::setDMPEnabled(bool enabled) {
02914     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
02915     MPU6050_USERCTRL_DMP_EN_BIT, enabled);
02916 }
02917 void MPU6050::resetDMP() {
02918     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_DMP_RESET_BIT, true);
02919     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_DMP_RESET_BIT, true);
02920     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, 0x00, true);
02921     I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, 0x80 | 0x40
02922     | 0x08, true);
02923 }
02924 // BANK_SEL register
02925 void MPU6050::setMemoryBank(uint8_t bank, bool prefetchEnabled, bool userBank) {
02926     bank &= 0x1F;
02927     if (userBank) bank |= 0x20;
02928     if (prefetchEnabled) bank |= 0x40;
02929     I2Cdev::writeByte(devAddr, MPU6050_RA_BANK_SEL, bank);
02930 }
02931 // MEM_START_ADDR register
02932 void MPU6050::setMemoryStartAddress(uint8_t address) {
02933     I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_START_ADDR,
02934     address);
02935 }
02936 // MEM_R_W register
02937 uint8_t MPU6050::readMemoryByte() {
02938     I2Cdev::readByte(devAddr, MPU6050_RA_MEM_R_W,
02939     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,
02946     uint8_t address) {
02947     setMemoryBank(bank);
02948     setMemoryStartAddress(address);
02949     uint8_t chunkSize;
02950     for (uint16_t i = 0; i < dataSize; i) {
02951         // determine correct chunk size according to bank position and data size
02952         chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;
02953         // make sure we don't go past the data size
02954         if (i + chunkSize > dataSize) chunkSize = dataSize - i;
02955         // 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; i) {
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; i++) {
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 // DMP_CFG_1 register
03134 // DMP_CFG_1 register
03135 uint8_t MPU6050::getDMPConfig1() {
03136     I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_1,
buffer);
03137     return buffer[0];
03138 }
03139 void MPU6050::setDMPConfig1(uint8_t config) {
03140     I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_1, config);
03141 }
03142 // DMP_CFG_2 register
03143 // DMP_CFG_2 register
03144 uint8_t MPU6050::getDMPConfig2() {
03145     I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_2,
buffer);
03146     return buffer[0];
03147 }
03148 void MPU6050::setDMPConfig2(uint8_t config) {
03149     I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_2, config);
03150 }
03151 }
03152 }
03153 }
03154 }
03155 }
03156 }

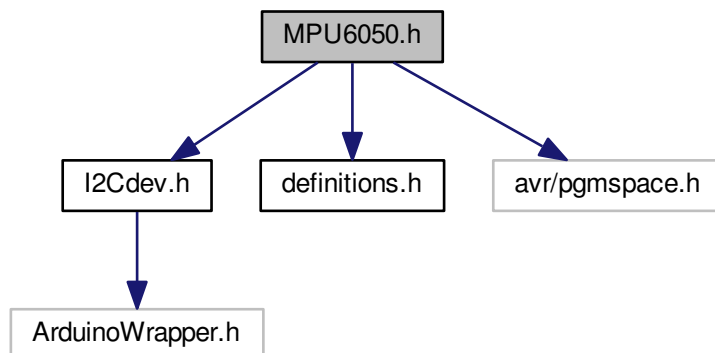
```

#### 4.13 MPU6050.h File Reference

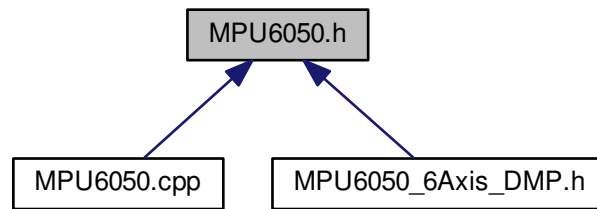
```

#include "I2Cdev.h"
#include "definitions.h"
#include <avr/pgmspace.h>
Include dependency graph for MPU6050.h:

```



This graph shows which files directly or indirectly include this file:



#### 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_SPLRT_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

#### 4.13.1 Macro Definition Documentation

##### 4.13.1.1 `#define MPU6050_ACCEL_FIFO_EN_BIT` 3

Definition at line 223 of file [MPU6050.h](#).

##### 4.13.1.2 `#define MPU6050_ACCEL_FS_16` 0x03

Definition at line 210 of file [MPU6050.h](#).

##### 4.13.1.3 `#define MPU6050_ACCEL_FS_2` 0x00

Definition at line 207 of file [MPU6050.h](#).

##### 4.13.1.4 `#define MPU6050_ACCEL_FS_4` 0x01

Definition at line 208 of file [MPU6050.h](#).

##### 4.13.1.5 `#define MPU6050_ACCEL_FS_8` 0x02

Definition at line 209 of file [MPU6050.h](#).

##### 4.13.1.6 `#define MPU6050_ACONFIG_ACCEL_HPF_BIT` 2

Definition at line 204 of file [MPU6050.h](#).

##### 4.13.1.7 `#define MPU6050_ACONFIG_ACCEL_HPF_LENGTH` 3

Definition at line 205 of file [MPU6050.h](#).

##### 4.13.1.8 `#define MPU6050_ACONFIG_AFS_SEL_BIT` 4

Definition at line 202 of file [MPU6050.h](#).

##### 4.13.1.9 `#define MPU6050_ACONFIG_AFS_SEL_LENGTH` 2

Definition at line 203 of file [MPU6050.h](#).

##### 4.13.1.10 `#define MPU6050_ACONFIG_XA_ST_BIT` 7

Definition at line 199 of file [MPU6050.h](#).

##### 4.13.1.11 `#define MPU6050_ACONFIG_YA_ST_BIT` 6

Definition at line 200 of file [MPU6050.h](#).

##### 4.13.1.12 `#define MPU6050_ACONFIG_ZA_ST_BIT` 5

Definition at line 201 of file [MPU6050.h](#).

##### 4.13.1.13 `#define MPU6050_BANKSEL_CFG_USER_BANK_BIT` 5

Definition at line 389 of file [MPU6050.h](#).

4.13.1.14 `#define MPU6050_BANKSEL_MEM_SEL_BIT 4`

Definition at line 390 of file [MPU6050.h](#).

4.13.1.15 `#define MPU6050_BANKSEL_MEM_SEL_LENGTH 5`

Definition at line 391 of file [MPU6050.h](#).

4.13.1.16 `#define MPU6050_BANKSEL_PRFTCH_EN_BIT 6`

Definition at line 388 of file [MPU6050.h](#).

4.13.1.17 `#define MPU6050_CFG_DLPF_CFG_BIT 2`

Definition at line 171 of file [MPU6050.h](#).

4.13.1.18 `#define MPU6050_CFG_DLPF_CFG_LENGTH 3`

Definition at line 172 of file [MPU6050.h](#).

4.13.1.19 `#define MPU6050_CFG_EXT_SYNC_SET_BIT 5`

Definition at line 169 of file [MPU6050.h](#).

4.13.1.20 `#define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3`

Definition at line 170 of file [MPU6050.h](#).

4.13.1.21 `#define MPU6050_CLOCK_DIV_258 0x8`

Definition at line 243 of file [MPU6050.h](#).

4.13.1.22 `#define MPU6050_CLOCK_DIV_267 0x7`

Definition at line 242 of file [MPU6050.h](#).

4.13.1.23 `#define MPU6050_CLOCK_DIV_276 0x6`

Definition at line 241 of file [MPU6050.h](#).

4.13.1.24 `#define MPU6050_CLOCK_DIV_286 0x5`

Definition at line 240 of file [MPU6050.h](#).

4.13.1.25 `#define MPU6050_CLOCK_DIV_296 0x4`

Definition at line 239 of file [MPU6050.h](#).

4.13.1.26 `#define MPU6050_CLOCK_DIV_308 0x3`

Definition at line 238 of file [MPU6050.h](#).

4.13.1.27 `#define MPU6050_CLOCK_DIV_320 0x2`

Definition at line 237 of file [MPU6050.h](#).

4.13.1.28 `#define MPU6050_CLOCK_DIV_333 0x1`

Definition at line 236 of file [MPU6050.h](#).

4.13.1.29 `#define MPU6050_CLOCK_DIV_348 0x0`

Definition at line 235 of file [MPU6050.h](#).

4.13.1.30 `#define MPU6050_CLOCK_DIV_364 0xF`

Definition at line 250 of file [MPU6050.h](#).

4.13.1.31 `#define MPU6050_CLOCK_DIV_381 0xE`

Definition at line 249 of file [MPU6050.h](#).

4.13.1.32 `#define MPU6050_CLOCK_DIV_400 0xD`

Definition at line 248 of file [MPU6050.h](#).

4.13.1.33 `#define MPU6050_CLOCK_DIV_421 0xC`

Definition at line 247 of file [MPU6050.h](#).

4.13.1.34 `#define MPU6050_CLOCK_DIV_444 0xB`

Definition at line 246 of file [MPU6050.h](#).

4.13.1.35 `#define MPU6050_CLOCK_DIV_471 0xA`

Definition at line 245 of file [MPU6050.h](#).

4.13.1.36 `#define MPU6050_CLOCK_DIV_500 0x9`

Definition at line 244 of file [MPU6050.h](#).

4.13.1.37 `#define MPU6050_CLOCK_INTERNAL 0x00`

Definition at line 366 of file [MPU6050.h](#).

4.13.1.38 `#define MPU6050_CLOCK_KEEP_RESET 0x07`

Definition at line 372 of file [MPU6050.h](#).

4.13.1.39 `#define MPU6050_CLOCK_PLL_EXT19M 0x05`

Definition at line 371 of file [MPU6050.h](#).

4.13.1.40 `#define MPU6050_CLOCK_PLL_EXT32K 0x04`

Definition at line 370 of file [MPU6050.h](#).

4.13.1.41 `#define MPU6050_CLOCK_PLL_XGYRO 0x01`

Definition at line 367 of file [MPU6050.h](#).

4.13.1.42 `#define MPU6050_CLOCK_PLL_YGYRO 0x02`

Definition at line 368 of file [MPU6050.h](#).

4.13.1.43 `#define MPU6050_CLOCK_PLL_ZGYRO 0x03`

Definition at line 369 of file [MPU6050.h](#).

4.13.1.44 `#define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT 7`

Definition at line 327 of file [MPU6050.h](#).

4.13.1.45 `#define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT 0`

Definition at line 332 of file [MPU6050.h](#).

4.13.1.46 `#define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT 1`

Definition at line 331 of file [MPU6050.h](#).

4.13.1.47 `#define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT 2`

Definition at line 330 of file [MPU6050.h](#).

4.13.1.48 `#define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT 3`

Definition at line 329 of file [MPU6050.h](#).

4.13.1.49 `#define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT 4`

Definition at line 328 of file [MPU6050.h](#).

4.13.1.50 `#define MPU6050_DETECT_ACCEL_ON_DELAY_BIT 5`

Definition at line 338 of file [MPU6050.h](#).

4.13.1.51 `#define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH 2`

Definition at line 339 of file [MPU6050.h](#).

4.13.1.52 `#define MPU6050_DETECT_DECREMENT_1 0x1`

Definition at line 346 of file [MPU6050.h](#).

4.13.1.53 `#define MPU6050_DETECT_DECREMENT_2 0x2`

Definition at line 347 of file [MPU6050.h](#).

4.13.1.54 `#define MPU6050_DETECT_DECREMENT_4 0x3`

Definition at line 348 of file [MPU6050.h](#).

4.13.1.55 `#define MPU6050_DETECT_DECREMENT_RESET 0x0`

Definition at line 345 of file [MPU6050.h](#).

4.13.1.56 `#define MPU6050_DETECT_FF_COUNT_BIT 3`

Definition at line 340 of file [MPU6050.h](#).

4.13.1.57 `#define MPU6050_DETECT_FF_COUNT_LENGTH 2`

Definition at line 341 of file [MPU6050.h](#).

4.13.1.58 `#define MPU6050_DETECT_MOT_COUNT_BIT 1`

Definition at line 342 of file [MPU6050.h](#).

4.13.1.59 `#define MPU6050_DETECT_MOT_COUNT_LENGTH 2`

Definition at line 343 of file [MPU6050.h](#).

4.13.1.60 `#define MPU6050_DHPF_0P63 0x04`

Definition at line 216 of file [MPU6050.h](#).

4.13.1.61 `#define MPU6050_DHPF_1P25 0x03`

Definition at line 215 of file [MPU6050.h](#).

4.13.1.62 `#define MPU6050_DHPF_2P5 0x02`

Definition at line 214 of file [MPU6050.h](#).

4.13.1.63 `#define MPU6050_DHPF_5 0x01`

Definition at line 213 of file [MPU6050.h](#).

4.13.1.64 `#define MPU6050_DHPF_HOLD 0x07`

Definition at line 217 of file [MPU6050.h](#).

4.13.1.65 `#define MPU6050_DHPF_RESET 0x00`

Definition at line 212 of file [MPU6050.h](#).

4.13.1.66 `#define MPU6050_DLPF_BW_10 0x05`

Definition at line 188 of file [MPU6050.h](#).

4.13.1.67 `#define MPU6050_DLPF_BW_188 0x01`

Definition at line 184 of file [MPU6050.h](#).

4.13.1.68 `#define MPU6050_DLPF_BW_20 0x04`

Definition at line 187 of file [MPU6050.h](#).

4.13.1.69 `#define MPU6050_DLPF_BW_256 0x00`

Definition at line 183 of file [MPU6050.h](#).

4.13.1.70 `#define MPU6050_DLPF_BW_42 0x03`

Definition at line 186 of file [MPU6050.h](#).

4.13.1.71 `#define MPU6050_DLPF_BW_5 0x06`

Definition at line 189 of file [MPU6050.h](#).

4.13.1.72 `#define MPU6050_DLPF_BW_98 0x02`

Definition at line 185 of file [MPU6050.h](#).

4.13.1.73 `#define MPU6050_DMP_MEMORY_BANK_SIZE 256`

Definition at line 397 of file [MPU6050.h](#).



4.13.1.74 `#define MPU6050_DMP_MEMORY_BANKS 8`

Definition at line 396 of file [MPU6050.h](#).

4.13.1.75 `#define MPU6050_DMP_MEMORY_CHUNK_SIZE 16`

Definition at line 398 of file [MPU6050.h](#).

4.13.1.76 `#define MPU6050_DMPINT_0_BIT 0`

Definition at line 317 of file [MPU6050.h](#).

4.13.1.77 `#define MPU6050_DMPINT_1_BIT 1`

Definition at line 316 of file [MPU6050.h](#).

4.13.1.78 `#define MPU6050_DMPINT_2_BIT 2`

Definition at line 315 of file [MPU6050.h](#).

4.13.1.79 `#define MPU6050_DMPINT_3_BIT 3`

Definition at line 314 of file [MPU6050.h](#).

4.13.1.80 `#define MPU6050_DMPINT_4_BIT 4`

Definition at line 313 of file [MPU6050.h](#).

4.13.1.81 `#define MPU6050_DMPINT_5_BIT 5`

Definition at line 312 of file [MPU6050.h](#).

4.13.1.82 `#define MPU6050_EXT_SYNC_ACCEL_XOUT_L 0x5`

Definition at line 179 of file [MPU6050.h](#).

4.13.1.83 `#define MPU6050_EXT_SYNC_ACCEL_YOUT_L 0x6`

Definition at line 180 of file [MPU6050.h](#).

4.13.1.84 `#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L 0x7`

Definition at line 181 of file [MPU6050.h](#).

4.13.1.85 `#define MPU6050_EXT_SYNC_DISABLED 0x0`

Definition at line 174 of file [MPU6050.h](#).

4.13.1.86 `#define MPU6050_EXT_SYNC_GYRO_XOUT_L 0x2`

Definition at line 176 of file [MPU6050.h](#).

4.13.1.87 `#define MPU6050_EXT_SYNC_GYRO_YOUT_L 0x3`

Definition at line 177 of file [MPU6050.h](#).

4.13.1.88 `#define MPU6050_EXT_SYNC_GYRO_ZOUT_L 0x4`

Definition at line 178 of file [MPU6050.h](#).

4.13.1.89 `#define MPU6050_EXT_SYNC_TEMP_OUT_L 0x1`

Definition at line 175 of file [MPU6050.h](#).

4.13.1.90 `#define MPU6050_GCONFIG_FS_SEL_BIT 4`

Definition at line 191 of file [MPU6050.h](#).

4.13.1.91 `#define MPU6050_GCONFIG_FS_SEL_LENGTH 2`

Definition at line 192 of file [MPU6050.h](#).

4.13.1.92 `#define MPU6050_GYRO_FS_1000 0x02`

Definition at line 196 of file [MPU6050.h](#).

4.13.1.93 `#define MPU6050_GYRO_FS_2000 0x03`

Definition at line 197 of file [MPU6050.h](#).

4.13.1.94 `#define MPU6050_GYRO_FS_250 0x00`

Definition at line 194 of file [MPU6050.h](#).

4.13.1.95 `#define MPU6050_GYRO_FS_500 0x01`

Definition at line 195 of file [MPU6050.h](#).

4.13.1.96 `#define MPU6050_I2C_MST_CLK_BIT 3`

Definition at line 232 of file [MPU6050.h](#).

4.13.1.97 `#define MPU6050_I2C_MST_CLK_LENGTH 4`

Definition at line 233 of file [MPU6050.h](#).

4.13.1.98 `#define MPU6050_I2C_MST_P_NSR_BIT 4`

Definition at line 231 of file [MPU6050.h](#).

4.13.1.99 `#define MPU6050_I2C_SLV4_ADDR_BIT 6`

Definition at line 263 of file [MPU6050.h](#).

4.13.1.100 `#define MPU6050_I2C_SLV4_ADDR_LENGTH 7`

Definition at line 264 of file [MPU6050.h](#).

4.13.1.101 `#define MPU6050_I2C_SLV4_EN_BIT 7`

Definition at line 265 of file [MPU6050.h](#).

4.13.1.102 `#define MPU6050_I2C_SLV4_INT_EN_BIT 6`

Definition at line 266 of file [MPU6050.h](#).

4.13.1.103 `#define MPU6050_I2C_SLV4_MST_DLY_BIT 4`

Definition at line 268 of file [MPU6050.h](#).

4.13.1.104 `#define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5`

Definition at line 269 of file [MPU6050.h](#).

4.13.1.105 `#define MPU6050_I2C_SLV4_REG_DIS_BIT 5`

Definition at line 267 of file [MPU6050.h](#).

4.13.1.106 `#define MPU6050_I2C_SLV4_RW_BIT 7`

Definition at line 262 of file [MPU6050.h](#).

4.13.1.107 `#define MPU6050_I2C_SLV_ADDR_BIT 6`

Definition at line 253 of file [MPU6050.h](#).

4.13.1.108 `#define MPU6050_I2C_SLV_ADDR_LENGTH 7`

Definition at line 254 of file [MPU6050.h](#).

4.13.1.109 `#define MPU6050_I2C_SLV_BYTE_SW_BIT 6`

Definition at line 256 of file [MPU6050.h](#).

4.13.1.110 `#define MPU6050_I2C_SLV_EN_BIT 7`

Definition at line 255 of file [MPU6050.h](#).

4.13.1.111 `#define MPU6050_I2C_SLV_GRP_BIT 4`

Definition at line 258 of file [MPU6050.h](#).

4.13.1.112 `#define MPU6050_I2C_SLV_LEN_BIT 3`

Definition at line 259 of file [MPU6050.h](#).

4.13.1.113 `#define MPU6050_I2C_SLV_LEN_LENGTH 4`

Definition at line 260 of file [MPU6050.h](#).

4.13.1.114 `#define MPU6050_I2C_SLV_REG_DIS_BIT 5`

Definition at line 257 of file [MPU6050.h](#).

4.13.1.115 `#define MPU6050_I2C_SLV_RW_BIT 7`

Definition at line 252 of file [MPU6050.h](#).

4.13.1.116 `#define MPU6050_INTCFG_CLKOUT_EN_BIT 0`

Definition at line 287 of file [MPU6050.h](#).

4.13.1.117 `#define MPU6050_INTCFG_FSYNC_INT_EN_BIT 2`

Definition at line 285 of file [MPU6050.h](#).

4.13.1.118 `#define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT 3`

Definition at line 284 of file [MPU6050.h](#).

4.13.1.119 `#define MPU6050_INTCFG_I2C_BYPASS_EN_BIT 1`

Definition at line 286 of file [MPU6050.h](#).

4.13.1.120 `#define MPU6050_INTCFG_INT_LEVEL_BIT 7`

Definition at line 280 of file [MPU6050.h](#).

4.13.1.121 `#define MPU6050_INTCFG_INT_OPEN_BIT 6`

Definition at line 281 of file [MPU6050.h](#).

4.13.1.122 `#define MPU6050_INTCFG_INT_RD_CLEAR_BIT 4`

Definition at line 283 of file [MPU6050.h](#).

4.13.1.123 `#define MPU6050_INTCFG_LATCH_INT_EN_BIT 5`

Definition at line 282 of file [MPU6050.h](#).

4.13.1.124 `#define MPU6050_INTCLEAR_ANYREAD 0x01`

Definition at line 299 of file [MPU6050.h](#).

4.13.1.125 `#define MPU6050_INTCLEAR_STATUSREAD 0x00`

Definition at line 298 of file [MPU6050.h](#).

4.13.1.126 `#define MPU6050_INTDRV_OPENDRAIN 0x01`

Definition at line 293 of file [MPU6050.h](#).

4.13.1.127 `#define MPU6050_INTDRV_PUSHPULL 0x00`

Definition at line 292 of file [MPU6050.h](#).

4.13.1.128 `#define MPU6050_INTERRUPT_DATA_RDY_BIT 0`

Definition at line 308 of file [MPU6050.h](#).

4.13.1.129 `#define MPU6050_INTERRUPT_DMP_INT_BIT 1`

Definition at line 307 of file [MPU6050.h](#).

4.13.1.130 `#define MPU6050_INTERRUPT_FF_BIT 7`

Definition at line 301 of file [MPU6050.h](#).

4.13.1.131 `#define MPU6050_INTERRUPT_FIFO_OFLOW_BIT 4`

Definition at line 304 of file [MPU6050.h](#).

4.13.1.132 `#define MPU6050_INTERRUPT_I2C_MST_INT_BIT 3`

Definition at line 305 of file [MPU6050.h](#).

4.13.1.133 `#define MPU6050_INTERRUPT_MOT_BIT 6`

Definition at line 302 of file [MPU6050.h](#).

4.13.1.134 `#define MPU6050_INTERRUPT_PLL_RDY_INT_BIT 2`

Definition at line 306 of file [MPU6050.h](#).

4.13.1.135 `#define MPU6050_INTERRUPT_ZMOT_BIT 5`

Definition at line 303 of file [MPU6050.h](#).

4.13.1.136 `#define MPU6050_INTLATCH_50USPULSE 0x00`

Definition at line 295 of file [MPU6050.h](#).

4.13.1.137 `#define MPU6050_INTLATCH_WAITCLEAR 0x01`

Definition at line 296 of file [MPU6050.h](#).

4.13.1.138 `#define MPU6050_INTMODE_ACTIVEHIGH 0x00`

Definition at line 289 of file [MPU6050.h](#).

4.13.1.139 `#define MPU6050_INTMODE_ACTIVELOW 0x01`

Definition at line 290 of file [MPU6050.h](#).

4.13.1.140 `#define MPU6050_MOTION_MOT_XNEG_BIT 7`

Definition at line 319 of file [MPU6050.h](#).

4.13.1.141 `#define MPU6050_MOTION_MOT_XPOS_BIT 6`

Definition at line 320 of file [MPU6050.h](#).

4.13.1.142 `#define MPU6050_MOTION_MOT_YNEG_BIT 5`

Definition at line 321 of file [MPU6050.h](#).

4.13.1.143 `#define MPU6050_MOTION_MOT_YPOS_BIT 4`

Definition at line 322 of file [MPU6050.h](#).

4.13.1.144 `#define MPU6050_MOTION_MOT_ZNEG_BIT 3`

Definition at line 323 of file [MPU6050.h](#).

4.13.1.145 `#define MPU6050_MOTION_MOT_ZPOS_BIT 2`

Definition at line 324 of file [MPU6050.h](#).

4.13.1.146 `#define MPU6050_MOTION_MOT_ZRMOT_BIT 0`

Definition at line 325 of file [MPU6050.h](#).

4.13.1.147 `#define MPU6050_MST_I2C_LOST_ARB_BIT 5`

Definition at line 273 of file [MPU6050.h](#).

4.13.1.148 `#define MPU6050_MST_I2C_SLV0_NACK_BIT 0`

Definition at line 278 of file [MPU6050.h](#).

4.13.1.149 `#define MPU6050_MST_I2C_SLV1_NACK_BIT 1`

Definition at line 277 of file [MPU6050.h](#).

4.13.1.150 `#define MPU6050_MST_I2C_SLV2_NACK_BIT 2`

Definition at line 276 of file [MPU6050.h](#).

4.13.1.151 `#define MPU6050_MST_I2C_SLV3_NACK_BIT 3`

Definition at line 275 of file [MPU6050.h](#).

4.13.1.152 `#define MPU6050_MST_I2C_SLV4_DONE_BIT 6`

Definition at line 272 of file [MPU6050.h](#).

4.13.1.153 `#define MPU6050_MST_I2C_SLV4_NACK_BIT 4`

Definition at line 274 of file [MPU6050.h](#).

4.13.1.154 `#define MPU6050_MST_PASS_THROUGH_BIT 7`

Definition at line 271 of file [MPU6050.h](#).

4.13.1.155 `#define MPU6050_MULT_MST_EN_BIT 7`

Definition at line 228 of file [MPU6050.h](#).

4.13.1.156 `#define MPU6050_PATHRESET_ACCEL_RESET_BIT 1`

Definition at line 335 of file [MPU6050.h](#).

4.13.1.157 `#define MPU6050_PATHRESET_GYRO_RESET_BIT 2`

Definition at line 334 of file [MPU6050.h](#).

4.13.1.158 `#define MPU6050_PATHRESET_TEMP_RESET_BIT 0`

Definition at line 336 of file [MPU6050.h](#).

4.13.1.159 `#define MPU6050_PWR1_CLKSEL_BIT 2`

Definition at line 363 of file [MPU6050.h](#).

4.13.1.160 `#define MPU6050_PWR1_CLKSEL_LENGTH 3`

Definition at line 364 of file [MPU6050.h](#).

4.13.1.161 `#define MPU6050_PWR1_CYCLE_BIT 5`

Definition at line 361 of file [MPU6050.h](#).

4.13.1.162 `#define MPU6050_PWR1_DEVICE_RESET_BIT 7`

Definition at line 359 of file [MPU6050.h](#).

4.13.1.163 `#define MPU6050_PWR1_SLEEP_BIT 6`

Definition at line 360 of file [MPU6050.h](#).

4.13.1.164 `#define MPU6050_PWR1_TEMP_DIS_BIT 3`

Definition at line 362 of file [MPU6050.h](#).

4.13.1.165 `#define MPU6050_PWR2_LP_WAKE_CTRL_BIT 7`

Definition at line 374 of file [MPU6050.h](#).

4.13.1.166 `#define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH 2`

Definition at line 375 of file [MPU6050.h](#).

4.13.1.167 `#define MPU6050_PWR2_STBY_XA_BIT 5`

Definition at line 376 of file [MPU6050.h](#).

4.13.1.168 `#define MPU6050_PWR2_STBY_XG_BIT 2`

Definition at line 379 of file [MPU6050.h](#).

4.13.1.169 `#define MPU6050_PWR2_STBY_YA_BIT 4`

Definition at line 377 of file [MPU6050.h](#).

4.13.1.170 `#define MPU6050_PWR2_STBY_YG_BIT 1`

Definition at line 380 of file [MPU6050.h](#).

4.13.1.171 `#define MPU6050_PWR2_STBY_ZA_BIT 3`

Definition at line 378 of file [MPU6050.h](#).

4.13.1.172 `#define MPU6050_PWR2_STBY_ZG_BIT 0`

Definition at line 381 of file [MPU6050.h](#).

4.13.1.173 `#define MPU6050_RA_ACCEL_CONFIG 0x1C`

Definition at line 71 of file [MPU6050.h](#).

4.13.1.174 `#define MPU6050_RA_ACCEL_XOUT_H 0x3B`

Definition at line 102 of file [MPU6050.h](#).

4.13.1.175 `#define MPU6050_RA_ACCEL_XOUT_L 0x3C`

Definition at line 103 of file [MPU6050.h](#).

4.13.1.176 `#define MPU6050_RA_ACCEL_YOUT_H 0x3D`

Definition at line 104 of file [MPU6050.h](#).

4.13.1.177 `#define MPU6050_RA_ACCEL_YOUT_L 0x3E`

Definition at line 105 of file [MPU6050.h](#).

4.13.1.178 `#define MPU6050_RA_ACCEL_ZOUT_H 0x3F`

Definition at line 106 of file [MPU6050.h](#).

4.13.1.179 `#define MPU6050_RA_ACCEL_ZOUT_L 0x40`

Definition at line 107 of file [MPU6050.h](#).

4.13.1.180 `#define MPU6050_RA_BANK_SEL 0x6D`

Definition at line 151 of file [MPU6050.h](#).

4.13.1.181 `#define MPU6050_RA_CONFIG 0x1A`

Definition at line 69 of file [MPU6050.h](#).

4.13.1.182 `#define MPU6050_RA_DMP_CFG_1 0x70`

Definition at line 154 of file [MPU6050.h](#).

4.13.1.183 `#define MPU6050_RA_DMP_CFG_2 0x71`

Definition at line 155 of file [MPU6050.h](#).

4.13.1.184 `#define MPU6050_RA_DMP_INT_STATUS 0x39`

Definition at line 100 of file [MPU6050.h](#).

4.13.1.185 `#define MPU6050_RA_EXT_SENS_DATA_00 0x49`

Definition at line 116 of file [MPU6050.h](#).

4.13.1.186 `#define MPU6050_RA_EXT_SENS_DATA_01 0x4A`

Definition at line 117 of file [MPU6050.h](#).

4.13.1.187 `#define MPU6050_RA_EXT_SENS_DATA_02 0x4B`

Definition at line 118 of file [MPU6050.h](#).

4.13.1.188 `#define MPU6050_RA_EXT_SENS_DATA_03 0x4C`

Definition at line 119 of file [MPU6050.h](#).

4.13.1.189 `#define MPU6050_RA_EXT_SENS_DATA_04 0x4D`

Definition at line 120 of file [MPU6050.h](#).

4.13.1.190 `#define MPU6050_RA_EXT_SENS_DATA_05 0x4E`

Definition at line 121 of file [MPU6050.h](#).

4.13.1.191 `#define MPU6050_RA_EXT_SENS_DATA_06 0x4F`

Definition at line 122 of file [MPU6050.h](#).

4.13.1.192 `#define MPU6050_RA_EXT_SENS_DATA_07 0x50`

Definition at line 123 of file [MPU6050.h](#).

4.13.1.193 `#define MPU6050_RA_EXT_SENS_DATA_08 0x51`

Definition at line 124 of file [MPU6050.h](#).



4.13.1.194 `#define MPU6050_RA_EXT_SENS_DATA_09 0x52`

Definition at line 125 of file [MPU6050.h](#).

4.13.1.195 `#define MPU6050_RA_EXT_SENS_DATA_10 0x53`

Definition at line 126 of file [MPU6050.h](#).

4.13.1.196 `#define MPU6050_RA_EXT_SENS_DATA_11 0x54`

Definition at line 127 of file [MPU6050.h](#).

4.13.1.197 `#define MPU6050_RA_EXT_SENS_DATA_12 0x55`

Definition at line 128 of file [MPU6050.h](#).

4.13.1.198 `#define MPU6050_RA_EXT_SENS_DATA_13 0x56`

Definition at line 129 of file [MPU6050.h](#).

4.13.1.199 `#define MPU6050_RA_EXT_SENS_DATA_14 0x57`

Definition at line 130 of file [MPU6050.h](#).

4.13.1.200 `#define MPU6050_RA_EXT_SENS_DATA_15 0x58`

Definition at line 131 of file [MPU6050.h](#).

4.13.1.201 `#define MPU6050_RA_EXT_SENS_DATA_16 0x59`

Definition at line 132 of file [MPU6050.h](#).

4.13.1.202 `#define MPU6050_RA_EXT_SENS_DATA_17 0x5A`

Definition at line 133 of file [MPU6050.h](#).

4.13.1.203 `#define MPU6050_RA_EXT_SENS_DATA_18 0x5B`

Definition at line 134 of file [MPU6050.h](#).

4.13.1.204 `#define MPU6050_RA_EXT_SENS_DATA_19 0x5C`

Definition at line 135 of file [MPU6050.h](#).

4.13.1.205 `#define MPU6050_RA_EXT_SENS_DATA_20 0x5D`

Definition at line 136 of file [MPU6050.h](#).

4.13.1.206 `#define MPU6050_RA_EXT_SENS_DATA_21 0x5E`

Definition at line 137 of file [MPU6050.h](#).

4.13.1.207 `#define MPU6050_RA_EXT_SENS_DATA_22 0x5F`

Definition at line 138 of file [MPU6050.h](#).

4.13.1.208 `#define MPU6050_RA_EXT_SENS_DATA_23 0x60`

Definition at line 139 of file [MPU6050.h](#).

4.13.1.209 `#define MPU6050_RA_FF_DUR 0x1E`

Definition at line 73 of file [MPU6050.h](#).

4.13.1.210 `#define MPU6050_RA_FF_THR 0x1D`

Definition at line 72 of file [MPU6050.h](#).

4.13.1.211 `#define MPU6050_RA_FIFO_COUNTH 0x72`

Definition at line 156 of file [MPU6050.h](#).

4.13.1.212 `#define MPU6050_RA_FIFO_COUNTL 0x73`

Definition at line 157 of file [MPU6050.h](#).

4.13.1.213 `#define MPU6050_RA_FIFO_EN 0x23`

Definition at line 78 of file [MPU6050.h](#).

4.13.1.214 `#define MPU6050_RA_FIFO_R_W 0x74`

Definition at line 158 of file [MPU6050.h](#).

4.13.1.215 `#define MPU6050_RA_GYRO_CONFIG 0x1B`

Definition at line 70 of file [MPU6050.h](#).

4.13.1.216 `#define MPU6050_RA_GYRO_XOUT_H 0x43`

Definition at line 110 of file [MPU6050.h](#).

4.13.1.217 `#define MPU6050_RA_GYRO_XOUT_L 0x44`

Definition at line 111 of file [MPU6050.h](#).

4.13.1.218 `#define MPU6050_RA_GYRO_YOUT_H 0x45`

Definition at line 112 of file [MPU6050.h](#).

4.13.1.219 `#define MPU6050_RA_GYRO_YOUT_L 0x46`

Definition at line 113 of file [MPU6050.h](#).

4.13.1.220 `#define MPU6050_RA_GYRO_ZOUT_H 0x47`

Definition at line 114 of file [MPU6050.h](#).

4.13.1.221 `#define MPU6050_RA_GYRO_ZOUT_L 0x48`

Definition at line 115 of file [MPU6050.h](#).

4.13.1.222 `#define MPU6050_RA_I2C_MST_CTRL 0x24`

Definition at line 79 of file [MPU6050.h](#).

4.13.1.223 `#define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67`

Definition at line 145 of file [MPU6050.h](#).

4.13.1.224 `#define MPU6050_RA_I2C_MST_STATUS 0x36`

Definition at line 97 of file [MPU6050.h](#).

4.13.1.225 `#define MPU6050_RA_I2C_SLV0_ADDR 0x25`

Definition at line 80 of file [MPU6050.h](#).

4.13.1.226 `#define MPU6050_RA_I2C_SLV0_CTRL 0x27`

Definition at line 82 of file [MPU6050.h](#).

4.13.1.227 `#define MPU6050_RA_I2C_SLV0_DO 0x63`

Definition at line 141 of file [MPU6050.h](#).

4.13.1.228 `#define MPU6050_RA_I2C_SLV0_REG 0x26`

Definition at line 81 of file [MPU6050.h](#).

4.13.1.229 `#define MPU6050_RA_I2C_SLV1_ADDR 0x28`

Definition at line 83 of file [MPU6050.h](#).

4.13.1.230 `#define MPU6050_RA_I2C_SLV1_CTRL 0x2A`

Definition at line 85 of file [MPU6050.h](#).

4.13.1.231 `#define MPU6050_RA_I2C_SLV1_DO 0x64`

Definition at line 142 of file [MPU6050.h](#).

4.13.1.232 `#define MPU6050_RA_I2C_SLV1_REG 0x29`

Definition at line 84 of file [MPU6050.h](#).

4.13.1.233 `#define MPU6050_RA_I2C_SLV2_ADDR 0x2B`

Definition at line 86 of file [MPU6050.h](#).

4.13.1.234 `#define MPU6050_RA_I2C_SLV2_CTRL 0x2D`

Definition at line 88 of file [MPU6050.h](#).

4.13.1.235 `#define MPU6050_RA_I2C_SLV2_DO 0x65`

Definition at line 143 of file [MPU6050.h](#).

4.13.1.236 `#define MPU6050_RA_I2C_SLV2_REG 0x2C`

Definition at line 87 of file [MPU6050.h](#).

4.13.1.237 `#define MPU6050_RA_I2C_SLV3_ADDR 0x2E`

Definition at line 89 of file [MPU6050.h](#).

4.13.1.238 `#define MPU6050_RA_I2C_SLV3_CTRL 0x30`

Definition at line 91 of file [MPU6050.h](#).

4.13.1.239 `#define MPU6050_RA_I2C_SLV3_DO 0x66`

Definition at line 144 of file [MPU6050.h](#).

4.13.1.240 `#define MPU6050_RA_I2C_SLV3_REG 0x2F`

Definition at line 90 of file [MPU6050.h](#).

4.13.1.241 `#define MPU6050_RA_I2C_SLV4_ADDR 0x31`

Definition at line 92 of file [MPU6050.h](#).

4.13.1.242 `#define MPU6050_RA_I2C_SLV4_CTRL 0x34`

Definition at line 95 of file [MPU6050.h](#).

4.13.1.243 `#define MPU6050_RA_I2C_SLV4_DI 0x35`

Definition at line 96 of file [MPU6050.h](#).

4.13.1.244 `#define MPU6050_RA_I2C_SLV4_DO 0x33`

Definition at line 94 of file [MPU6050.h](#).

4.13.1.245 `#define MPU6050_RA_I2C_SLV4_REG 0x32`

Definition at line 93 of file [MPU6050.h](#).

4.13.1.246 `#define MPU6050_RA_INT_ENABLE 0x38`

Definition at line 99 of file [MPU6050.h](#).

4.13.1.247 `#define MPU6050_RA_INT_PIN_CFG 0x37`

Definition at line 98 of file [MPU6050.h](#).

4.13.1.248 `#define MPU6050_RA_INT_STATUS 0x3A`

Definition at line 101 of file [MPU6050.h](#).

4.13.1.249 `#define MPU6050_RA_MEM_R_W 0x6F`

Definition at line 153 of file [MPU6050.h](#).

4.13.1.250 `#define MPU6050_RA_MEM_START_ADDR 0x6E`

Definition at line 152 of file [MPU6050.h](#).

4.13.1.251 `#define MPU6050_RA_MOT_DETECT_CTRL 0x69`

Definition at line 147 of file [MPU6050.h](#).

4.13.1.252 `#define MPU6050_RA_MOT_DETECT_STATUS 0x61`

Definition at line 140 of file [MPU6050.h](#).

4.13.1.253 `#define MPU6050_RA_MOT_DUR 0x20`

Definition at line 75 of file [MPU6050.h](#).

4.13.1.254 `#define MPU6050_RA_MOT_THR 0x1F`

Definition at line 74 of file [MPU6050.h](#).

4.13.1.255 `#define MPU6050_RA_PWR_MGMT_1 0x6B`

Definition at line 149 of file [MPU6050.h](#).

4.13.1.256 `#define MPU6050_RA_PWR_MGMT_2 0x6C`

Definition at line 150 of file [MPU6050.h](#).

4.13.1.257 `#define MPU6050_RA_SIGNAL_PATH_RESET 0x68`

Definition at line 146 of file [MPU6050.h](#).

4.13.1.258 `#define MPU6050_RA_SMPLRT_DIV 0x19`

Definition at line 68 of file [MPU6050.h](#).

4.13.1.259 `#define MPU6050_RA_TEMP_OUT_H 0x41`

Definition at line 108 of file [MPU6050.h](#).

4.13.1.260 `#define MPU6050_RA_TEMP_OUT_L 0x42`

Definition at line 109 of file [MPU6050.h](#).

4.13.1.261 `#define MPU6050_RA_USER_CTRL 0x6A`

Definition at line 148 of file [MPU6050.h](#).

4.13.1.262 `#define MPU6050_RA_WHO_AM_I 0x75`

Definition at line 159 of file [MPU6050.h](#).

4.13.1.263 `#define MPU6050_RA_X_FINE_GAIN 0x03`

Definition at line 53 of file [MPU6050.h](#).

4.13.1.264 `#define MPU6050_RA_XA_OFFS_H 0x06`

Definition at line 56 of file [MPU6050.h](#).

4.13.1.265 `#define MPU6050_RA_XA_OFFS_L_TC 0x07`

Definition at line 57 of file [MPU6050.h](#).

4.13.1.266 `#define MPU6050_RA_XG_OFFS_TC 0x00`

Definition at line 50 of file [MPU6050.h](#).

4.13.1.267 `#define MPU6050_RA_XG_OFFS_USRH 0x13`

Definition at line 62 of file [MPU6050.h](#).

4.13.1.268 `#define MPU6050_RA_XG_OFFS_USRL 0x14`

Definition at line 63 of file [MPU6050.h](#).

4.13.1.269 `#define MPU6050_RA_Y_FINE_GAIN 0x04`

Definition at line 54 of file [MPU6050.h](#).

4.13.1.270 `#define MPU6050_RA_YA_OFFS_H 0x08`

Definition at line 58 of file [MPU6050.h](#).

4.13.1.271 `#define MPU6050_RA_YA_OFFS_L_TC 0x09`

Definition at line 59 of file [MPU6050.h](#).

4.13.1.272 `#define MPU6050_RA_YG_OFFS_TC 0x01`

Definition at line 51 of file [MPU6050.h](#).

4.13.1.273 `#define MPU6050_RA_YG_OFFS_USRH 0x15`

Definition at line 64 of file [MPU6050.h](#).

4.13.1.274 `#define MPU6050_RA_YG_OFFS_USRL 0x16`

Definition at line 65 of file [MPU6050.h](#).

4.13.1.275 `#define MPU6050_RA_Z_FINE_GAIN 0x05`

Definition at line 55 of file [MPU6050.h](#).

4.13.1.276 `#define MPU6050_RA_ZA_OFFS_H 0x0A`

Definition at line 60 of file [MPU6050.h](#).

4.13.1.277 `#define MPU6050_RA_ZA_OFFS_L_TC 0x0B`

Definition at line 61 of file [MPU6050.h](#).

4.13.1.278 `#define MPU6050_RA_ZG_OFFS_TC 0x02`

Definition at line 52 of file [MPU6050.h](#).

4.13.1.279 `#define MPU6050_RA_ZG_OFFS_USRH 0x17`

Definition at line 66 of file [MPU6050.h](#).

4.13.1.280 `#define MPU6050_RA_ZG_OFFS_USRL 0x18`

Definition at line 67 of file [MPU6050.h](#).

4.13.1.281 `#define MPU6050_RA_ZRMOT_DUR 0x22`

Definition at line 77 of file [MPU6050.h](#).

4.13.1.282 `#define MPU6050_RA_ZRMOT_THR 0x21`

Definition at line 76 of file [MPU6050.h](#).

4.13.1.283 `#define MPU6050_SLV0_FIFO_EN_BIT 0`

Definition at line 226 of file [MPU6050.h](#).

4.13.1.284 `#define MPU6050_SLV1_FIFO_EN_BIT 1`

Definition at line 225 of file [MPU6050.h](#).

4.13.1.285 `#define MPU6050_SLV2_FIFO_EN_BIT 2`

Definition at line 224 of file [MPU6050.h](#).

4.13.1.286 `#define MPU6050_SLV_3_FIFO_EN_BIT 5`

Definition at line 230 of file [MPU6050.h](#).

4.13.1.287 `#define MPU6050_TC_OFFSET_BIT 6`

Definition at line 162 of file [MPU6050.h](#).

4.13.1.288 `#define MPU6050_TC_OFFSET_LENGTH 6`

Definition at line 163 of file [MPU6050.h](#).

4.13.1.289 `#define MPU6050_TC_OTP_BNK_VLD_BIT 0`

Definition at line 164 of file [MPU6050.h](#).

4.13.1.290 `#define MPU6050_TC_PWR_MODE_BIT 7`

Definition at line 161 of file [MPU6050.h](#).

4.13.1.291 `#define MPU6050_TEMP_FIFO_EN_BIT 7`

Definition at line 219 of file [MPU6050.h](#).

4.13.1.292 `#define MPU6050_USERCTRL_DMP_EN_BIT 7`

Definition at line 350 of file [MPU6050.h](#).

4.13.1.293 `#define MPU6050_USERCTRL_DMP_RESET_BIT 3`

Definition at line 354 of file [MPU6050.h](#).

4.13.1.294 `#define MPU6050_USERCTRL_FIFO_EN_BIT 6`

Definition at line 351 of file [MPU6050.h](#).

4.13.1.295 `#define MPU6050_USERCTRL_FIFO_RESET_BIT 2`

Definition at line 355 of file [MPU6050.h](#).

4.13.1.296 `#define MPU6050_USERCTRL_I2C_IF_DIS_BIT 4`

Definition at line 353 of file [MPU6050.h](#).

4.13.1.297 `#define MPU6050_USERCTRL_I2C_MST_EN_BIT 5`

Definition at line 352 of file [MPU6050.h](#).

4.13.1.298 `#define MPU6050_USERCTRL_I2C_MST_RESET_BIT 1`

Definition at line 356 of file [MPU6050.h](#).

4.13.1.299 `#define MPU6050_USERCTRL_SIG_COND_RESET_BIT 0`

Definition at line 357 of file [MPU6050.h](#).

4.13.1.300 `#define MPU6050_VDDIO_LEVEL_VDD 1`

Definition at line 167 of file [MPU6050.h](#).

4.13.1.301 `#define MPU6050_VDDIO_LEVEL_VLOGIC 0`

Definition at line 166 of file [MPU6050.h](#).

4.13.1.302 `#define MPU6050_WAIT_FOR_ES_BIT 6`

Definition at line 229 of file [MPU6050.h](#).

4.13.1.303 `#define MPU6050_WAKE_FREQ_10 0x3`

Definition at line 386 of file [MPU6050.h](#).

4.13.1.304 `#define MPU6050_WAKE_FREQ_1P25 0x0`

Definition at line 383 of file [MPU6050.h](#).

4.13.1.305 `#define MPU6050_WAKE_FREQ_2P5 0x1`

Definition at line 384 of file [MPU6050.h](#).

4.13.1.306 `#define MPU6050_WAKE_FREQ_5 0x2`

Definition at line 385 of file [MPU6050.h](#).

4.13.1.307 `#define MPU6050_WHO_AM_I_BIT 6`

Definition at line 393 of file [MPU6050.h](#).

4.13.1.308 `#define MPU6050_WHO_AM_I_LENGTH 6`

Definition at line 394 of file [MPU6050.h](#).

4.13.1.309 `#define MPU6050_XG_FIFO_EN_BIT 6`

Definition at line 220 of file [MPU6050.h](#).

4.13.1.310 `#define MPU6050_YG_FIFO_EN_BIT 5`

Definition at line 221 of file [MPU6050.h](#).

4.13.1.311 `#define MPU6050_ZG_FIFO_EN_BIT 4`

Definition at line 222 of file [MPU6050.h](#).

## 4.14 MPU6050.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
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 obvious
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 getSlave4InputByte();
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 getIntDMPEEnabled();
00738     void setIntDMPEEnabled(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 getDMPEEnabled();
00754     void setDMPEEnabled(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 writeDMPCConfigurationSet(const uint8_t *data, uint16_t dataSize, bool
useProgMem=false);
00771     bool writeProgDMPCConfigurationSet(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     #ifndef 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,
00939         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,
00943         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,
00946         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_ */

```

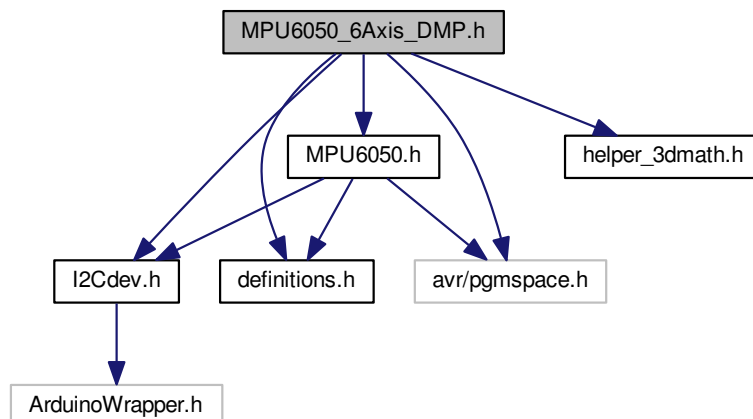
## 4.15 MPU6050\_6Axis\_DMP.h File Reference

```

#include "I2Cdev.h"
#include "helper_3dmath.h"
#include "definitions.h"
#include "MPU6050.h"
#include <avr/pgmspace.h>

```

Include dependency graph for MPU6050\_6Axis\_DMP.h:



## Macros

- `#define MPU6050_INCLUDE_DMP_MOTIONAPPS20`
- `#define MPU6050_DMP_CODE_SIZE 1929`
- `#define MPU6050_DMP_CONFIG_SIZE 174`
- `#define MPU6050_DMP_UPDATES_SIZE 47`

## Variables

- `const prog_uchar dmpMemory[MPU6050_DMP_CODE_SIZE] PROGMEM`

### 4.15.1 Macro Definition Documentation

#### 4.15.1.1 `#define MPU6050_DMP_CODE_SIZE 1929`

Definition at line 62 of file [MPU6050\\_6Axis\\_DMP.h](#).

#### 4.15.1.2 `#define MPU6050_DMP_CONFIG_SIZE 174`

Definition at line 63 of file [MPU6050\\_6Axis\\_DMP.h](#).

#### 4.15.1.3 `#define MPU6050_DMP_UPDATES_SIZE 47`

Definition at line 64 of file [MPU6050\\_6Axis\\_DMP.h](#).

#### 4.15.1.4 `#define MPU6050_INCLUDE_DMP_MOTIONAPPS20`

Definition at line 41 of file [MPU6050\\_6Axis\\_DMP.h](#).

### 4.15.2 Variable Documentation

#### 4.15.2.1 `const prog_uchar dmpUpdates [MPU6050_DMP_UPDATES_SIZE] PROGMEM`

Definition at line 79 of file [MPU6050\\_6Axis\\_DMP.h](#).

## 4.16 MPU6050\_6Axis\_DMP.h

```
00001 // I2Cdev library collection - MPU6050 I2C device class, 6-axis MotionApps 2.0 implementation
00002 // Based on InvenSense MPU-6050 register map document rev. 2.0, 5/19/2011 (RM-MPU-6000A-00)
00003 // 6/18/2012 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 /* =====
00010 I2Cdev device library code is placed under the MIT license
00011 Copyright (c) 2012 Jeff Rowberg
00012
00013 Permission is hereby granted, free of charge, to any person obtaining a copy
00014 of this software and associated documentation files (the "Software"), to deal
00015 in the Software without restriction, including without limitation the rights
00016 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00017 copies of the Software, and to permit persons to whom the Software is
00018 furnished to do so, subject to the following conditions:
00019
00020 The above copyright notice and this permission notice shall be included in
00021 all copies or substantial portions of the Software.
00022
00023 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00024 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00025 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00026 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00027 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00028 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00029 THE SOFTWARE.
```

Generated on Tue Aug 18 2015 22:51:26 for Arduino Brushless Gimbal Driver by Doxygen

```

00117 0x00, 0x00, 0x00, 0x00, 0x00, 0x65, 0x00, 0x54, 0xFF, 0xEF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00118 0x00, 0x00, 0x01, 0x00, 0x00, 0x44, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x01, 0x00,
00119 0x00, 0x00, 0x00, 0x00, 0x00, 0x65, 0x00, 0x00, 0x00, 0x00, 0x54, 0x00, 0x00, 0xFF, 0xEF, 0x00, 0x00,
00120 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00121 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00122 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00123 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00124 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00125 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00126 0x00, 0x1B, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00127 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00128 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00,
00129 0x00, 0x1B, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00130 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00131 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00132 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00133
00134 // bank 3, 256 bytes
00135 0xD8, 0xDC, 0xBA, 0xA2, 0xF1, 0xDE, 0xB2, 0xB8, 0xB4, 0xA8, 0x81, 0x91, 0xF7, 0x4A, 0x90, 0x7F,
00136 0x91, 0x6A, 0xF3, 0xF9, 0xDB, 0xA8, 0xF9, 0xB0, 0xBA, 0xA0, 0x80, 0xF2, 0xCE, 0x81, 0xF3, 0xC2,
00137 0xF1, 0xC1, 0xF2, 0xC3, 0xF3, 0xCC, 0xA2, 0xB2, 0x80, 0xF1, 0xC6, 0xD8, 0x80, 0xBA, 0xA7, 0xDF,
00138 0xDF, 0xDF, 0xF2, 0xA7, 0xC3, 0xCB, 0xC5, 0xB6, 0xF0, 0x87, 0xA2, 0x94, 0x24, 0x48, 0x70, 0x3C,
00139 0x95, 0x40, 0x68, 0x34, 0x58, 0x9B, 0x78, 0xA2, 0xF1, 0x83, 0x92, 0x2D, 0x55, 0x7D, 0xD8, 0xB1,
00140 0xB4, 0xB8, 0xA1, 0xD0, 0x91, 0x80, 0xF2, 0x70, 0xF3, 0x70, 0xF2, 0x7C, 0x80, 0xA8, 0xF1, 0x01,
00141 0xB0, 0x98, 0x87, 0xD9, 0x43, 0xD8, 0x86, 0xC9, 0x88, 0xBA, 0xA1, 0xF2, 0x0E, 0xB8, 0x97, 0x80,
00142 0xF1, 0xA9, 0xDF, 0xDF, 0xDF, 0xAA, 0xDF, 0xDF, 0xDF, 0xF2, 0xAA, 0xC5, 0xCD, 0xC7, 0xA9, 0xC0,
00143 0xC9, 0x2C, 0x97, 0x97, 0x97, 0x97, 0xF1, 0xA9, 0x89, 0x26, 0x46, 0x66, 0xB0, 0xB4, 0xBA, 0x80,
00144 0xAC, 0xDE, 0xF2, 0xCA, 0xF1, 0xB2, 0x8C, 0x02, 0xA9, 0xB6, 0x98, 0x00, 0x89, 0x0E, 0x16, 0x1E,
00145 0xB8, 0xA9, 0xB4, 0x99, 0x2C, 0x54, 0x7C, 0xB0, 0x8A, 0xA8, 0x96, 0x36, 0x56, 0x76, 0xF1, 0xB9,
00146 0xAF, 0xB4, 0xB0, 0x83, 0xC0, 0xB8, 0xA8, 0x97, 0x11, 0xB1, 0x8F, 0x98, 0xB9, 0xAF, 0xF0, 0x24,
00147 0x08, 0x44, 0x10, 0x64, 0x18, 0xF1, 0xA3, 0x29, 0x55, 0x7D, 0xAF, 0x83, 0xB5, 0x93, 0xAF, 0xF0,
00148 0x00, 0x28, 0x50, 0xF1, 0xA3, 0x86, 0x9F, 0x61, 0xA6, 0xDA, 0xDE, 0xDF, 0xD9, 0xFA, 0xA3, 0x86,
00149 0x96, 0xDB, 0x31, 0xA6, 0xD9, 0xF8, 0xDF, 0xBA, 0xA6, 0x8F, 0xC2, 0xC5, 0xC7, 0xB2, 0x8C, 0xC1,
00150 0xB8, 0xA2, 0xDF, 0xDF, 0xDF, 0xA3, 0xDF, 0xDF, 0xDF, 0xD8, 0xD8, 0xF1, 0xB8, 0xA8, 0xB2, 0x86,
00151
00152 // bank 4, 256 bytes
00153 0xB4, 0x98, 0xD0, 0x35, 0x5D, 0xB8, 0xAA, 0x98, 0xB0, 0x87, 0x2D, 0x35, 0x3D, 0xB2, 0xB6, 0xBA,
00154 0xAF, 0x8C, 0x96, 0x19, 0x8F, 0x9F, 0xA7, 0x0E, 0x16, 0x1E, 0xB4, 0x9A, 0xB8, 0xAA, 0x87, 0x2C,
00155 0x54, 0x7C, 0xB9, 0xA3, 0xDE, 0xDF, 0xDF, 0xA3, 0xB1, 0x80, 0xF2, 0xC4, 0xCD, 0xC9, 0xF1, 0xB8,
00156 0xA9, 0xB4, 0x99, 0x83, 0xD0, 0x35, 0x5D, 0x89, 0xB9, 0xA3, 0x2D, 0x55, 0x7D, 0xB5, 0x93, 0xA3,
00157 0x0E, 0x16, 0x1E, 0xA9, 0x2C, 0x54, 0x7C, 0xB8, 0xB4, 0xB0, 0xF1, 0x97, 0x83, 0xA8, 0x11, 0x84,
00158 0xA5, 0x09, 0x98, 0xA3, 0x83, 0xF0, 0xDA, 0x24, 0x08, 0x44, 0x10, 0x64, 0x18, 0xDA, 0xF1, 0xA5,
00159 0x29, 0x55, 0x7D, 0xA5, 0x85, 0x95, 0x02, 0x1A, 0x2E, 0x3A, 0x56, 0x5A, 0x40, 0x48, 0xF9, 0xF3,
00160 0xA3, 0xD9, 0xF8, 0xF0, 0x98, 0x83, 0x24, 0x08, 0x44, 0x10, 0x64, 0x18, 0xDA, 0xF3, 0xDE, 0xD8, 0x83, 0xA5,
00161 0x11, 0xF0, 0x98, 0xA2, 0x24, 0x08, 0x44, 0x10, 0x64, 0x18, 0xDA, 0xF3, 0xDE, 0xD8, 0x83, 0xA5,
00162 0x94, 0x01, 0xD9, 0xA3, 0x02, 0xF1, 0xA2, 0xC3, 0xC5, 0xC7, 0xD8, 0xF1, 0x84, 0x92, 0xA2, 0x4D,
00163 0xDA, 0x2A, 0xD8, 0x48, 0x69, 0xD9, 0x2A, 0xD8, 0x68, 0x55, 0xDA, 0x32, 0xD8, 0x50, 0x71, 0xD9,
00164 0x32, 0xD8, 0x70, 0x5D, 0xDA, 0x3A, 0xD8, 0x58, 0x79, 0xD9, 0x3A, 0xD8, 0x78, 0x93, 0xA3, 0x4D,
00165 0xDA, 0x2A, 0xD8, 0x48, 0x69, 0xD9, 0x2A, 0xD8, 0x68, 0x55, 0xDA, 0x32, 0xD8, 0x50, 0x71, 0xD9,
00166 0x32, 0xD8, 0x70, 0x5D, 0xDA, 0x3A, 0xD8, 0x58, 0x79, 0xD9, 0x3A, 0xD8, 0x78, 0xA8, 0x8A, 0x9A,
00167 0xF0, 0x28, 0x50, 0x78, 0x9E, 0xF3, 0x88, 0x18, 0xF1, 0x9F, 0x1D, 0x98, 0xA8, 0xD9, 0x08, 0xD8,
00168 0xC8, 0x9F, 0x12, 0x9E, 0xF3, 0x15, 0xA8, 0xDA, 0x12, 0x10, 0xD8, 0xF1, 0xAF, 0xC8, 0x97, 0x87,
00169
00170 // bank 5, 256 bytes
00171 0x34, 0xB5, 0xB9, 0x94, 0xA4, 0x21, 0xF3, 0xD9, 0x22, 0xD8, 0xF2, 0x2D, 0xF3, 0xD9, 0x2A, 0xD8,
00172 0xF2, 0x35, 0xF3, 0xD9, 0x32, 0xD8, 0x81, 0xA4, 0x60, 0x60, 0x61, 0xD9, 0x61, 0xD8, 0x6C, 0x68,
00173 0x69, 0xD9, 0x69, 0xD8, 0x74, 0x70, 0x71, 0xD9, 0x71, 0xD8, 0xB1, 0xA3, 0x84, 0x19, 0x3D, 0x5D,
00174 0xA3, 0x83, 0x1A, 0x3E, 0x5E, 0x93, 0x10, 0x30, 0x81, 0x10, 0x11, 0xB8, 0xB0, 0xAF, 0x8F, 0x94,
00175 0xF2, 0xDA, 0x3E, 0xD8, 0xB4, 0x9A, 0xA8, 0x87, 0x29, 0xDA, 0xF8, 0xD8, 0x87, 0x9A, 0x35, 0xDA,
00176 0xF8, 0xD8, 0x87, 0x9A, 0x3D, 0xDA, 0xF8, 0xD8, 0xB1, 0xB9, 0xA4, 0x98, 0x85, 0x02, 0x2E, 0x56,
00177 0xA5, 0x81, 0x00, 0x0C, 0x14, 0xA3, 0x97, 0xB0, 0x8A, 0xF1, 0x2D, 0xD9, 0x28, 0xD8, 0x4D, 0xD9,
00178 0x48, 0xD8, 0x6D, 0xD9, 0x68, 0xD8, 0xB1, 0x84, 0xD0, 0xDA, 0x0E, 0xD8, 0xA3, 0x29, 0x83, 0xDA,
00179 0x2C, 0x0E, 0xD8, 0xA3, 0x84, 0x49, 0x83, 0xDA, 0x2C, 0x4C, 0x0E, 0xD8, 0xB8, 0xB0, 0xA8, 0x8A,
00180 0x9A, 0xF5, 0x20, 0xAA, 0xDA, 0xDF, 0xD8, 0xA8, 0x40, 0xAA, 0xD0, 0xDA, 0xDE, 0xD8, 0xA8, 0x60,
00181 0xAA, 0xDA, 0xD0, 0xDF, 0xD8, 0xF1, 0x97, 0x86, 0xA8, 0x31, 0x9B, 0x06, 0x99, 0x07, 0xAB, 0x97,
00182 0x2A, 0x88, 0x9B, 0xF0, 0x0C, 0x20, 0x14, 0x40, 0xB8, 0xB0, 0xB4, 0xA8, 0x8C, 0x9C, 0xF0, 0x04,
00183 0x28, 0x51, 0x79, 0x1D, 0x30, 0x14, 0x38, 0xB2, 0x82, 0xAB, 0xD0, 0x98, 0x2C, 0x50, 0x50, 0x78,
00184 0x78, 0x9B, 0xF1, 0x1A, 0xB0, 0xF0, 0x8A, 0x9C, 0xA8, 0x29, 0x51, 0x79, 0x8B, 0x29, 0x51, 0x79,
00185 0x8A, 0x24, 0x70, 0x58, 0x8B, 0x20, 0x58, 0x71, 0x8A, 0x44, 0x69, 0x38, 0x8B, 0x39, 0x40, 0x68,
00186 0x8A, 0x64, 0x48, 0x31, 0x8B, 0x30, 0x49, 0x60, 0xA5, 0x88, 0x20, 0x09, 0x71, 0x58, 0x44, 0x68,
00187
00188 // bank 6, 256 bytes
00189 0x11, 0x39, 0x64, 0x49, 0x30, 0x19, 0xF1, 0xAC, 0x00, 0x2C, 0x54, 0x7C, 0xF0, 0x8C, 0xA8, 0x04,
00190 0x28, 0x50, 0x78, 0xF1, 0x88, 0x97, 0x26, 0xA8, 0x59, 0x98, 0xAC, 0x8C, 0x02, 0x26, 0x46, 0x66,
00191 0xF0, 0x89, 0x9C, 0xA8, 0x29, 0x51, 0x79, 0x24, 0x70, 0x59, 0x44, 0x69, 0x38, 0x64, 0x48, 0x31,
00192 0xA9, 0x88, 0x09, 0x20, 0x59, 0x70, 0xAB, 0x11, 0x38, 0x40, 0x69, 0xA8, 0x19, 0x31, 0x48, 0x60,
00193 0x8C, 0xA8, 0x3C, 0x41, 0x5C, 0x20, 0x7C, 0x00, 0xF1, 0x87, 0x98, 0x19, 0x86, 0xA8, 0x6E, 0x76,
00194 0x7E, 0xA9, 0x99, 0x88, 0x2D, 0x55, 0x7D, 0x9E, 0xB9, 0xA3, 0x8A, 0x22, 0x8A, 0x6E, 0x8A, 0x56,
00195 0x8A, 0x5E, 0x9F, 0xB1, 0x83, 0x06, 0x26, 0x46, 0x66, 0x0E, 0x2E, 0x4E, 0x6E, 0x9D, 0xB8, 0xAD,
00196 0x00, 0x2C, 0x54, 0x7C, 0xF2, 0xB1, 0x8C, 0xB4, 0x99, 0xB9, 0xA3, 0x2D, 0x55, 0x7D, 0x81, 0x91,
00197 0xAC, 0x38, 0xAD, 0x3A, 0xB5, 0x83, 0x91, 0xAC, 0x2D, 0xD9, 0x28, 0xD8, 0x4D, 0xD9, 0x48, 0xD8,
00198 0x6D, 0xD9, 0x68, 0xD8, 0x8C, 0x9D, 0xAE, 0x29, 0xD9, 0x04, 0xAE, 0xD8, 0x51, 0xD9, 0x04, 0xAE,
00199 0xD8, 0x79, 0xD9, 0x04, 0xD8, 0x81, 0xF3, 0x9D, 0xAD, 0x00, 0x8D, 0xAE, 0x19, 0x81, 0xAD, 0xD9,
00200 0x01, 0xD8, 0xF2, 0xAE, 0xDA, 0x26, 0xD8, 0x8E, 0x91, 0x29, 0x83, 0xA7, 0xD9, 0xAD, 0xAD, 0xAD,
00201 0xAD, 0xF3, 0x2A, 0xD8, 0xD8, 0xF1, 0xB0, 0xAC, 0x89, 0x91, 0x3E, 0x5E, 0x76, 0xF3, 0xAC, 0x2E,
00202 0x2E, 0xF1, 0xB1, 0x8C, 0x5A, 0x9C, 0xAC, 0x2C, 0x28, 0x28, 0x28, 0x9C, 0xAC, 0x30, 0x18, 0xA8,
00203 0x98, 0x81, 0x28, 0x34, 0x3C, 0x97, 0x24, 0xA7, 0x28, 0x34, 0x3C, 0x9C, 0x24, 0xF2, 0xB0, 0x89,

```



```

00204     0xAC, 0x91, 0x2C, 0x4C, 0x6C, 0x8A, 0x9B, 0x2D, 0xD9, 0xD8, 0xD8, 0x51, 0xD9, 0xD8, 0xD8, 0x79,
00205
00206     // bank 7, 138 bytes (remainder)
00207     0xD9, 0xD8, 0xD8, 0xF1, 0x9E, 0x88, 0xA3, 0x31, 0xDA, 0xD8, 0xD8, 0x91, 0x2D, 0xD9, 0x28, 0xD8,
00208     0x4D, 0xD9, 0x48, 0xD8, 0x6D, 0xD9, 0x68, 0xD8, 0xB1, 0x83, 0x93, 0x35, 0x3D, 0x80, 0x25, 0xDA,
00209     0xD8, 0xD8, 0x85, 0x69, 0xDA, 0xD8, 0xD8, 0xB4, 0x93, 0x81, 0xA3, 0x28, 0x34, 0x3C, 0xF3, 0xAB,
00210     0x8B, 0xF8, 0xA3, 0x91, 0xB6, 0x09, 0xB4, 0xD9, 0xAB, 0xDE, 0xFA, 0xB0, 0x87, 0x9C, 0xB9, 0xA3,
00211     0xDD, 0xF1, 0xA3, 0xA3, 0xA3, 0xA3, 0x95, 0xF1, 0xA3, 0xA3, 0x9D, 0xF1, 0xA3, 0xA3, 0xA3,
00212     0xA3, 0xF2, 0xA3, 0xB4, 0x90, 0x80, 0xF2, 0xA3, 0xA3, 0xA3, 0xA3, 0xA3, 0xA3, 0xA3, 0xA3,
00213     0xA3, 0xB2, 0xA3, 0xA3, 0xA3, 0xA3, 0xA3, 0xA3, 0xB0, 0x87, 0xB5, 0x99, 0xF1, 0xA3, 0xA3, 0xA3,
00214     0x98, 0xF1, 0xA3, 0xA3, 0xA3, 0xA3, 0x97, 0xA3, 0xA3, 0xA3, 0xA3, 0xF3, 0x9B, 0xA3, 0xA3, 0xDC,
00215     0xB9, 0xA7, 0xF1, 0x26, 0x26, 0x26, 0xD8, 0xD8, 0xFF
00216 };
00217
00218 // thanks to Noah Zerkon for piecing this stuff together!
00219 const prog_uchar dmpConfig[MPU6050_DMP_CONFIG_SIZE]
00220     PROGMEM = {
00221     // BANK   OFFSET   LENGTH   [DATA]
00222     0x03,    0x7B,    0x03,    0x4C, 0xCD, 0x6C,           // FCFG_1 inv_set_gyro_calibration
00223     0x03,    0xAB,    0x03,    0x36, 0x56, 0x76,           // FCFG_3 inv_set_gyro_calibration
00224     0x00,    0x68,    0x04,    0x02, 0xCB, 0x47, 0xA2,     // D_0_104 inv_set_gyro_calibration
00225     0x02,    0x18,    0x04,    0x00, 0x05, 0x8B, 0xC1,     // D_0_24 inv_set_gyro_calibration
00226     0x01,    0x0C,    0x04,    0x00, 0x00, 0x00, 0x00,     // D_1_152 inv_set_accel_calibration
00227     0x03,    0x7F,    0x06,    0x0C, 0xC9, 0x2C, 0x97, 0x97, // FCFG_2 inv_set_accel_calibration
00228     0x03,    0x89,    0x03,    0x26, 0x46, 0x66,           // FCFG_7 inv_set_accel_calibration
00229     0x00,    0x6C,    0x02,    0x20, 0x00,               // D_0_108 inv_set_accel_calibration
00230     0x02,    0x40,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_00 inv_set_compass_calibration
00231     0x02,    0x44,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_01
00232     0x02,    0x48,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_02
00233     0x02,    0x4C,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_10
00234     0x02,    0x50,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_11
00235     0x02,    0x54,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_12
00236     0x02,    0x58,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_20
00237     0x02,    0x5C,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_21
00238     0x02,    0xBC,    0x04,    0x00, 0x00, 0x00, 0x00,     // CPASS_MTX_22
00239     0x01,    0xEC,    0x04,    0x00, 0x00, 0x40, 0x00,     // D_1_236 inv_apply_endian_accel
00240     0x03,    0x7F,    0x06,    0x0C, 0xC9, 0x2C, 0x97, 0x97, // FCFG_2 inv_set_mpu_sensors
00241     0x04,    0x02,    0x03,    0x0D, 0x35, 0x5D,         // CFG_MOTION_BIAS inv_turn_on_bias_from_no_motion
00242     0x04,    0x09,    0x04,    0x87, 0x2D, 0x35, 0x3D,     // FCFG_5 inv_set_bias_update
00243     0x00,    0xA3,    0x01,    0x00,                     // D_0_163 inv_set_dead_zone
00244     0x00,    0x00,    0x00,    0x01,                     // SPECIAL 0x01 = enable interrupts
00245     0x07,    0x86,    0x01,    0xFE,                     // SET INT_ENABLE at i=22, SPECIAL INSTRUCTION
00246     0x07,    0x41,    0x05,    0xF1, 0x20, 0x28, 0x30, 0x38, // CFG_8 inv_send_quaternion
00247     0x07,    0x7E,    0x01,    0x30,                     // CFG_16 inv_set_footer
00248     // 0x07,    0x46,    0x01,    0x9A,                 // CFG_GYRO_SOURCE inv_send_gyro
00249     // 0x07,    0x47,    0x04,    0xF1, 0x28, 0x30, 0x38,     // CFG_9 inv_send_gyro -> inv_construct3_fifo
00250     // 0x07,    0x6C,    0x04,    0xF1, 0x28, 0x30, 0x38,     // CFG_12 inv_send_accel -> inv_construct3_fifo
00251     0x02,    0x16,    0x02,    0x00, 0x07                 // D_0_22 inv_set_fifo_rate
00252     // This very last 0x01 WAS a 0x09, which drops the FIFO rate down to 20 Hz. 0x07 is 25 Hz,
00253     // 0x01 is 100Hz. Going faster than 100Hz (0x00=200Hz) tends to result in very noisy data.
00254     // DMP output frequency is calculated easily using this equation: (200Hz / (1 + value))
00255
00256     // It is important to make sure the host processor can keep up with reading and processing
00257     // the FIFO output at the desired rate. Handling FIFO overflow cleanly is also a good idea.
00258 };
00259
00260 const prog_uchar dmpUpdates[MPU6050_DMP_UPDATES_SIZE]
00261     PROGMEM = {
00262     0x01,    0xB2,    0x02,    0xFF, 0xFF,
00263     0x01,    0x90,    0x04,    0x09, 0x23, 0xA1, 0x35,
00264     0x01,    0x6A,    0x02,    0x06, 0x00,
00265     0x01,    0x60,    0x08,    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
00266     0x00,    0x60,    0x04,    0x40, 0x00, 0x00, 0x00,
00267     0x01,    0x62,    0x02,    0x00, 0x00,
00268     0x00,    0x60,    0x04,    0x00, 0x40, 0x00, 0x00
00269 };
00270
00271 uint8_t MPU6050::dmpInitialize() {
00272     // reset device
00273     DEBUG_PRINTLN(F("\n\nResetting MPU6050..."));
00274     reset();
00275     delay(30); // wait after reset
00276
00277     // enable sleep mode and wake cycle
00278     /*Serial.println(F("Enabling sleep mode..."));
00279     setSleepEnabled(true);
00280     Serial.println(F("Enabling wake cycle..."));
00281     setWakeCycleEnabled(true);*/
00282
00283     // disable sleep mode
00284     DEBUG_PRINTLN(F("Disabling sleep mode..."));
00285     setSleepEnabled(false);
00286
00287     // get MPU hardware revision
00288     DEBUG_PRINTLN(F("Selecting user bank 16..."));
00289     setMemoryBank(0x10, true, true);

```



```

00289     DEBUG_PRINTLN(F("Selecting memory byte 6..."));
00290     setMemoryStartAddress(0x06);
00291     DEBUG_PRINTLN(F("Checking hardware revision..."));
00292     uint8_t hwRevision = readMemoryByte();
00293     DEBUG_PRINT(F("Revision @ user[16][6] = "));
00294     DEBUG_PRINTLN(F(hwRevision, HEX));
00295     DEBUG_PRINTLN(F("Resetting memory bank selection to 0..."));
00296     setMemoryBank(0, false, false);
00297
00298     // check OTP bank valid
00299     DEBUG_PRINTLN(F("Reading OTP bank valid flag..."));
00300     uint8_t otpValid = getOTPBANKValid();
00301     DEBUG_PRINT(F("OTP bank is "));
00302     DEBUG_PRINTLN(otpValid ? F("valid!") : F("invalid!"));
00303
00304     // get X/Y/Z gyro offsets
00305     DEBUG_PRINTLN(F("Reading gyro offset values..."));
00306     int8_t xgOffset = getXGyroOffset();
00307     int8_t ygOffset = getYGyroOffset();
00308     int8_t zgOffset = getZGyroOffset();
00309     DEBUG_PRINT(F("X gyro offset = "));
00310     DEBUG_PRINTLN(xgOffset);
00311     DEBUG_PRINT(F("Y gyro offset = "));
00312     DEBUG_PRINTLN(ygOffset);
00313     DEBUG_PRINT(F("Z gyro offset = "));
00314     DEBUG_PRINTLN(zgOffset);
00315
00316     // setup weird slave stuff (?)
00317     DEBUG_PRINTLN(F("Setting slave 0 address to 0x7F..."));
00318     setSlaveAddress(0, 0x7F);
00319     DEBUG_PRINTLN(F("Disabling I2C Master mode..."));
00320     setI2CMasterModeEnabled(false);
00321     DEBUG_PRINTLN(F("Setting slave 0 address to 0x68 (self)..."));
00322     setSlaveAddress(0, 0x68);
00323     DEBUG_PRINTLN(F("Resetting I2C Master control..."));
00324     resetI2CMaster();
00325     delay(20);
00326
00327     // load DMP code into memory banks
00328     DEBUG_PRINT(F("Writing DMP code to MPU memory banks ("));
00329     DEBUG_PRINT(MPU6050_DMP_CODE_SIZE);
00330     DEBUG_PRINTLN(F(" bytes)"));
00331     if (writeProgMemoryBlock(dmpMemory,
00332         MPU6050_DMP_CODE_SIZE)) {
00332         DEBUG_PRINTLN(F("Success! DMP code written and verified.));
00333
00334         // write DMP configuration
00335         DEBUG_PRINT(F("Writing DMP configuration to MPU memory banks ("));
00336         DEBUG_PRINT(MPU6050_DMP_CONFIG_SIZE);
00337         DEBUG_PRINTLN(F(" bytes in config def)"));
00338         if (writeProgDMPConfigurationSet(dmpConfig,
00339             MPU6050_DMP_CONFIG_SIZE)) {
00339             DEBUG_PRINTLN(F("Success! DMP configuration written and verified.));
00340
00341             DEBUG_PRINTLN(F("Setting clock source to Z Gyro..."));
00342             setClockSource(MPU6050_CLOCK_PLL_ZGYRO);
00343
00344             DEBUG_PRINTLN(F("Setting DMP and FIFO_OFLOW interrupts enabled..."));
00345             setIntEnabled(0x12);
00346
00347             DEBUG_PRINTLN(F("Setting sample rate to 200Hz..."));
00348             setRate(4); // 1khz / (1 + X) = X=4 => 200 Hz
00349
00350             DEBUG_PRINTLN(F("Setting external frame sync to TEMP_OUT_L[0]..."));
00351             setExternalFrameSync(MPU6050_EXT_SYNC_TEMP_OUT_L
00352         );
00353
00354             DEBUG_PRINTLN(F("Setting DLPF bandwidth to 42Hz..."));
00355             setDLPFMode(MPU6050_DLPF_BW_42);
00356
00357             DEBUG_PRINTLN(F("Setting gyro sensitivity to +/- 2000 deg/sec..."));
00358             setFullScaleGyroRange(MPU6050_GYRO_FS_2000);
00359
00360             DEBUG_PRINTLN(F("Setting DMP configuration bytes (function unknown)..."));
00361             setDMPConfig1(0x03);
00362             setDMPConfig2(0x00);
00363
00364             DEBUG_PRINTLN(F("Clearing OTP Bank flag..."));
00365             setOTPBANKValid(false);
00366
00367             DEBUG_PRINTLN(F("Setting X/Y/Z gyro offsets to previous values..."));
00368             setXGyroOffset(xgOffset);
00369             setYGyroOffset(ygOffset);
00370             setZGyroOffset(zgOffset);
00371
00372             DEBUG_PRINTLN(F("Setting X/Y/Z gyro user offsets to zero..."));
00373             setXGyroOffsetUser(0);

```

```

00373         setYGyroOffsetUser(0);
00374         setZGyroOffsetUser(0);
00375
00376         DEBUG_PRINTLN(F("Writing final memory update 1/7 (function unknown)..."));
00377         uint8_t dmpUpdate[16], j;
00378         uint16_t pos = 0;
00379         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00380         writeMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00381
00382         DEBUG_PRINTLN(F("Writing final memory update 2/7 (function unknown)..."));
00383         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00384         writeMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00385
00386         DEBUG_PRINTLN(F("Resetting FIFO..."));
00387         resetFIFO();
00388
00389         DEBUG_PRINTLN(F("Reading FIFO count..."));
00390         uint8_t fifoCount = getFIFOCount();
00391         uint8_t fifoBuffer[128];
00392
00393         DEBUG_PRINT(F("Current FIFO count="));
00394         DEBUG_PRINTLN(fifoCount);
00395         getFIFOBytes(fifoBuffer, fifoCount);
00396
00397         DEBUG_PRINTLN(F("Setting motion detection threshold to 2..."));
00398         setMotionDetectionThreshold(2);
00399
00400         DEBUG_PRINTLN(F("Setting zero-motion detection threshold to 156..."));
00401         setZeroMotionDetectionThreshold(156);
00402
00403         DEBUG_PRINTLN(F("Setting motion detection duration to 80..."));
00404         setMotionDetectionDuration(80);
00405
00406         DEBUG_PRINTLN(F("Setting zero-motion detection duration to 0..."));
00407         setZeroMotionDetectionDuration(0);
00408
00409         DEBUG_PRINTLN(F("Resetting FIFO..."));
00410         resetFIFO();
00411
00412         DEBUG_PRINTLN(F("Enabling FIFO..."));
00413         setFIFOEnabled(true);
00414
00415         DEBUG_PRINTLN(F("Enabling DMP..."));
00416         setDMPEnabled(true);
00417
00418         DEBUG_PRINTLN(F("Resetting DMP..."));
00419         resetDMP();
00420
00421         DEBUG_PRINTLN(F("Writing final memory update 3/7 (function unknown)..."));
00422         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00423         writeMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00424
00425         DEBUG_PRINTLN(F("Writing final memory update 4/7 (function unknown)..."));
00426         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00427         writeMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00428
00429         DEBUG_PRINTLN(F("Writing final memory update 5/7 (function unknown)..."));
00430         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00431         writeMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00432
00433         DEBUG_PRINTLN(F("Waiting for FIFO count > 2..."));
00434         while ((fifoCount = getFIFOCount()) < 3);
00435
00436         DEBUG_PRINT(F("Current FIFO count="));
00437         DEBUG_PRINTLN(fifoCount);
00438         DEBUG_PRINTLN(F("Reading FIFO data..."));
00439         getFIFOBytes(fifoBuffer, fifoCount);
00440
00441         DEBUG_PRINTLN(F("Reading interrupt status..."));
00442         uint8_t mpuIntStatus = getIntStatus();
00443
00444         DEBUG_PRINT(F("Current interrupt status="));
00445         DEBUG_PRINTLN(mpuIntStatus, HEX);
00446
00447         DEBUG_PRINTLN(F("Reading final memory update 6/7 (function unknown)..."));
00448         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00449         readMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00450
00451         resetFIFO();
00452
00453         DEBUG_PRINTLN(F("Waiting for FIFO count > 2..."));

```

```

00454         while ((fifoCount = getFIFOCount()) < 3);
00455
00456         DEBUG_PRINT(F("Current FIFO count="));
00457         DEBUG_PRINTLN(fifoCount);
00458
00459         if(fifoCount>128) resetFIFO();
00460
00461         DEBUG_PRINTLN(F("Reading FIFO data..."));
00462         getFIFOBytes(fifoBuffer, fifoCount);
00463
00464         DEBUG_PRINTLN(F("Reading interrupt status..."));
00465         mpuIntStatus = getIntStatus();
00466
00467         DEBUG_PRINT(F("Current interrupt status="));
00468         DEBUG_PRINTLN(mpuIntStatus, HEX);
00469
00470         DEBUG_PRINTLN(F("Writing final memory update 7/7 (function unknown)..."));
00471         for (j = 0; j < 4 || j < dmpUpdate[2] + 3; j++, pos++) dmpUpdate[j] = pgm_read_byte(&dmpUpdates
[pos]);
00472         writeMemoryBlock(dmpUpdate + 3, dmpUpdate[2], dmpUpdate[0], dmpUpdate[1]);
00473
00474         DEBUG_PRINTLN(F("DMP is good to go! Finally.));
00475
00476         DEBUG_PRINTLN(F("Disabling DMP (you turn it on later)..."));
00477         setDMPEnabled(false);
00478
00479         DEBUG_PRINTLN(F("Setting up internal 42-byte (default) DMP packet buffer..."));
00480         dmpPacketSize = 42;
00481         /*if ((dmpPacketBuffer = (uint8_t *)malloc(42)) == 0) {
00482             return 3; // TODO: proper error code for no memory
00483         }*/
00484
00485         DEBUG_PRINTLN(F("Resetting FIFO and clearing INT status one last time..."));
00486         resetFIFO();
00487         getIntStatus();
00488     } else {
00489         DEBUG_PRINTLN(F("ERROR! DMP configuration verification failed.));
00490         return 2; // configuration block loading failed
00491     }
00492 } else {
00493     DEBUG_PRINTLN(F("ERROR! DMP code verification failed.));
00494     return 1; // main binary block loading failed
00495 }
00496 return 0; // success
00497 }
00498
00499 bool MPU6050::dmpPacketAvailable() {
00500     return getFIFOCount() >= dmpGetFIFOPacketSize();
00501 }
00502
00503 // uint8_t MPU6050::dmpSetFIFORate(uint8_t fifoRate);
00504 // uint8_t MPU6050::dmpGetFIFORate();
00505 // uint8_t MPU6050::dmpGetSampleStepSizeMS();
00506 // uint8_t MPU6050::dmpGetSampleFrequency();
00507 // int32_t MPU6050::dmpDecodeTemperature(int8_t tempReg);
00508
00509 //uint8_t MPU6050::dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
00510 //uint8_t MPU6050::dmpUnregisterFIFORateProcess(inv_obj_func func);
00511 //uint8_t MPU6050::dmpRunFIFORateProcesses();
00512
00513 // uint8_t MPU6050::dmpSendQuaternion(uint_fast16_t accuracy);
00514 // uint8_t MPU6050::dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
00515 // uint8_t MPU6050::dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00516 // uint8_t MPU6050::dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00517 // uint8_t MPU6050::dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t accuracy);
00518 // uint8_t MPU6050::dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
00519 // uint8_t MPU6050::dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
00520 // uint8_t MPU6050::dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
00521 // uint8_t MPU6050::dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
00522 // uint8_t MPU6050::dmpSendPacketNumber(uint_fast16_t accuracy);
00523 // uint8_t MPU6050::dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t accuracy);
00524 // uint8_t MPU6050::dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);
00525
00526 uint8_t MPU6050::dmpGetAccel(int32_t *data, const uint8_t* packet) {
00527     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00528     if (packet == 0) packet = dmpPacketBuffer;
00529     data[0] = ((packet[28] << 24) + (packet[29] << 16) + (packet[30] << 8) + packet[31]);
00530     data[1] = ((packet[32] << 24) + (packet[33] << 16) + (packet[34] << 8) + packet[35]);
00531     data[2] = ((packet[36] << 24) + (packet[37] << 16) + (packet[38] << 8) + packet[39]);
00532     return 0;
00533 }
00534 uint8_t MPU6050::dmpGetAccel(int16_t *data, const uint8_t* packet) {
00535     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00536     if (packet == 0) packet = dmpPacketBuffer;
00537     data[0] = (packet[28] << 8) + packet[29];
00538     data[1] = (packet[32] << 8) + packet[33];
00539     data[2] = (packet[36] << 8) + packet[37];

```

```

00540     return 0;
00541 }
00542 uint8_t MPU6050::dmpGetAccel(VectorInt16 *v, const uint8_t* packet) {
00543     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00544     if (packet == 0) packet = dmpPacketBuffer;
00545     v->x = (packet[28] << 8) + packet[29];
00546     v->y = (packet[32] << 8) + packet[33];
00547     v->z = (packet[36] << 8) + packet[37];
00548     return 0;
00549 }
00550 uint8_t MPU6050::dmpGetQuaternion(int32_t *data, const uint8_t* packet) {
00551     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00552     if (packet == 0) packet = dmpPacketBuffer;
00553     data[0] = ((packet[0] << 24) + (packet[1] << 16) + (packet[2] << 8) + packet[3]);
00554     data[1] = ((packet[4] << 24) + (packet[5] << 16) + (packet[6] << 8) + packet[7]);
00555     data[2] = ((packet[8] << 24) + (packet[9] << 16) + (packet[10] << 8) + packet[11]);
00556     data[3] = ((packet[12] << 24) + (packet[13] << 16) + (packet[14] << 8) + packet[15]);
00557     return 0;
00558 }
00559 uint8_t MPU6050::dmpGetQuaternion(int16_t *data, const uint8_t* packet) {
00560     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00561     if (packet == 0) packet = dmpPacketBuffer;
00562     data[0] = ((packet[0] << 8) + packet[1]);
00563     data[1] = ((packet[4] << 8) + packet[5]);
00564     data[2] = ((packet[8] << 8) + packet[9]);
00565     data[3] = ((packet[12] << 8) + packet[13]);
00566     return 0;
00567 }
00568 uint8_t MPU6050::dmpGetQuaternion(Quaternion *q, const uint8_t* packet) {
00569     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00570     int16_t qI[4];
00571     uint8_t status = dmpGetQuaternion(qI, packet);
00572     if (status == 0) {
00573         q->w = (float)qI[0] / 16384.0f;
00574         q->x = (float)qI[1] / 16384.0f;
00575         q->y = (float)qI[2] / 16384.0f;
00576         q->z = (float)qI[3] / 16384.0f;
00577         return 0;
00578     }
00579     return status; // int16 return value, indicates error if this line is reached
00580 }
00581 // uint8_t MPU6050::dmpGet6AxisQuaternion(long *data, const uint8_t* packet);
00582 // uint8_t MPU6050::dmpGetRelativeQuaternion(long *data, const uint8_t* packet);
00583 uint8_t MPU6050::dmpGetGyro(int32_t *data, const uint8_t* packet) {
00584     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00585     if (packet == 0) packet = dmpPacketBuffer;
00586     data[0] = ((packet[16] << 24) + (packet[17] << 16) + (packet[18] << 8) + packet[19]);
00587     data[1] = ((packet[20] << 24) + (packet[21] << 16) + (packet[22] << 8) + packet[23]);
00588     data[2] = ((packet[24] << 24) + (packet[25] << 16) + (packet[26] << 8) + packet[27]);
00589     return 0;
00590 }
00591 uint8_t MPU6050::dmpGetGyro(int16_t *data, const uint8_t* packet) {
00592     // TODO: accommodate different arrangements of sent data (ONLY default supported now)
00593     if (packet == 0) packet = dmpPacketBuffer;
00594     data[0] = (packet[16] << 8) + packet[17];
00595     data[1] = (packet[20] << 8) + packet[21];
00596     data[2] = (packet[24] << 8) + packet[25];
00597     return 0;
00598 }
00599 // uint8_t MPU6050::dmpSetLinearAccelFilterCoefficient(float coef);
00600 // uint8_t MPU6050::dmpGetLinearAccel(long *data, const uint8_t* packet);
00601 uint8_t MPU6050::dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw,
    VectorFloat *gravity) {
00602     // get rid of the gravity component (+lg = +4096 in standard DMP FIFO packet)
00603     v->x = vRaw->x - gravity->x*4096;
00604     v->y = vRaw->y - gravity->y*4096;
00605     v->z = vRaw->z - gravity->z*4096;
00606     return 0;
00607 }
00608 // uint8_t MPU6050::dmpGetLinearAccelInWorld(long *data, const uint8_t* packet);
00609 uint8_t MPU6050::dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal,
    Quaternion *q) {
00610     // rotate measured 3D acceleration vector into original state
00611     // frame of reference based on orientation quaternion
00612     memcpy(v, vReal, sizeof(VectorInt16));
00613     v->rotate(q);
00614     return 0;
00615 }
00616 // uint8_t MPU6050::dmpGetGyroAndAccelSensor(long *data, const uint8_t* packet);
00617 // uint8_t MPU6050::dmpGetGyroSensor(long *data, const uint8_t* packet);
00618 // uint8_t MPU6050::dmpGetControlData(long *data, const uint8_t* packet);
00619 // uint8_t MPU6050::dmpGetTemperature(long *data, const uint8_t* packet);
00620 // uint8_t MPU6050::dmpGetGravity(long *data, const uint8_t* packet);
00621 uint8_t MPU6050::dmpGetGravity(VectorFloat *v, Quaternion *q) {
00622     v->x = 2 * (q->x*q->z - q->y*q->y);
00623     v->y = 2 * (q->x*q->x + q->y*q->z);
00624     v->z = q->x*q->x - q->y*q->y - q->z*q->z;

```

```

00625     return 0;
00626 }
00627 // uint8_t MPU6050::dmpGetUnquantizedAccel(long *data, const uint8_t* packet);
00628 // uint8_t MPU6050::dmpGetQuantizedAccel(long *data, const uint8_t* packet);
00629 // uint8_t MPU6050::dmpGetExternalSensorData(long *data, int size, const uint8_t* packet);
00630 // uint8_t MPU6050::dmpGetEIS(long *data, const uint8_t* packet);
00631
00632 uint8_t MPU6050::dmpGetEuler(float *data, Quaternion *q) {
00633     data[0] = atan2(2*q -> x*q -> y - 2*q -> w*q -> z, 2*q -> w*q -> w + 2*q -> x*q -> x - 1); // psi
00634     data[1] = -asin(2*q -> x*q -> z + 2*q -> w*q -> y); // theta
00635     data[2] = atan2(2*q -> y*q -> z - 2*q -> w*q -> x, 2*q -> w*q -> w + 2*q -> z*q -> z - 1); // phi
00636     return 0;
00637 }
00638 uint8_t MPU6050::dmpGetYawPitchRoll(float *data, Quaternion *q,
    VectorFloat *gravity) {
00639     // yaw: (about Z axis)
00640     data[0] = atan2(2*q -> x*q -> y - 2*q -> w*q -> z, 2*q -> w*q -> w + 2*q -> x*q -> x - 1);
00641     // pitch: (nose up/down, about Y axis)
00642     data[1] = atan(gravity -> x / sqrt(gravity -> y*gravity -> y + gravity -> z*gravity -> z));
00643     // roll: (tilt left/right, about X axis)
00644     data[2] = atan(gravity -> y / sqrt(gravity -> x*gravity -> x + gravity -> z*gravity -> z));
00645     return 0;
00646 }
00647
00648 // uint8_t MPU6050::dmpGetAccelFloat(float *data, const uint8_t* packet);
00649 // uint8_t MPU6050::dmpGetQuaternionFloat(float *data, const uint8_t* packet);
00650
00651 uint8_t MPU6050::dmpProcessFIFOPacket(const unsigned char *dmpData) {
00652     /*for (uint8_t k = 0; k < dmpPacketSize; k++) {
00653         if (dmpData[k] < 0x10) Serial.print("0");
00654         Serial.print(dmpData[k], HEX);
00655         Serial.print(" ");
00656     }
00657     Serial.print("\n");*/
00658     //Serial.println((uint16_t)dmpPacketBuffer);
00659     return 0;
00660 }
00661 uint8_t MPU6050::dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t *processed) {
00662     uint8_t status;
00663     uint8_t buf[dmpPacketSize];
00664     for (uint8_t i = 0; i < numPackets; i++) {
00665         // read packet from FIFO
00666         getFIFOBytes(buf, dmpPacketSize);
00667
00668         // process packet
00669         if ((status = dmpProcessFIFOPacket(buf)) > 0) return status;
00670
00671         // increment external process count variable, if supplied
00672         if (processed != 0) *processed++;
00673     }
00674     return 0;
00675 }
00676
00677 // uint8_t MPU6050::dmpSetFIFOProcessedCallback(void (*func) (void));
00678
00679 // uint8_t MPU6050::dmpInitFIFOParam();
00680 // uint8_t MPU6050::dmpCloseFIFO();
00681 // uint8_t MPU6050::dmpSetGyroDataSource(uint_fast8_t source);
00682 // uint8_t MPU6050::dmpDecodeQuantizedAccel();
00683 // uint32_t MPU6050::dmpGetGyroSumOfSquare();
00684 // uint32_t MPU6050::dmpGetAccelSumOfSquare();
00685 // void MPU6050::dmpOverrideQuaternion(long *q);
00686 uint16_t MPU6050::dmpGetFIFOPacketSize() {
00687     return dmpPacketSize;
00688 }
00689
00690 #endif /* _MPU6050_6AXIS_MOTIONAPPS20_H_ */

```

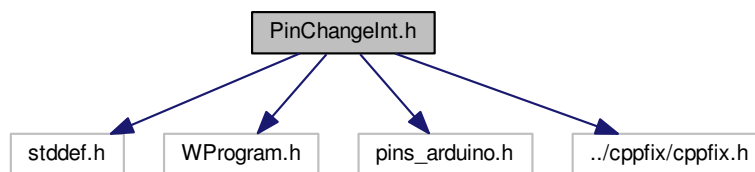
## 4.17 PinChangeInt.h File Reference

```

#include "stdint.h"
#include <WProgram.h>
#include <pins_arduino.h>
#include "../cppfix/cppfix.h"

```

Include dependency graph for PinChangeInt.h:



## Classes

- class [PCintPort](#)
- class [PCintPort::PCintPin](#)

## Macros

- `#define PCINT_VERSION 2190`
- `#define INLINE_PCINT`
- `#define NO_PORTA_PINCHANGES`
- `#define PCdetachInterrupt(pin) PCintPort::detachInterrupt(pin)`
- `#define PCattachInterrupt(pin, userFunc, mode) PCintPort::attachInterrupt(pin, userFunc,mode)`
- `#define PCgetArduinoPin() PCintPort::getArduinoPin()`
- `#define PORTBVECT PCINT0_vect`
- `#define PORTCVECT PCINT1_vect`
- `#define PORTDVECT PCINT2_vect`

## Typedefs

- `typedef void(* PCIntvoidFuncPtr) (void)`

## Functions

- static [PCintPort](#) \* [lookupPortNumToPort](#) (int portNum)
- [ISR](#) ([PORTBVECT](#))
- [ISR](#) ([PORTCVECT](#))
- [ISR](#) ([PORTDVECT](#))

## Variables

- [PCintPort](#) [portB](#) = [PCintPort](#)(2, 0, PCMSK0)
- [PCintPort](#) [portC](#) = [PCintPort](#)(3, 1, PCMSK1)
- [PCintPort](#) [portD](#) = [PCintPort](#)(4, 2, PCMSK2)

### 4.17.1 Macro Definition Documentation

#### 4.17.1.1 `#define INLINE_PCINT`

Definition at line 130 of file [PinChangeInt.h](#).

#### 4.17.1.2 `#define NO_PORTA_PINCHANGES`

Definition at line 149 of file [PinChangeInt.h](#).

#### 4.17.1.3 `#define PCattachInterrupt( pin, userFunc, mode ) PCintPort::attachInterrupt(pin, userFunc, mode)`

Definition at line 163 of file [PinChangeInt.h](#).

#### 4.17.1.4 `#define PCdetachInterrupt( pin ) PCintPort::detachInterrupt(pin)`

Definition at line 162 of file [PinChangeInt.h](#).

#### 4.17.1.5 `#define PCgetArduinoPin( ) PCintPort::getArduinoPin()`

Definition at line 164 of file [PinChangeInt.h](#).

#### 4.17.1.6 `#define PCINT_VERSION 2190`

Definition at line 96 of file [PinChangeInt.h](#).

#### 4.17.1.7 `#define PORTBVECT PCINT0_vect`

Definition at line 557 of file [PinChangeInt.h](#).

#### 4.17.1.8 `#define PORTCVECT PCINT1_vect`

Definition at line 558 of file [PinChangeInt.h](#).

#### 4.17.1.9 `#define PORTDVECT PCINT2_vect`

Definition at line 559 of file [PinChangeInt.h](#).

### 4.17.2 Typedef Documentation

#### 4.17.2.1 `typedef void(* PCIntvoidFuncPtr) (void)`

Definition at line 167 of file [PinChangeInt.h](#).

### 4.17.3 Function Documentation

#### 4.17.3.1 `ISR ( PORTBVECT )`

Definition at line 563 of file [PinChangeInt.h](#).

#### 4.17.3.2 `ISR ( PORTCVECT )`

Definition at line 573 of file [PinChangeInt.h](#).

#### 4.17.3.3 `ISR ( PORTDVECT )`

Definition at line 583 of file [PinChangeInt.h](#).

#### 4.17.3.4 `static PCintPort* lookupPortNumToPort ( int portNum ) [static]`

Definition at line 321 of file [PinChangeInt.h](#).

### 4.17.4 Variable Documentation

## 4.17.4.1 PCIntPort portB = PCIntPort(2, 0, PCMSK0)

Definition at line 301 of file [PinChangeInt.h](#).

## 4.17.4.2 PCIntPort portC = PCIntPort(3, 1, PCMSK1)

Definition at line 304 of file [PinChangeInt.h](#).

## 4.17.4.3 PCIntPort portD = PCIntPort(4, 2, PCMSK2)

Definition at line 307 of file [PinChangeInt.h](#).

## 4.18 PinChangeInt.h

```

00001 // We use 4-character tabstops, so IN VIM: <esc>:set ts=4 and <esc>:set sw=4
00002 // ...that's: ESCAPE key, colon key, then "s-e-t SPACE key t-s=-4"
00003 //
00004 /*
00005  * This is the PinChangeInt library for the Arduino.
00006
00007     See google code project for latest, bugs and info http://code.google.com/p/arduino-pinchangeint/
00008     For more information Refer to avr-gcc header files, arduino source and atmega datasheet.
00009
00010     This library was inspired by and derived from "johnboiles" (it seems)
00011     PCInt Arduino Playground example here: http://www.arduino.cc/playground/Main/PcInt
00012     If you are the original author, please let us know at the google code page
00013
00014     It provides an extension to the interrupt support for arduino by
00015     adding pin change interrupts, giving a way for users to have
00016     interrupts drive off of any pin.
00017
00018     This program is free software: you can redistribute it and/or modify
00019     it under the terms of the GNU General Public License as published by
00020     the Free Software Foundation, either version 3 of the License, or
00021     (at your option) any later version.
00022
00023     This program is distributed in the hope that it will be useful,
00024     but WITHOUT ANY WARRANTY; without even the implied warranty of
00025     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00026     GNU General Public License for more details.
00027
00028     You should have received a copy of the GNU General Public License
00029     along with this program. If not, see <http://www.gnu.org/licenses/>.
00030     (the file gpl.txt is included with the library's zip package)
00031 */
00032 //----- define these in your sketch, if applicable
00033 //----- These must go in your sketch ahead of the #include <PinChangeInt.h> statement
00034 //-----
00035 // You can reduce the memory footprint of this handler by declaring that there will be no pin change
00036 // interrupts
00037 // on any one or two of the three ports. If only a single port remains, the handler will be declared
00038 // inline
00039 // reducing the size and latency of the handler.
00040 // #define NO_PORTB_PINCHANGES // to indicate that port b will not be used for pin change interrupts
00041 // #define NO_PORTC_PINCHANGES // to indicate that port c will not be used for pin change interrupts
00042 // #define NO_PORTD_PINCHANGES // to indicate that port d will not be used for pin change interrupts
00043 // --- Mega support ---
00044 // #define NO_PORTB_PINCHANGES // to indicate that port b will not be used for pin change interrupts
00045 // #define NO_PORTJ_PINCHANGES // to indicate that port c will not be used for pin change interrupts
00046 // #define NO_PORTK_PINCHANGES // to indicate that port d will not be used for pin change interrupts
00047 // In the Mega, there is no Port C, no Port D. Instead, you get Port J and Port K. Port B remains.
00048 // Port J, however, is practically useless because there is only 1 pin available for interrupts. Most
00049 // of the Port J pins are not even connected to a header connection. // </end> "Mega Support" notes
00050 // --- Sanguino, Mioduino support ---
00051 // #define NO_PORTA_PINCHANGES // to indicate that port a will not be used for pin change interrupts
00052 // -----
00053 // Other preprocessor directives...
00054 // You can reduce the code size by 20-50 bytes, and you can speed up the interrupt routine
00055 // slightly by declaring that you don't care if the static variables PCIntPort::pinState and/or
00056 // PCIntPort::arduinoPin are set and made available to your interrupt routine.
00057 // #define NO_PIN_STATE // to indicate that you don't need the pinState
00058 // #define NO_PIN_NUMBER // to indicate that you don't need the arduinoPin
00059 // #define DISABLE_PCINT_MULTI_SERVICE // to limit the handler to servicing a single interrupt per
00060 // invocation.
00061 // #define GET_PCINT_VERSION // to enable the uint16_t getPCIntVersion () function.
00062 // The following is intended for testing purposes. If defined, then a whole host of static variables can
00063 // be read
00064 // in your interrupt subroutine. It is not defined by default, and you DO NOT want to define this in

```



```

00061 // Production code!:
00062 // #define PINMODE
00063 //----- define the above in your sketch, if applicable
00064 -----
00064
00065 /*
00066     PinChangeInt.h
00067     ---- VERSIONS ---- (NOTE TO SELF: Update the PCINT_VERSION define, below) -----
00068     Version 2.19 (beta) Tue Nov 20 07:33:37 CST 2012
00069     Version 2.17 (beta) Sat Nov 17 09:46:50 CST 2012
00070     Version 2.11 (beta) Mon Nov 12 09:33:06 CST 2012
00071
00072     Version 2.01 (beta) Thu Jun 28 12:35:48 CDT 2012
00073
00074     Version 1.72 Wed Mar 14 18:57:55 CDT 2012
00075
00076     Version 1.71beta Sat Mar 10 12:57:05 CST 2012
00077
00078     Version 1.6beta Fri Feb 10 08:48:35 CST 2012
00079
00080     Version 1.51 Sun Feb  5 23:28:02 CST 2012
00081
00082     Version 1.5 Thu Feb  2 18:09:49 CST 2012
00083
00084     Version 1.4 Tue Jan 10 09:41:14 CST 2012
00085
00086     Version 1.3 Sat Dec  3 22:56:20 CST 2011
00087
00088     Version 1.2 Sat Dec  3 Sat Dec  3 09:15:52 CST 2011
00089
00090     Version 1.1 Sat Dec  3 00:06:03 CST 2011
00091 */
00092
00093 #ifndef PinChangeInt_h
00094 #define PinChangeInt_h
00095
00096 #define PCINT_VERSION 2190 // This number MUST agree with the version number, above.
00097
00098 #include "stdint.h"
00099
00100 // Thanks to Maurice Beelen, nms277, Akesson Karlpetter, and Orly Andico for these fixes.
00101 #if defined(ARDUINO) && ARDUINO >= 100
00102     #include <Arduino.h>
00103     #include <new.h>
00104     #include <wiring_private.h> // cby and sbi defined here
00105 #else
00106     #include <WProgram.h>
00107     #include <pins_arduino.h>
00108     #ifndef LIBCALL_PINCHANGEINT
00109         #include "../cppfix/cppfix.h"
00110     #endif
00111 #endif
00112
00113 #undef DEBUG
00114
00115
00116 /*
00117 * Theory: all IO pins on Atmega168 are covered by Pin Change Interrupts.
00118 * The PCINT corresponding to the pin must be enabled and masked, and
00119 * an ISR routine provided. Since PCINTs are per port, not per pin, the ISR
00120 * must use some logic to actually implement a per-pin interrupt service.
00121 */
00122
00123 /* Pin to interrupt map:
00124 * D0-D7 = PCINT 16-23 = PCIR2 = PD = PCIE2 = pcmsk2
00125 * D8-D13 = PCINT 0-5 = PCIR0 = PB = PCIE0 = pcmsk0
00126 * A0-A5 (D14-D19) = PCINT 8-13 = PCIR1 = PC = PCIE1 = pcmsk1
00127 */
00128
00129 #undef INLINE_PCINT
00130 #define INLINE_PCINT
00131 // Thanks to cserveny...@gmail.com for MEGA support!
00132 #if defined __AVR_ATmega2560__ || defined __AVR_ATmega1280__ || defined __AVR_ATmega1281__ || defined
    __AVR_ATmega2561__ || defined __AVR_ATmega640__
00133     #define __USE_PORT_JK
00134     // Mega does not have PORTA, C or D
00135     #define NO_PORTA_PINCHANGES
00136     #define NO_PORTC_PINCHANGES
00137     #define NO_PORTD_PINCHANGES
00138     #if ((defined(NO_PORTB_PINCHANGES) && defined(NO_PORTJ_PINCHANGES)) || \
00139         (defined(NO_PORTJ_PINCHANGES) && defined(NO_PORTK_PINCHANGES)) || \
00140         (defined(NO_PORTK_PINCHANGES) && defined(NO_PORTB_PINCHANGES)))
00141         #define INLINE_PCINT inline
00142     #endif
00143 #else
00144     #if defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644__)
00145         #ifndef NO_PORTA_PINCHANGES

```

```

00146         #define __USE_PORT_A
00147     #endif
00148     #else
00149         #define NO_PORTA_PINCHANGES
00150     #endif
00151     // if defined only D .OR. only C .OR. only B .OR. only A, then inline it
00152     #if ( (defined(NO_PORTA_PINCHANGES) && defined(NO_PORTB_PINCHANGES) && defined(NO_PORTC_PINCHANGES))
00153     || \
00154         (defined(NO_PORTA_PINCHANGES) && defined(NO_PORTB_PINCHANGES) && defined(NO_PORTD_PINCHANGES))
00155     || \
00156         (defined(NO_PORTA_PINCHANGES) && defined(NO_PORTC_PINCHANGES) && defined(NO_PORTD_PINCHANGES))
00157     || \
00158         (defined(NO_PORTB_PINCHANGES) && defined(NO_PORTC_PINCHANGES) && defined(NO_PORTD_PINCHANGES))
00159     )
00160     #define INLINE_PCINT inline
00161     #endif
00162 #endif
00163 // Provide drop in compatibility with johnboiles PCInt project at
00164 // http://www.arduino.cc/playground/Main/PcInt
00165 #define PCdetachInterrupt(pin) PCIntPort::detachInterrupt(pin)
00166 #define PCattachInterrupt(pin,userFunc,mode) PCIntPort::attachInterrupt(pin, userFunc,mode)
00167 #define PCgetArduinoPin() PCIntPort::getArduinoPin()
00168
00169 typedef void (*PCIntvoidFuncPtr)(void);
00170
00171 class PCIntPort {
00172 public:
00173     PCIntPort(int index,int pcindex, volatile uint8_t& maskReg) :
00174         portInputReg(*portInputRegister(index)),
00175         portPCMask(maskReg),
00176         PCICRbit(1 << pcindex),
00177         portRisingPins(0),
00178         portFallingPins(0),
00179         firstPin(NULL)
00180 #ifdef PINMODE
00181         ,intrCount(0)
00182 #endif
00183     {
00184         #ifdef FLASH
00185         ledsetup();
00186         #endif
00187     }
00188     volatile uint8_t& portInputReg;
00189     static int8_t attachInterrupt(uint8_t pin,
00190 PCIntvoidFuncPtr userFunc, int mode);
00191     static void detachInterrupt(uint8_t pin);
00192     INLINE_PCINT void PCint();
00193     static volatile uint8_t curr;
00194     #ifndef NO_PIN_NUMBER
00195     static volatile uint8_t arduinoPin;
00196     #endif
00197     #ifndef NO_PIN_STATE
00198     static volatile uint8_t pinState;
00199     #endif
00200     #ifdef PINMODE
00201     static volatile uint8_t pinmode;
00202     static volatile uint8_t s_portRisingPins;
00203     static volatile uint8_t s_portFallingPins;
00204     static volatile uint8_t s_lastPinView;
00205     static volatile uint8_t s_pmask;
00206     static volatile char s_PORT;
00207     static volatile uint8_t s_changedPins;
00208     static volatile uint8_t s_portRisingPins_nCurr;
00209     static volatile uint8_t s_portFallingPins_nNCurr;
00210     static volatile uint8_t s_currXORlastPinView;
00211     volatile uint8_t intrCount;
00212     static volatile uint8_t s_count;
00213     static volatile uint8_t pcint_multi;
00214     static volatile uint8_t PCIFRbug;
00215     #endif
00216     #ifdef FLASH
00217     static void ledsetup(void);
00218     #endif
00219 protected:
00220     class PCIntPin {
00221     public:
00222         PCIntPin() :
00223             PCIntFunc((PCIntvoidFuncPtr) NULL),
00224             mode(0) {}
00225         PCIntvoidFuncPtr PCIntFunc;
00226         uint8_t mode;
00227         uint8_t mask;
00228         uint8_t arduinoPin;
00229         PCIntPin* next;

```

```

00228     };
00229     void          enable(PCintPin* pin, PCIntvoidFuncPtr userFunc, uint8_t mode
);
00230     int8_t        addPin(uint8_t arduinoPin,PCIntvoidFuncPtr userFunc, uint8_t mode);
00231     volatile      uint8_t&      portPCMask;
00232     const          uint8_t      PCICRbit;
00233     volatile      uint8_t      portRisingPins;
00234     volatile      uint8_t      portFallingPins;
00235     volatile      uint8_t      lastPinView;
00236     PCintPin*     firstPin;
00237 };
00238
00239 #ifndef LIBCALL_PINCHANGEINT // LIBCALL_PINCHANGEINT *****
00240 volatile uint8_t PCintPort::curr=0;
00241 #ifndef NO_PIN_NUMBER
00242 volatile uint8_t PCintPort::arduinoPin=0;
00243 #endif
00244 #ifndef NO_PIN_STATE
00245 volatile uint8_t PCintPort::pinState=0;
00246 #endif
00247 #ifdef PINMODE
00248 volatile uint8_t PCintPort::pinmode=0;
00249 volatile uint8_t PCintPort::s_portRisingPins=0;
00250 volatile uint8_t PCintPort::s_portFallingPins=0;
00251 volatile uint8_t PCintPort::s_lastPinView=0;
00252 volatile uint8_t PCintPort::s_pmask=0;
00253 volatile char    PCintPort::s_PORT='x';
00254 volatile uint8_t PCintPort::s_changedPins=0;
00255 volatile uint8_t PCintPort::s_portRisingPins_nCurr=0;
00256 volatile uint8_t PCintPort::s_portFallingPins_nCurr=0;
00257 volatile uint8_t PCintPort::s_currXORlastPinView=0;
00258 volatile uint8_t PCintPort::s_count=0;
00259 volatile uint8_t PCintPort::pcint_multi=0;
00260 volatile uint8_t PCintPort::PCIFRbug=0;
00261 #endif
00262
00263 #ifdef FLASH
00264 #define PINLED 13
00265 volatile uint8_t *led_port;
00266 uint8_t led_mask;
00267 uint8_t not_led_mask;
00268 boolean ledsetup_run=false;
00269 void PCintPort::ledsetup(void) {
00270     if (! ledsetup_run) {
00271         led_port=portOutputRegister(digitalPinToPort(PINLED));
00272         led_mask=digitalPinToBitMask(PINLED);
00273         not_led_mask=led_mask^0xFF;
00274         pinMode(PINLED, OUTPUT); digitalWrite(PINLED, LOW);
00275         ledsetup_run=true;
00276     }
00277 };
00278 #endif
00279
00280
00281 // ATMEGA 644
00282 //
00283 #if defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644__) // Sanguino, Mosquino uino bobino
    bonanafannafofino, me my momino...
00284
00285 #ifndef NO_PORTA_PINCHANGES
00286 PCintPort portA=PCintPort(1, 0,PCMSK0); // port PB==2 (from Arduino.h, Arduino version
    1.0)
00287 #endif
00288 #ifndef NO_PORTB_PINCHANGES
00289 PCintPort portB=PCintPort(2, 1,PCMSK1); // port PB==2 (from Arduino.h, Arduino
    version 1.0)
00290 #endif
00291 #ifndef NO_PORTC_PINCHANGES
00292 PCintPort portC=PCintPort(3, 2,PCMSK2); // port PC==3 (also in pins_arduino.c,
    Arduino version 022)
00293 #endif
00294 #ifndef NO_PORTD_PINCHANGES
00295 PCintPort portD=PCintPort(4, 3,PCMSK3); // port PD==4
00296 #endif
00297
00298 #else // others
00299
00300 #ifndef NO_PORTB_PINCHANGES
00301 PCintPort portB=PCintPort(2, 0,PCMSK0); // port PB==2 (from Arduino.h, Arduino version
    1.0)
00302 #endif
00303 #ifndef NO_PORTC_PINCHANGES // note: no PORTC on MEGA
00304 PCintPort portC=PCintPort(3, 1,PCMSK1); // port PC==3 (also in pins_arduino.c, Arduino
    version 022)
00305 #endif
00306 #ifndef NO_PORTD_PINCHANGES // note: no PORTD on MEGA
00307 PCintPort portD=PCintPort(4, 2,PCMSK2); // port PD==4

```

```

00308 #endif
00309
00310 #endif // defined __AVR_ATmega644__
00311
00312 #ifdef __USE_PORT_JK
00313 #ifndef NO_PORTJ_PINCHANGES
00314 PCintPort portJ=PCintPort(10,1,PCMSK1); // port PJ==10
00315 #endif
00316 #ifndef NO_PORTK_PINCHANGES
00317 PCintPort portK=PCintPort(11,2,PCMSK2); // port PK==11
00318 #endif
00319 #endif // USE_PORT_JK
00320
00321 static PCintPort *lookupPortNumToPort( int portNum ) {
00322     PCintPort *port = NULL;
00323
00324     switch (portNum) {
00325 #ifndef NO_PORTA_PINCHANGES
00326     case 1:
00327         port=&portA;
00328         break;
00329 #endif
00330 #ifndef NO_PORTB_PINCHANGES
00331     case 2:
00332         port=&portB;
00333         break;
00334 #endif
00335 #ifndef NO_PORTC_PINCHANGES
00336     case 3:
00337         port=&portC;
00338         break;
00339 #endif
00340 #ifndef NO_PORTD_PINCHANGES
00341     case 4:
00342         port=&portD;
00343         break;
00344 #endif
00345 #ifdef __USE_PORT_JK
00346 #ifndef NO_PORTJ_PINCHANGES
00347     case 10:
00348         port=&portJ;
00349         break;
00350 #endif
00351 #endif
00352 #ifndef NO_PORTK_PINCHANGES
00353     case 11:
00354         port=&portK;
00355         break;
00356 #endif
00357 #endif
00358     }
00359 #endif
00360     }
00361
00362     return port;
00363 }
00364
00365
00366 void PCintPort::enable(PCintPin* p, PCIntvoidFuncPtr userFunc,
    uint8_t mode) {
00367     // Enable the pin for interrupts by adding to the PCMSKx register.
00368     // ...The final steps; at this point the interrupt is enabled on this pin.
00369     p->mode=mode;
00370     p->PCintFunc=userFunc;
00371     portPCMask |= p->mask;
00372     if ((p->mode == RISING) || (p->mode == CHANGE)) portRisingPins |= p->
    mask;
00373     if ((p->mode == FALLING) || (p->mode == CHANGE)) portFallingPins |= p->
    mask;
00374     PCICR |= PCICRbit;
00375 }
00376
00377 int8_t PCintPort::addPin(uint8_t arduinoPin, PCIntvoidFuncPtr userFunc,
    uint8_t mode)
00378 {
00379     PCintPin* tmp;
00380
00381     // Add to linked list, starting with firstPin. If pin already exists, just enable.
00382     if (firstPin != NULL) {
00383         tmp=firstPin;
00384         do {
00385             if (tmp->arduinoPin == arduinoPin) { enable(tmp, userFunc, mode); return(0); }
00386             if (tmp->next == NULL) break;
00387             tmp=tmp->next;
00388         } while (true);
00389     }
00390 }

```

```

00391 // Create pin p: fill in the data.
00392 PCintPin* p=new PCintPin;
00393 if (p == NULL) return(-1);
00394 p->arduinoPin=arduinoPin;
00395 p->mode = mode;
00396 p->next=NULL;
00397 p->mask = digitalPinToBitMask(arduinoPin); // the mask
00398
00399 if (firstPin == NULL) firstPin=p;
00400 else tmp->next=p;
00401
00402 #ifdef DEBUG
00403 Serial.print("addPin. pin given: "); Serial.print(arduinoPin, DEC);
00404 int addr = (int) p;
00405 Serial.print(" instance addr: "); Serial.println(addr, HEX);
00406 Serial.print("userFunc addr: "); Serial.println((int)p->PCintFunc, HEX);
00407 #endif
00408
00409 enable(p, userFunc, mode);
00410 #ifdef DEBUG
00411 Serial.print("addPin. pin given: "); Serial.print(arduinoPin, DEC), Serial.print (" pin stored: ");
00412 int addr = (int) p;
00413 Serial.print(" instance addr: "); Serial.println(addr, HEX);
00414 #endif
00415 return(1);
00416 }
00417
00418 /*
00419 * attach an interrupt to a specific pin using pin change interrupts.
00420 */
00421 int8_t PCintPort::attachInterrupt(uint8_t arduinoPin,
PCintvoidFuncPtr userFunc, int mode)
00422 {
00423     PCintPort *port;
00424     uint8_t portNum = digitalPinToPort(arduinoPin);
00425     if ((portNum == NOT_A_PORT) || (userFunc == NULL)) return(-1);
00426
00427     port=lookupPortNumToPort(portNum);
00428     // Added by GreyGnome... must set the initial value of lastPinView for it to be correct on the 1st
interrupt.
00429     // ...but even then, how do you define "correct"? Ultimately, the user must specify (not provisioned
for yet).
00430     port->lastPinView=port->portInputReg;
00431
00432 #ifdef DEBUG
00433 Serial.print("attachInterrupt FUNC: "); Serial.println(arduinoPin, DEC);
00434 #endif
00435 // map pin to PCIR register
00436 return(port->addPin(arduinoPin,userFunc,mode));
00437 }
00438
00439 void PCintPort::detachInterrupt(uint8_t arduinoPin)
00440 {
00441     PCintPort *port;
00442     PCintPin* current;
00443     uint8_t mask;
00444 #ifdef DEBUG
00445 Serial.print("detachInterrupt: "); Serial.println(arduinoPin, DEC);
00446 #endif
00447     uint8_t portNum = digitalPinToPort(arduinoPin);
00448     if (portNum == NOT_A_PORT) return;
00449     port=lookupPortNumToPort(portNum);
00450     mask=digitalPinToBitMask(arduinoPin);
00451     current=port->firstPin;
00452     //PCintPin* prev=NULL;
00453     while (current) {
00454         if (current->mask == mask) { // found the target
00455             uint8_t oldSREG = SREG;
00456             cli(); // disable interrupts
00457             port->portPCMask &= ~mask; // disable the mask entry.
00458             if (port->portPCMask == 0) PCICR &= ~(port->PCICRbit);
00459             port->portRisingPins &= ~current->mask; port->
portFallingPins &= ~current->mask;
00460             // Link the previous' next to the found next. Then remove the found.
00461             //if (prev != NULL) prev->next=current->next; // linked list skips over current.
00462             //else firstPin=current->next; // at the first pin; save the new first pin
00463             SREG = oldSREG; // Restore register; reenables interrupts
00464             return;
00465         }
00466         //prev=current;
00467         current=current->next;
00468     }
00469 }
00470
00471 // common code for isr handler. "port" is the PCINT number.
00472 // there isn't really a good way to back-map ports and masks to pins.
00473 void PCintPort::PCint() {

```

```

00474     uint8_t thisChangedPin; //MIKE
00475
00476     #ifdef FLASH
00477     if (*led_port & led_mask) *led_port&=not_led_mask;
00478     else *led_port|=led_mask;
00479     #endif
00480     #ifndef DISABLE_PCINT_MULTI_SERVICE
00481     uint8_t pcifr;
00482     while (true) {
00483     #endif
00484         // get the pin states for the indicated port.
00485         #ifdef PINMODE
00486         PCintPort::s_lastPinView=lastPinView;
00487         intrCount++;
00488         PCintPort::s_count=intrCount;
00489         #endif
00490         // OLD v. 2.01 technique: Test 1: 3163; Test 7: 3993
00491         // From robtillaart online: ----- (starting v. 2.11beta)
00492         // uint8_t changedPins = PCintPort::curr ^ lastPinView;
00493         // lastPinView = PCintPort::curr;
00494         // uint8_t fastMask = changedPins & ((portRisingPins & PCintPort::curr ) | ( portFallingPins &
~PCintPort::curr ));
00495         // NEW v. 2.11 technique: Test 1: 3270 Test 7: 3987
00496         // -----
00497         // was: uint8_t changedPins = PCintPort::curr ^ lastPinView;
00498         // makes test 6 of the PinChangeIntSpeedTest go from 3867 to 3923. Not good.
00499         uint8_t changedPins = (PCintPort::curr ^ lastPinView) &
00500             ((portRisingPins & PCintPort::curr ) | (
portFallingPins & ~PCintPort::curr ));
00501
00502         #ifdef PINMODE
00503         PCintPort::s_currXORlastPinView=PCintPort::curr ^ lastPinView;
00504         PCintPort::s_portRisingPins_nCurr=portRisingPins &
PCintPort::curr;
00505         PCintPort::s_portFallingPins_nNCurr=portFallingPins & ~
PCintPort::curr;
00506         #endif
00507         lastPinView = PCintPort::curr;
00508
00509         PCintPin* p = firstPin;
00510         while (p) {
00511             // Trigger interrupt if the bit is high and it's set to trigger on mode RISING or CHANGE
00512             // Trigger interrupt if the bit is low and it's set to trigger on mode FALLING or CHANGE
00513             thisChangedPin=p->mask & changedPins; // PinChangeIntSpeedTest makes this 3673... weird.
But GOOD!!!
00514             if (p->mask & changedPins) {
00515                 #ifndef NO_PIN_STATE
00516                 PCintPort::pinState=PCintPort::curr & p->mask ? HIGH : LOW;
00517                 #endif
00518                 #ifndef NO_PIN_NUMBER
00519                 PCintPort::arduinoPin=p->arduinoPin;
00520                 #endif
00521                 #ifdef PINMODE
00522                 PCintPort::pinmode=p->mode;
00523                 PCintPort::s_portRisingPins=portRisingPins;
00524                 PCintPort::s_portFallingPins=portFallingPins;
00525                 PCintPort::s_pmask=p->mask;
00526                 PCintPort::s_changedPins=changedPins;
00527                 #endif
00528                 p->PCintFunc();
00529             }
00530             p=p->next;
00531         }
00532     #ifndef DISABLE_PCINT_MULTI_SERVICE
00533     pcifr = PCIFR & PCICRbit;
00534     if (pcifr == 0) break;
00535     PCIFR |= PCICRbit;
00536     #ifdef PINMODE
00537     PCintPort::pcint_multi++;
00538     if (PCIFR & PCICRbit) PCintPort::PCIFRbug=1; // PCIFR & PCICRbit should ALWAYS be 0 here!
00539     #endif
00540     PCintPort::curr=portInputReg;
00541     }
00542     #endif
00543 }
00544
00545 #ifndef NO_PORTA_PINCHANGES
00546 ISR(PCINT0_vect) {
00547     #ifdef PINMODE
00548     PCintPort::s_PORT='A';
00549     #endif
00550     PCintPort::curr = portA.portInputReg;
00551     portA.PCint();
00552 }
00553 #define PORTBVECT PCINT1_vect
00554 #define PORTCVECT PCINT2_vect
00555 #define PORTDVECT PCINT3_vect

```

```

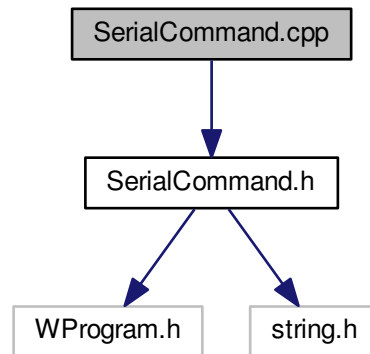
00556 #else
00557 #define PORTBVECT PCINT0_vect
00558 #define PORTCVECT PCINT1_vect
00559 #define PORTDVECT PCINT2_vect
00560 #endif
00561
00562 #ifndef NO_PORTB_PINCHANGES
00563 ISR(PORTBVECT) {
00564     #ifndef PINMODE
00565         PCIntPort::s_PORT='B';
00566     #endif
00567     PCIntPort::curr = portB.portInputReg;
00568     portB.PCint();
00569 }
00570 #endif
00571
00572 #ifndef NO_PORTC_PINCHANGES
00573 ISR(PORTCVECT) {
00574     #ifndef PINMODE
00575         PCIntPort::s_PORT='C';
00576     #endif
00577     PCIntPort::curr = portC.portInputReg;
00578     portC.PCint();
00579 }
00580 #endif
00581
00582 #ifndef NO_PORTD_PINCHANGES
00583 ISR(PORTDVECT){
00584     #ifndef PINMODE
00585         PCIntPort::s_PORT='D';
00586     #endif
00587     PCIntPort::curr = portD.portInputReg;
00588     portD.PCint();
00589 }
00590 #endif
00591
00592 #ifdef __USE_PORT_JK
00593 #ifndef NO_PORTJ_PINCHANGES
00594 ISR(PCINT1_vect) {
00595     #ifndef PINMODE
00596         PCIntPort::s_PORT='J';
00597     #endif
00598     PCIntPort::curr = portJ.portInputReg;
00599     portJ.PCint();
00600 }
00601 #endif
00602
00603 #ifndef NO_PORTK_PINCHANGES
00604 ISR(PCINT2_vect){
00605     #ifndef PINMODE
00606         PCIntPort::s_PORT='K';
00607     #endif
00608     PCIntPort::curr = portK.portInputReg;
00609     portK.PCint();
00610 }
00611 #endif
00612
00613 #endif // __USE_PORT_JK
00614
00615 #ifdef GET_PCINT_VERSION
00616 uint16_t getPCIntVersion () {
00617     return ((uint16_t) PCINT_VERSION);
00618 }
00619 #endif // GET_PCINT_VERSION
00620 #endif // #ifndef LIBCALL_PINCHANGEINT *****
00621 #endif // #ifndef PinChangeInt_h *****

```

## 4.19 SerialCommand.cpp File Reference

```
#include "SerialCommand.h"
```

Include dependency graph for SerialCommand.cpp:



## 4.20 SerialCommand.cpp

```

00001
00024 #include "SerialCommand.h"
00025
00029 SerialCommand::SerialCommand()
00030 : commandList(NULL),
00031   commandCount(0),
00032   defaultHandler(NULL),
00033   term('\n'), // default terminator for commands, newline character
00034   last(NULL)
00035 {
00036   strcpy(delim, " "); // strtok_r needs a null-terminated string
00037   clearBuffer();
00038 }
00039
00045 void SerialCommand::addCommand(const char *command, void (*function)()) {
00046   #ifdef SERIALCOMMAND_DEBUG
00047     Serial.print(F("Adding command "));
00048     Serial.print(commandCount);
00049     Serial.print(F(": "));
00050     Serial.println(command);
00051   #endif
00052
00053   commandList = (SerialCommandCallback *) realloc(
commandList, (commandCount + 1) * sizeof(
SerialCommandCallback));
00054   strncpy(commandList[commandCount].command, command,
SERIALCOMMAND_MAXCOMMANDELENGTH);
00055   commandList[commandCount].function = function;
00056   commandCount++;
00057 }
00058
00063 void SerialCommand::setDefaultHandler(void (*function)(const char *)) {
00064   defaultHandler = function;
00065 }
00066
00067
00073 void SerialCommand::readSerial() {
00074   while (Serial.available() > 0) {
00075     char inChar = Serial.read(); // Read single available character, there may be more waiting
00076     #ifdef SERIALCOMMAND_DEBUG
00077       Serial.print(inChar); // Echo back to serial stream
00078     #endif
00079
00080     if (inChar == term) { // Check for the terminator (default '\r') meaning end of command
00081       #ifdef SERIALCOMMAND_DEBUG
00082         Serial.print(F("Received: "));

```



```

00083     Serial.println(buffer);
00084 #endif
00085
00086     char *command = strtok_r(buffer, delim, &last);    // Search for command at start of
buffer
00087     if (command != NULL) {
00088         boolean matched = false;
00089         for (int i = 0; i < commandCount; i++) {
00090             #ifdef SERIALCOMMAND_DEBUG
00091                 Serial.print(F("Comparing "));
00092                 Serial.print(command);
00093                 Serial.print(F("] to ["));
00094                 Serial.print(commandList[i].command);
00095                 Serial.println(F("]"));
00096             #endif
00097
00098             // Compare the found command against the list of known commands for a match
00099             if (strcmp(command, commandList[i].command,
SERIALCOMMAND_MAXCOMMANDELENGTH) == 0) {
00100                 #ifdef SERIALCOMMAND_DEBUG
00101                     Serial.print(F("Matched Command: "));
00102                     Serial.println(command);
00103                 #endif
00104
00105                 // Execute the stored handler function for the command
00106                 (*commandList[i].function)();
00107                 matched = true;
00108                 break;
00109             }
00110         }
00111         if (!matched && (defaultHandler != NULL)) {
00112             (*defaultHandler)(command);
00113         }
00114     }
00115     clearBuffer();
00116 }
00117 else if (isprint(inChar)) {    // Only printable characters into the buffer
00118     if (bufPos < SERIALCOMMAND_BUFFER) {
00119         buffer[bufPos++] = inChar;    // Put character into buffer
00120         buffer[bufPos] = '\0';    // Null terminate
00121     } else {
00122         #ifdef SERIALCOMMAND_DEBUG
00123             Serial.println(F("Line buffer is full - increase SERIALCOMMAND_BUFFER"));
00124         #endif
00125     }
00126 }
00127 }
00128 }
00129
00130 /*
00131  * Clear the input buffer.
00132  */
00133 void SerialCommand::clearBuffer() {
00134     buffer[0] = '\0';
00135     bufPos = 0;
00136 }
00137
00142 char *SerialCommand::next() {
00143     return strtok_r(NULL, delim, &last);
00144 }

```

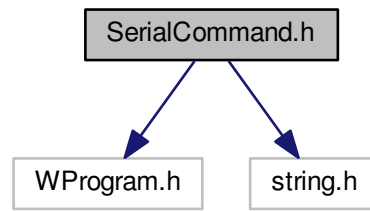
## 4.21 SerialCommand.h File Reference

```

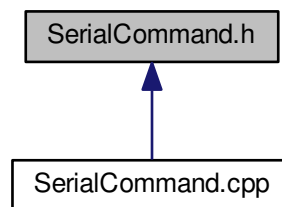
#include <WProgram.h>
#include <string.h>

```

Include dependency graph for SerialCommand.h:



This graph shows which files directly or indirectly include this file:



#### Classes

- class [SerialCommand](#)
- struct [SerialCommand::SerialCommandCallback](#)

#### Macros

- `#define` [SERIALCOMMAND\\_BUFFER](#) 32
- `#define` [SERIALCOMMAND\\_MAXCOMMANLENGTH](#) 3

#### 4.21.1 Macro Definition Documentation

##### 4.21.1.1 `#define` SERIALCOMMAND\_BUFFER 32

[SerialCommand](#) - A Wiring/Arduino library to tokenize and parse commands received over a serial port.

Copyright (C) 2012 Stefan Rado Copyright (C) 2011 Steven Cogswell [steven.cogswell@gmail.com](mailto:steven.cogswell@gmail.com)  
<http://husks.wordpress.com>

Version 20120522

This library is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU General Public License along with this library. If not, see <http://www.gnu.org/licenses/>.

Definition at line 39 of file [SerialCommand.h](#).

#### 4.21.1.2 #define SERIALCOMMAND\_MAXCOMMANDELENGTH 3

Definition at line 41 of file [SerialCommand.h](#).

## 4.22 SerialCommand.h

```

00001
00024 #ifndef SerialCommand_h
00025 #define SerialCommand_h
00026
00027
00028 #if defined(WIRING) && WIRING >= 100
00029     #include <Wiring.h>
00030 #elif defined(ARDUINO) && ARDUINO >= 100
00031     #include <Arduino.h>
00032 #else
00033     #include <WProgram.h>
00034 #endif
00035
00036 #include <string.h>
00037
00038 // Size of the input buffer in bytes (maximum length of one command plus arguments)
00039 #define SERIALCOMMAND_BUFFER 32
00040 // Maximum length of a command excluding the terminating null
00041 #define SERIALCOMMAND_MAXCOMMANDELENGTH 3
00042
00043 // Uncomment the next line to run the library in debug mode (verbose messages)
00044 // #define SERIALCOMMAND_DEBUG
00045
00046
00047 class SerialCommand {
00048     public:
00049         SerialCommand();           // Constructor
00050         void addCommand(const char *command, void(*function)()); // Add a command to the processing
dictionary.
00051         void setDefaultHandler(void (*function)(const char *)); // A handler to call when no
valid command received.
00052
00053         void readSerial();         // Main entry point.
00054         void clearBuffer();        // Clears the input buffer.
00055         char *next();              // Returns pointer to next token found in command buffer (for getting
arguments to commands).
00056
00057     private:
00058         // Command/handler dictionary
00059         struct SerialCommandCallback {
00060             char command[SERIALCOMMAND_MAXCOMMANDELENGTH + 1];
00061             void (*function)();
00062         };
00063         SerialCommandCallback *commandList; // Actual definition for
command/handler array
00064         uint8_t commandCount;
00065
00066         // Pointer to the default handler function
00067         void (*defaultHandler)(const char *);
00068
00069         char delim[2]; // null-terminated list of character to be used as delimiters for tokenizing
(default " ")
00070         char term;      // Character that signals end of command (default '\n')
00071
00072         char buffer[SERIALCOMMAND_BUFFER + 1]; // Buffer of stored characters while
waiting for terminator character
00073         uint8_t bufPos;                          // Current position in the buffer
00074         char *last;                              // State variable used by strtok_r during processing
00075     };
00076
00077 #endif //SerialCommand_h

```

## Index

- addCommand
  - SerialCommand, 78
- addPin
  - PCintPort, 74
- arduinoPin
  - PCintPort, 74
  - PCintPort::PCintPin, 72
- attachInterrupt
  - PCintPort, 74
- bufPos
  - SerialCommand, 79
- buffer
  - MPU6050, 71
  - SerialCommand, 78
- CC\_FACTOR
  - definitions.h, 84
- clearBuffer
  - SerialCommand, 78
- command
  - SerialCommand::SerialCommandCallback, 79
- commandCount
  - SerialCommand, 79
- commandList
  - SerialCommand, 79
- curr
  - PCintPort, 74
- DEBUG\_PRINT
  - definitions.h, 84
- DEBUG\_PRINTF
  - definitions.h, 84
- DEBUG\_PRINTLN
  - definitions.h, 84
- DEBUG\_PRINTLNf
  - definitions.h, 84
- DMP\_50HZ
  - definitions.h, 84
- defaultHandler
  - SerialCommand, 79
- definitions.h, 83
  - CC\_FACTOR, 84
  - DEBUG\_PRINT, 84
  - DEBUG\_PRINTF, 84
  - DEBUG\_PRINTLN, 84
  - DEBUG\_PRINTLNf, 84
  - DMP\_50HZ, 84
  - I2C\_SPEED, 84
  - LEDPIN\_OFF, 84
  - LEDPIN\_ON, 84
  - LEDPIN\_PINMODE, 84
  - LEDPIN\_SWITCH, 84
  - MAX\_RC, 84
  - MID\_RC, 84
  - MIN\_RC, 84
  - MOTORUPDATE\_FREQ, 84
  - MPU6050\_ADDRESS\_AD0\_HIGH, 85
  - MPU6050\_ADDRESS\_AD0\_LOW, 85
  - MPU6050\_DEFAULT\_ADDRESS, 85
  - MPU6050\_DLPF\_BW, 85
  - MPU6050\_GYRO\_FS, 85
  - N\_SIN, 85
  - PWM\_32KHZ\_PHASE, 85
  - PWM\_A\_MOTOR0, 85
  - PWM\_A\_MOTOR1, 85
  - PWM\_B\_MOTOR0, 85
  - PWM\_B\_MOTOR1, 85
  - PWM\_C\_MOTOR0, 85
  - PWM\_C\_MOTOR1, 85
  - RC\_DEADBAND, 85
  - RC\_PIN\_PITCH, 85
  - RC\_PIN\_ROLL, 86
  - SCALE\_ACC, 86
  - SCALE\_PID\_PARAMS, 86
- delim
  - SerialCommand, 79
- detachInterrupt
  - PCintPort, 74
- devAddr
  - MPU6050, 71
- EEPROM\_readAnything
  - EEPROMAnything.h, 87
- EEPROM\_writeAnything
  - EEPROMAnything.h, 87
- EEPROMAnything.h, 87
  - EEPROM\_readAnything, 87
  - EEPROM\_writeAnything, 87
- enable
  - PCintPort, 74
- firstPin
  - PCintPort, 74
- function
  - SerialCommand::SerialCommandCallback, 79
- getAccelFIFOEnabled
  - MPU6050, 15
- getAccelXSelfTest
  - MPU6050, 16
- getAccelYSelfTest
  - MPU6050, 16
- getAccelZSelfTest
  - MPU6050, 16
- getAcceleration
  - MPU6050, 14
- getAccelerationX
  - MPU6050, 14
- getAccelerationY
  - MPU6050, 14
- getAccelerationZ

- MPU6050, [15](#)
- getAccelerometerPowerOnDelay
  - MPU6050, [15](#)
- getAuxVDDIOLevel
  - MPU6050, [16](#)
- getClockOutputEnabled
  - MPU6050, [16](#)
- getClockSource
  - MPU6050, [17](#)
- getConjugate
  - Quaternion, [76](#)
- getDHPFMode
  - MPU6050, [17](#)
- getDLPFMode
  - MPU6050, [18](#)
- getDMPConfig1
  - MPU6050, [18](#)
- getDMPConfig2
  - MPU6050, [18](#)
- getDMPEnabled
  - MPU6050, [18](#)
- getDMPInt0Status
  - MPU6050, [18](#)
- getDMPInt1Status
  - MPU6050, [18](#)
- getDMPInt2Status
  - MPU6050, [19](#)
- getDMPInt3Status
  - MPU6050, [19](#)
- getDMPInt4Status
  - MPU6050, [19](#)
- getDMPInt5Status
  - MPU6050, [19](#)
- getDeviceID
  - MPU6050, [17](#)
- getExternalFrameSync
  - MPU6050, [19](#)
- getExternalSensorByte
  - MPU6050, [19](#)
- getExternalSensorDWord
  - MPU6050, [20](#)
- getExternalSensorWord
  - MPU6050, [20](#)
- getExternalShadowDelayEnabled
  - MPU6050, [21](#)
- getFIFOByte
  - MPU6050, [21](#)
- getFIFOBytes
  - MPU6050, [21](#)
- getFIFOCount
  - MPU6050, [21](#)
- getFIFOEnabled
  - MPU6050, [21](#)
- getFSyncInterruptEnabled
  - MPU6050, [23](#)
- getFSyncInterruptLevel
  - MPU6050, [23](#)
- getFreefallDetectionCounterDecrement
  - MPU6050, [21](#)
- getFreefallDetectionDuration
  - MPU6050, [22](#)
- getFreefallDetectionThreshold
  - MPU6050, [22](#)
- getFullScaleAccelRange
  - MPU6050, [23](#)
- getFullScaleGyroRange
  - MPU6050, [24](#)
- getI2CBypassEnabled
  - MPU6050, [24](#)
- getI2CMasterModeEnabled
  - MPU6050, [24](#)
- getIntDMPEEnabled
  - MPU6050, [25](#)
- getIntDMPStatus
  - MPU6050, [25](#)
- getIntDataReadyEnabled
  - MPU6050, [25](#)
- getIntDataReadyStatus
  - MPU6050, [25](#)
- getIntEnabled
  - MPU6050, [25](#)
- getIntFIFOBufferOverflowEnabled
  - MPU6050, [27](#)
- getIntFIFOBufferOverflowStatus
  - MPU6050, [27](#)
- getIntFreefallEnabled
  - MPU6050, [27](#)
- getIntFreefallStatus
  - MPU6050, [28](#)
- getIntI2CMasterEnabled
  - MPU6050, [28](#)
- getIntI2CMasterStatus
  - MPU6050, [28](#)
- getIntMotionEnabled
  - MPU6050, [28](#)
- getIntMotionStatus
  - MPU6050, [29](#)
- getIntPLLReadyEnabled
  - MPU6050, [29](#)
- getIntPLLReadyStatus
  - MPU6050, [29](#)
- getIntStatus
  - MPU6050, [29](#)
- getIntZeroMotionEnabled
  - MPU6050, [29](#)
- getIntZeroMotionStatus
  - MPU6050, [30](#)
- getInterruptDrive
  - MPU6050, [26](#)
- getInterruptLatch
  - MPU6050, [26](#)
- getInterruptLatchClear
  - MPU6050, [26](#)
- getInterruptMode
  - MPU6050, [26](#)
- getLostArbitration

MPU6050, [30](#)  
getMagnitude  
  Quaternion, [76](#)  
  VectorFloat, [80](#)  
  VectorInt16, [82](#)  
getMasterClockSpeed  
  MPU6050, [30](#)  
getMotion6  
  MPU6050, [31](#)  
getMotion9  
  MPU6050, [31](#)  
getMotionDetectionCounterDecrement  
  MPU6050, [32](#)  
getMotionDetectionDuration  
  MPU6050, [32](#)  
getMotionDetectionThreshold  
  MPU6050, [32](#)  
getMultiMasterEnabled  
  MPU6050, [33](#)  
getNormalized  
  Quaternion, [76](#)  
  VectorFloat, [80](#)  
  VectorInt16, [82](#)  
getOTPBANKValid  
  MPU6050, [33](#)  
getPassthroughStatus  
  MPU6050, [33](#)  
getProduct  
  Quaternion, [76](#)  
getRate  
  MPU6050, [33](#)  
getRotated  
  VectorFloat, [80](#)  
  VectorInt16, [82](#)  
getRotation  
  MPU6050, [34](#)  
getRotationX  
  MPU6050, [34](#)  
getRotationXY  
  MPU6050, [35](#)  
getRotationY  
  MPU6050, [35](#)  
getRotationZ  
  MPU6050, [35](#)  
getSlave4InputByte  
  MPU6050, [35](#)  
getSlave0FIFOEnabled  
  MPU6050, [35](#)  
getSlave0Nack  
  MPU6050, [36](#)  
getSlave1FIFOEnabled  
  MPU6050, [36](#)  
getSlave1Nack  
  MPU6050, [36](#)  
getSlave2FIFOEnabled  
  MPU6050, [36](#)  
getSlave2Nack  
  MPU6050, [37](#)  
getSlave3FIFOEnabled  
  MPU6050, [37](#)  
getSlave3Nack  
  MPU6050, [37](#)  
getSlave4Address  
  MPU6050, [37](#)  
getSlave4Enabled  
  MPU6050, [38](#)  
getSlave4InterruptEnabled  
  MPU6050, [38](#)  
getSlave4IsDone  
  MPU6050, [38](#)  
getSlave4MasterDelay  
  MPU6050, [39](#)  
getSlave4Nack  
  MPU6050, [39](#)  
getSlave4Register  
  MPU6050, [39](#)  
getSlave4WriteMode  
  MPU6050, [39](#)  
getSlaveAddress  
  MPU6050, [40](#)  
getSlaveDataLength  
  MPU6050, [40](#)  
getSlaveDelayEnabled  
  MPU6050, [41](#)  
getSlaveEnabled  
  MPU6050, [41](#)  
getSlaveReadWriteTransitionEnabled  
  MPU6050, [42](#)  
getSlaveRegister  
  MPU6050, [42](#)  
getSlaveWordByteSwap  
  MPU6050, [42](#)  
getSlaveWordGroupOffset  
  MPU6050, [43](#)  
getSlaveWriteMode  
  MPU6050, [43](#)  
getSleepEnabled  
  MPU6050, [43](#)  
getStandbyXAccelEnabled  
  MPU6050, [44](#)  
getStandbyXGyroEnabled  
  MPU6050, [44](#)  
getStandbyYAccelEnabled  
  MPU6050, [44](#)  
getStandbyYGyroEnabled  
  MPU6050, [44](#)  
getStandbyZAccelEnabled  
  MPU6050, [44](#)  
getStandbyZGyroEnabled  
  MPU6050, [44](#)  
getTempFIFOEnabled  
  MPU6050, [44](#)  
getTempSensorEnabled  
  MPU6050, [44](#)  
getTemperature  
  MPU6050, [44](#)

- getWaitForExternalSensorEnabled
  - MPU6050, [45](#)
- getWakeCycleEnabled
  - MPU6050, [45](#)
- getWakeFrequency
  - MPU6050, [45](#)
- getXAccelOffset
  - MPU6050, [45](#)
- getXFineGain
  - MPU6050, [45](#)
- getXGyroFIFOEnabled
  - MPU6050, [46](#)
- getXGyroOffset
  - MPU6050, [46](#)
- getXGyroOffsetUser
  - MPU6050, [46](#)
- getXNegMotionDetected
  - MPU6050, [46](#)
- getXPosMotionDetected
  - MPU6050, [46](#)
- getYAccelOffset
  - MPU6050, [46](#)
- getYFineGain
  - MPU6050, [46](#)
- getYGyroFIFOEnabled
  - MPU6050, [46](#)
- getYGyroOffset
  - MPU6050, [47](#)
- getYGyroOffsetUser
  - MPU6050, [47](#)
- getYNegMotionDetected
  - MPU6050, [47](#)
- getYPosMotionDetected
  - MPU6050, [47](#)
- getZAccelOffset
  - MPU6050, [47](#)
- getZFineGain
  - MPU6050, [48](#)
- getZGyroFIFOEnabled
  - MPU6050, [48](#)
- getZGyroOffset
  - MPU6050, [49](#)
- getZGyroOffsetUser
  - MPU6050, [49](#)
- getZNegMotionDetected
  - MPU6050, [49](#)
- getZPosMotionDetected
  - MPU6050, [49](#)
- getZeroMotionDetected
  - MPU6050, [47](#)
- getZeroMotionDetectionDuration
  - MPU6050, [47](#)
- getZeroMotionDetectionThreshold
  - MPU6050, [48](#)
- helper\_3dmath.h, [88](#)
- I2C\_SPEED
  - definitions.h, [84](#)
- I2CDEV\_ARDUINO\_WIRE
  - I2Cdev.h, [107](#)
- I2CDEV\_BUILTIN\_FASTWIRE
  - I2Cdev.h, [107](#)
- I2CDEV\_BUILTIN\_NBWIRE
  - I2Cdev.h, [107](#)
- I2CDEV\_DEFAULT\_READ\_TIMEOUT
  - I2Cdev.h, [107](#)
- I2CDEV\_IMPLEMENTATION
  - I2Cdev.h, [107](#)
- I2CDEV\_IMPLEMENTATION\_WARNINGS
  - I2Cdev.h, [107](#)
- I2Cdev, [2](#)
  - I2Cdev, [3](#)
  - readBit, [3](#)
  - readBitW, [4](#)
  - readBits, [3](#)
  - readBitsW, [3](#)
  - readByte, [4](#)
  - readBytes, [4](#)
  - readTimeout, [8](#)
  - readWord, [5](#)
  - readWords, [5](#)
  - writeBit, [5](#)
  - writeBitW, [6](#)
  - writeBits, [6](#)
  - writeBitsW, [6](#)
  - writeByte, [7](#)
  - writeBytes, [7](#)
  - writeWord, [7](#)
  - writeWords, [7](#)
- I2Cdev.cpp, [91](#)
- I2Cdev.h, [106](#)
  - I2CDEV\_ARDUINO\_WIRE, [107](#)
  - I2CDEV\_BUILTIN\_FASTWIRE, [107](#)
  - I2CDEV\_BUILTIN\_NBWIRE, [107](#)
  - I2CDEV\_DEFAULT\_READ\_TIMEOUT, [107](#)
  - I2CDEV\_IMPLEMENTATION, [107](#)
  - I2CDEV\_IMPLEMENTATION\_WARNINGS, [107](#)
- INLINE\_PCINT
  - PinChangeInt.h, [179](#)
- ISR
  - PinChangeInt.h, [180](#)
- initialize
  - MPU6050, [49](#)
- LEDPIN\_OFF
  - definitions.h, [84](#)
- LEDPIN\_ON
  - definitions.h, [84](#)
- LEDPIN\_PINMODE
  - definitions.h, [84](#)
- LEDPIN\_SWITCH
  - definitions.h, [84](#)
- last
  - SerialCommand, [79](#)
- lastPinView
  - PCintPort, [75](#)
- lookupPortNumToPort

- PinChangeInt.h, [180](#)
- MAX\_RC
  - definitions.h, [84](#)
- MID\_RC
  - definitions.h, [84](#)
- MIN\_RC
  - definitions.h, [84](#)
- MOTORUPDATE\_FREQ
  - definitions.h, [84](#)
- MPU6050, [8](#)
  - buffer, [71](#)
  - devAddr, [71](#)
  - getAccelFIFOEnabled, [15](#)
  - getAccelXSelfTest, [16](#)
  - getAccelYSelfTest, [16](#)
  - getAccelZSelfTest, [16](#)
  - getAcceleration, [14](#)
  - getAccelerationX, [14](#)
  - getAccelerationY, [14](#)
  - getAccelerationZ, [15](#)
  - getAccelerometerPowerOnDelay, [15](#)
  - getAuxVDDIOLevel, [16](#)
  - getClockOutputEnabled, [16](#)
  - getClockSource, [17](#)
  - getDHPFMode, [17](#)
  - getDLPFMode, [18](#)
  - getDMPConfig1, [18](#)
  - getDMPConfig2, [18](#)
  - getDMPEEnabled, [18](#)
  - getDMPInt0Status, [18](#)
  - getDMPInt1Status, [18](#)
  - getDMPInt2Status, [19](#)
  - getDMPInt3Status, [19](#)
  - getDMPInt4Status, [19](#)
  - getDMPInt5Status, [19](#)
  - getDeviceID, [17](#)
  - getExternalFrameSync, [19](#)
  - getExternalSensorByte, [19](#)
  - getExternalSensorDWord, [20](#)
  - getExternalSensorWord, [20](#)
  - getExternalShadowDelayEnabled, [21](#)
  - getFIFOByte, [21](#)
  - getFIFOBytes, [21](#)
  - getFIFOCount, [21](#)
  - getFIFOEnabled, [21](#)
  - getFSyncInterruptEnabled, [23](#)
  - getFSyncInterruptLevel, [23](#)
  - getFreefallDetectionCounterDecrement, [21](#)
  - getFreefallDetectionDuration, [22](#)
  - getFreefallDetectionThreshold, [22](#)
  - getFullScaleAccelRange, [23](#)
  - getFullScaleGyroRange, [24](#)
  - getI2CBypassEnabled, [24](#)
  - getI2CMasterModeEnabled, [24](#)
  - getIntDMPEEnabled, [25](#)
  - getIntDMPStatus, [25](#)
  - getIntDataReadyEnabled, [25](#)
  - getIntDataReadyStatus, [25](#)
  - getIntEnabled, [25](#)
  - getIntFIFOBufferOverflowEnabled, [27](#)
  - getIntFIFOBufferOverflowStatus, [27](#)
  - getIntFreefallEnabled, [27](#)
  - getIntFreefallStatus, [28](#)
  - getIntI2CMasterEnabled, [28](#)
  - getIntI2CMasterStatus, [28](#)
  - getIntMotionEnabled, [28](#)
  - getIntMotionStatus, [29](#)
  - getIntPLLReadyEnabled, [29](#)
  - getIntPLLReadyStatus, [29](#)
  - getIntStatus, [29](#)
  - getIntZeroMotionEnabled, [29](#)
  - getIntZeroMotionStatus, [30](#)
  - getInterruptDrive, [26](#)
  - getInterruptLatch, [26](#)
  - getInterruptLatchClear, [26](#)
  - getInterruptMode, [26](#)
  - getLostArbitration, [30](#)
  - getMasterClockSpeed, [30](#)
  - getMotion6, [31](#)
  - getMotion9, [31](#)
  - getMotionDetectionCounterDecrement, [32](#)
  - getMotionDetectionDuration, [32](#)
  - getMotionDetectionThreshold, [32](#)
  - getMultiMasterEnabled, [33](#)
  - getOTPBANKValid, [33](#)
  - getPassthroughStatus, [33](#)
  - getRate, [33](#)
  - getRotation, [34](#)
  - getRotationX, [34](#)
  - getRotationXY, [35](#)
  - getRotationY, [35](#)
  - getRotationZ, [35](#)
  - getSlate4InputByte, [35](#)
  - getSlave0FIFOEnabled, [35](#)
  - getSlave0Nack, [36](#)
  - getSlave1FIFOEnabled, [36](#)
  - getSlave1Nack, [36](#)
  - getSlave2FIFOEnabled, [36](#)
  - getSlave2Nack, [37](#)
  - getSlave3FIFOEnabled, [37](#)
  - getSlave3Nack, [37](#)
  - getSlave4Address, [37](#)
  - getSlave4Enabled, [38](#)
  - getSlave4InterruptEnabled, [38](#)
  - getSlave4IsDone, [38](#)
  - getSlave4MasterDelay, [39](#)
  - getSlave4Nack, [39](#)
  - getSlave4Register, [39](#)
  - getSlave4WriteMode, [39](#)
  - getSlaveAddress, [40](#)
  - getSlaveDataLength, [40](#)
  - getSlaveDelayEnabled, [41](#)
  - getSlaveEnabled, [41](#)
  - getSlaveReadWriteTransitionEnabled, [42](#)
  - getSlaveRegister, [42](#)
  - getSlaveWordByteSwap, [42](#)



getSlaveWordGroupOffset, 43  
getSlaveWriteMode, 43  
getSleepEnabled, 43  
getStandbyXAccelEnabled, 44  
getStandbyXGyroEnabled, 44  
getStandbyYAccelEnabled, 44  
getStandbyYGyroEnabled, 44  
getStandbyZAccelEnabled, 44  
getStandbyZGyroEnabled, 44  
getTempFIFOEnabled, 44  
getTempSensorEnabled, 44  
getTemperature, 44  
getWaitForExternalSensorEnabled, 45  
getWakeCycleEnabled, 45  
getWakeFrequency, 45  
getXAccelOffset, 45  
getXFineGain, 45  
getXGyroFIFOEnabled, 46  
getXGyroOffset, 46  
getXGyroOffsetUser, 46  
getXNegMotionDetected, 46  
getXPosMotionDetected, 46  
getYAccelOffset, 46  
getYFineGain, 46  
getYGyroFIFOEnabled, 46  
getYGyroOffset, 47  
getYGyroOffsetUser, 47  
getYNegMotionDetected, 47  
getYPosMotionDetected, 47  
getZAccelOffset, 47  
getZFineGain, 48  
getZGyroFIFOEnabled, 48  
getZGyroOffset, 49  
getZGyroOffsetUser, 49  
getZNegMotionDetected, 49  
getZPosMotionDetected, 49  
getZeroMotionDetected, 47  
getZeroMotionDetectionDuration, 47  
getZeroMotionDetectionThreshold, 48  
initialize, 49  
MPU6050, 13  
readMemoryBlock, 49  
readMemoryByte, 49  
reset, 49  
resetAccelerometerPath, 50  
resetDMP, 50  
resetFIFO, 50  
resetGyroscopePath, 50  
resetI2CMaster, 50  
resetSensors, 50  
resetTemperaturePath, 51  
setAccelFIFOEnabled, 51  
setAccelXSelfTest, 51  
setAccelYSelfTest, 52  
setAccelZSelfTest, 52  
setAccelerometerPowerOnDelay, 51  
setAuxVDDIOLevel, 52  
setClockOutputEnabled, 52  
setClockSource, 53  
setDHPFMode, 53  
setDLPFMode, 53  
setDMPCfg1, 55  
setDMPCfg2, 55  
setDMPEnabled, 55  
setDeviceID, 53  
setExternalFrameSync, 55  
setExternalShadowDelayEnabled, 55  
setFIFOByte, 55  
setFIFOEnabled, 55  
setFSyncInterruptEnabled, 56  
setFSyncInterruptLevel, 57  
setFreefallDetectionCounterDecrement, 56  
setFreefallDetectionDuration, 56  
setFreefallDetectionThreshold, 56  
setFullScaleAccelRange, 57  
setFullScaleGyroRange, 57  
setI2CBypassEnabled, 57  
setI2CMasterModeEnabled, 58  
setIntDMPEnabled, 58  
setIntDataReadyEnabled, 58  
setIntEnabled, 58  
setIntFIFOBufferOverflowEnabled, 60  
setIntFreefallEnabled, 60  
setIntI2CMasterEnabled, 60  
setIntMotionEnabled, 60  
setIntPLLReadyEnabled, 61  
setIntZeroMotionEnabled, 61  
setInterruptDrive, 59  
setInterruptLatch, 59  
setInterruptLatchClear, 59  
setInterruptMode, 59  
setMasterClockSpeed, 61  
setMemoryBank, 61  
setMemoryStartAddress, 61  
setMotionDetectionCounterDecrement, 61  
setMotionDetectionDuration, 61  
setMotionDetectionThreshold, 62  
setMultiMasterEnabled, 62  
setOTPBANKValid, 62  
setRate, 62  
setSlave0FIFOEnabled, 62  
setSlave1FIFOEnabled, 63  
setSlave2FIFOEnabled, 63  
setSlave3FIFOEnabled, 63  
setSlave4Address, 63  
setSlave4Enabled, 64  
setSlave4InterruptEnabled, 64  
setSlave4MasterDelay, 64  
setSlave4OutputByte, 64  
setSlave4Register, 65  
setSlave4WriteMode, 65  
setSlaveAddress, 65  
setSlaveDataLength, 65  
setSlaveDelayEnabled, 66  
setSlaveEnabled, 66  
setSlaveOutputByte, 66

- setSlaveReadWriteTransitionEnabled, 66
- setSlaveRegister, 67
- setSlaveWordByteSwap, 67
- setSlaveWordGroupOffset, 67
- setSlaveWriteMode, 67
- setSleepEnabled, 68
- setStandbyXAccelEnabled, 68
- setStandbyXGyroEnabled, 68
- setStandbyYAccelEnabled, 68
- setStandbyYGyroEnabled, 68
- setStandbyZAccelEnabled, 68
- setStandbyZGyroEnabled, 68
- setTempFIFOEnabled, 68
- setTempSensorEnabled, 68
- setWaitForExternalSensorEnabled, 69
- setWakeCycleEnabled, 69
- setWakeFrequency, 69
- setXAccelOffset, 69
- setXFineGain, 69
- setXGyroFIFOEnabled, 69
- setXGyroOffset, 69
- setXGyroOffsetUser, 70
- setYAccelOffset, 70
- setYFineGain, 70
- setYGyroFIFOEnabled, 70
- setYGyroOffset, 70
- setYGyroOffsetUser, 70
- setZAccelOffset, 70
- setZFineGain, 70
- setZGyroFIFOEnabled, 70
- setZGyroOffset, 71
- setZGyroOffsetUser, 71
- setZeroMotionDetectionDuration, 70
- setZeroMotionDetectionThreshold, 70
- switchSPIEnabled, 71
- testConnection, 71
- writeDMPConfigurationSet, 71
- writeMemoryBlock, 71
- writeMemoryByte, 71
- writeProgDMPConfigurationSet, 71
- writeProgMemoryBlock, 71
- MPU6050.cpp, 111
- MPU6050.h, 130
  - MPU6050\_ACCEL\_FIFO\_EN\_BIT, 137
  - MPU6050\_ACCEL\_FS\_16, 137
  - MPU6050\_ACCEL\_FS\_2, 137
  - MPU6050\_ACCEL\_FS\_4, 137
  - MPU6050\_ACCEL\_FS\_8, 137
  - MPU6050\_ACONFIG\_ACCEL\_HPF\_BIT, 137
  - MPU6050\_ACONFIG\_ACCEL\_HPF\_LENGTH, 137
  - MPU6050\_ACONFIG\_AFS\_SEL\_BIT, 137
  - MPU6050\_ACONFIG\_AFS\_SEL\_LENGTH, 137
  - MPU6050\_ACONFIG\_XA\_ST\_BIT, 137
  - MPU6050\_ACONFIG\_YA\_ST\_BIT, 137
  - MPU6050\_ACONFIG\_ZA\_ST\_BIT, 137
  - MPU6050\_BANKSEL\_CFG\_USER\_BANK\_BIT, 137
  - MPU6050\_BANKSEL\_MEM\_SEL\_BIT, 137
  - MPU6050\_BANKSEL\_MEM\_SEL\_LENGTH, 138
  - MPU6050\_BANKSEL\_PRFTCH\_EN\_BIT, 138
  - MPU6050\_CFG\_DLPF\_CFG\_BIT, 138
  - MPU6050\_CFG\_DLPF\_CFG\_LENGTH, 138
  - MPU6050\_CFG\_EXT\_SYNC\_SET\_BIT, 138
  - MPU6050\_CFG\_EXT\_SYNC\_SET\_LENGTH, 138
  - MPU6050\_CLOCK\_DIV\_258, 138
  - MPU6050\_CLOCK\_DIV\_267, 138
  - MPU6050\_CLOCK\_DIV\_276, 138
  - MPU6050\_CLOCK\_DIV\_286, 138
  - MPU6050\_CLOCK\_DIV\_296, 138
  - MPU6050\_CLOCK\_DIV\_308, 138
  - MPU6050\_CLOCK\_DIV\_320, 138
  - MPU6050\_CLOCK\_DIV\_333, 138
  - MPU6050\_CLOCK\_DIV\_348, 138
  - MPU6050\_CLOCK\_DIV\_364, 139
  - MPU6050\_CLOCK\_DIV\_381, 139
  - MPU6050\_CLOCK\_DIV\_400, 139
  - MPU6050\_CLOCK\_DIV\_421, 139
  - MPU6050\_CLOCK\_DIV\_444, 139
  - MPU6050\_CLOCK\_DIV\_471, 139
  - MPU6050\_CLOCK\_DIV\_500, 139
  - MPU6050\_CLOCK\_INTERNAL, 139
  - MPU6050\_CLOCK\_KEEP\_RESET, 139
  - MPU6050\_CLOCK\_PLL\_EXT19M, 139
  - MPU6050\_CLOCK\_PLL\_EXT32K, 139
  - MPU6050\_CLOCK\_PLL\_XGYRO, 139
  - MPU6050\_CLOCK\_PLL\_YGYRO, 139
  - MPU6050\_CLOCK\_PLL\_ZGYRO, 139
  - MPU6050\_DELAYCTRL\_DELAY\_ES\_SHADOW\_BIT, 139
  - MPU6050\_DELAYCTRL\_I2C\_SLV0\_DLY\_EN\_BIT, 140
  - MPU6050\_DELAYCTRL\_I2C\_SLV1\_DLY\_EN\_BIT, 140
  - MPU6050\_DELAYCTRL\_I2C\_SLV2\_DLY\_EN\_BIT, 140
  - MPU6050\_DELAYCTRL\_I2C\_SLV3\_DLY\_EN\_BIT, 140
  - MPU6050\_DELAYCTRL\_I2C\_SLV4\_DLY\_EN\_BIT, 140
  - MPU6050\_DETECT\_ACCEL\_ON\_DELAY\_BIT, 140
  - MPU6050\_DETECT\_ACCEL\_ON\_DELAY\_LENGTH, 140
  - MPU6050\_DETECT\_DECREMENT\_1, 140
  - MPU6050\_DETECT\_DECREMENT\_2, 140
  - MPU6050\_DETECT\_DECREMENT\_4, 140
  - MPU6050\_DETECT\_DECREMENT\_RESET, 140
  - MPU6050\_DETECT\_FF\_COUNT\_BIT, 140
  - MPU6050\_DETECT\_FF\_COUNT\_LENGTH, 140
  - MPU6050\_DETECT\_MOT\_COUNT\_BIT, 140
  - MPU6050\_DETECT\_MOT\_COUNT\_LENGTH, 140
  - MPU6050\_DHPF\_0P63, 141
  - MPU6050\_DHPF\_1P25, 141
  - MPU6050\_DHPF\_2P5, 141

- MPU6050\_DHPF\_5, [141](#)
- MPU6050\_DHPF\_HOLD, [141](#)
- MPU6050\_DHPF\_RESET, [141](#)
- MPU6050\_DLPF\_BW\_10, [141](#)
- MPU6050\_DLPF\_BW\_188, [141](#)
- MPU6050\_DLPF\_BW\_20, [141](#)
- MPU6050\_DLPF\_BW\_256, [141](#)
- MPU6050\_DLPF\_BW\_42, [141](#)
- MPU6050\_DLPF\_BW\_5, [141](#)
- MPU6050\_DLPF\_BW\_98, [141](#)
- MPU6050\_DMP\_MEMORY\_BANK\_SIZE, [141](#)
- MPU6050\_DMP\_MEMORY\_BANKS, [141](#)
- MPU6050\_DMP\_MEMORY\_CHUNK\_SIZE, [142](#)
- MPU6050\_DMPINT\_0\_BIT, [142](#)
- MPU6050\_DMPINT\_1\_BIT, [142](#)
- MPU6050\_DMPINT\_2\_BIT, [142](#)
- MPU6050\_DMPINT\_3\_BIT, [142](#)
- MPU6050\_DMPINT\_4\_BIT, [142](#)
- MPU6050\_DMPINT\_5\_BIT, [142](#)
- MPU6050\_EXT\_SYNC\_ACCEL\_XOUT\_L, [142](#)
- MPU6050\_EXT\_SYNC\_ACCEL\_YOUT\_L, [142](#)
- MPU6050\_EXT\_SYNC\_ACCEL\_ZOUT\_L, [142](#)
- MPU6050\_EXT\_SYNC\_DISABLED, [142](#)
- MPU6050\_EXT\_SYNC\_GYRO\_XOUT\_L, [142](#)
- MPU6050\_EXT\_SYNC\_GYRO\_YOUT\_L, [142](#)
- MPU6050\_EXT\_SYNC\_GYRO\_ZOUT\_L, [142](#)
- MPU6050\_EXT\_SYNC\_TEMP\_OUT\_L, [142](#)
- MPU6050\_GCONFIG\_FS\_SEL\_BIT, [143](#)
- MPU6050\_GCONFIG\_FS\_SEL\_LENGTH, [143](#)
- MPU6050\_GYRO\_FS\_1000, [143](#)
- MPU6050\_GYRO\_FS\_2000, [143](#)
- MPU6050\_GYRO\_FS\_250, [143](#)
- MPU6050\_GYRO\_FS\_500, [143](#)
- MPU6050\_I2C\_MST\_CLK\_BIT, [143](#)
- MPU6050\_I2C\_MST\_CLK\_LENGTH, [143](#)
- MPU6050\_I2C\_MST\_P\_NSR\_BIT, [143](#)
- MPU6050\_I2C\_SLV4\_ADDR\_BIT, [143](#)
- MPU6050\_I2C\_SLV4\_ADDR\_LENGTH, [143](#)
- MPU6050\_I2C\_SLV4\_EN\_BIT, [143](#)
- MPU6050\_I2C\_SLV4\_INT\_EN\_BIT, [143](#)
- MPU6050\_I2C\_SLV4\_MST\_DLY\_BIT, [143](#)
- MPU6050\_I2C\_SLV4\_MST\_DLY\_LENGTH, [143](#)
- MPU6050\_I2C\_SLV4\_REG\_DIS\_BIT, [144](#)
- MPU6050\_I2C\_SLV4\_RW\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_ADDR\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_ADDR\_LENGTH, [144](#)
- MPU6050\_I2C\_SLV\_BYTE\_SW\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_EN\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_GRP\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_LEN\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_LEN\_LENGTH, [144](#)
- MPU6050\_I2C\_SLV\_REG\_DIS\_BIT, [144](#)
- MPU6050\_I2C\_SLV\_RW\_BIT, [144](#)
- MPU6050\_INTCFG\_CLKOUT\_EN\_BIT, [144](#)
- MPU6050\_INTCFG\_FSYNC\_INT\_EN\_BIT, [144](#)
- MPU6050\_INTCFG\_FSYNC\_INT\_LEVEL\_BIT, [144](#)
- MPU6050\_INTCFG\_I2C\_BYPASS\_EN\_BIT, [144](#)
- MPU6050\_INTCFG\_INT\_LEVEL\_BIT, [145](#)
- MPU6050\_INTCFG\_INT\_OPEN\_BIT, [145](#)
- MPU6050\_INTCFG\_INT\_RD\_CLEAR\_BIT, [145](#)
- MPU6050\_INTCFG\_LATCH\_INT\_EN\_BIT, [145](#)
- MPU6050\_INTCLEAR\_ANYREAD, [145](#)
- MPU6050\_INTCLEAR\_STATUSREAD, [145](#)
- MPU6050\_INTDRV\_OPENDRAIN, [145](#)
- MPU6050\_INTDRV\_PUSHPULL, [145](#)
- MPU6050\_INTERRUPT\_DATA\_RDY\_BIT, [145](#)
- MPU6050\_INTERRUPT\_DMP\_INT\_BIT, [145](#)
- MPU6050\_INTERRUPT\_FF\_BIT, [145](#)
- MPU6050\_INTERRUPT\_FIFO\_OFLOW\_BIT, [145](#)
- MPU6050\_INTERRUPT\_I2C\_MST\_INT\_BIT, [145](#)
- MPU6050\_INTERRUPT\_MOT\_BIT, [145](#)
- MPU6050\_INTERRUPT\_PLL\_RDY\_INT\_BIT, [145](#)
- MPU6050\_INTERRUPT\_ZMOT\_BIT, [146](#)
- MPU6050\_INTLATCH\_50USPULSE, [146](#)
- MPU6050\_INTLATCH\_WAITCLEAR, [146](#)
- MPU6050\_INTMODE\_ACTIVEHIGH, [146](#)
- MPU6050\_INTMODE\_ACTIVELOW, [146](#)
- MPU6050\_MOTION\_MOT\_XNEG\_BIT, [146](#)
- MPU6050\_MOTION\_MOT\_XPOS\_BIT, [146](#)
- MPU6050\_MOTION\_MOT\_YNEG\_BIT, [146](#)
- MPU6050\_MOTION\_MOT\_YPOS\_BIT, [146](#)
- MPU6050\_MOTION\_MOT\_ZNEG\_BIT, [146](#)
- MPU6050\_MOTION\_MOT\_ZPOS\_BIT, [146](#)
- MPU6050\_MOTION\_MOT\_ZRMOT\_BIT, [146](#)
- MPU6050\_MST\_I2C\_LOST\_ARB\_BIT, [146](#)
- MPU6050\_MST\_I2C\_SLV0\_NACK\_BIT, [146](#)
- MPU6050\_MST\_I2C\_SLV1\_NACK\_BIT, [146](#)
- MPU6050\_MST\_I2C\_SLV2\_NACK\_BIT, [147](#)
- MPU6050\_MST\_I2C\_SLV3\_NACK\_BIT, [147](#)
- MPU6050\_MST\_I2C\_SLV4\_DONE\_BIT, [147](#)
- MPU6050\_MST\_I2C\_SLV4\_NACK\_BIT, [147](#)
- MPU6050\_MST\_PASS\_THROUGH\_BIT, [147](#)
- MPU6050\_MULT\_MST\_EN\_BIT, [147](#)
- MPU6050\_PATHRESET\_ACCEL\_RESET\_BIT, [147](#)
- MPU6050\_PATHRESET\_GYRO\_RESET\_BIT, [147](#)
- MPU6050\_PATHRESET\_TEMP\_RESET\_BIT, [147](#)
- MPU6050\_PWR1\_CLKSEL\_BIT, [147](#)
- MPU6050\_PWR1\_CLKSEL\_LENGTH, [147](#)
- MPU6050\_PWR1\_CYCLE\_BIT, [147](#)
- MPU6050\_PWR1\_DEVICE\_RESET\_BIT, [147](#)
- MPU6050\_PWR1\_SLEEP\_BIT, [147](#)
- MPU6050\_PWR1\_TEMP\_DIS\_BIT, [147](#)
- MPU6050\_PWR2\_LP\_WAKE\_CTRL\_BIT, [148](#)
- MPU6050\_PWR2\_LP\_WAKE\_CTRL\_LENGTH, [148](#)
- MPU6050\_PWR2\_STBY\_XA\_BIT, [148](#)
- MPU6050\_PWR2\_STBY\_XG\_BIT, [148](#)
- MPU6050\_PWR2\_STBY\_YA\_BIT, [148](#)
- MPU6050\_PWR2\_STBY\_YG\_BIT, [148](#)
- MPU6050\_PWR2\_STBY\_ZA\_BIT, [148](#)
- MPU6050\_PWR2\_STBY\_ZG\_BIT, [148](#)
- MPU6050\_RA\_ACCEL\_CONFIG, [148](#)
- MPU6050\_RA\_ACCEL\_XOUT\_H, [148](#)

MPU6050\_RA\_ACCEL\_XOUT\_L, [148](#)  
MPU6050\_RA\_ACCEL\_YOUT\_H, [148](#)  
MPU6050\_RA\_ACCEL\_YOUT\_L, [148](#)  
MPU6050\_RA\_ACCEL\_ZOUT\_H, [148](#)  
MPU6050\_RA\_ACCEL\_ZOUT\_L, [148](#)  
MPU6050\_RA\_BANK\_SEL, [149](#)  
MPU6050\_RA\_CONFIG, [149](#)  
MPU6050\_RA\_DMP\_CFG\_1, [149](#)  
MPU6050\_RA\_DMP\_CFG\_2, [149](#)  
MPU6050\_RA\_DMP\_INT\_STATUS, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_00, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_01, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_02, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_03, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_04, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_05, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_06, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_07, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_08, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_09, [149](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_10, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_11, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_12, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_13, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_14, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_15, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_16, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_17, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_18, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_19, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_20, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_21, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_22, [150](#)  
MPU6050\_RA\_EXT\_SENS\_DATA\_23, [150](#)  
MPU6050\_RA\_FF\_DUR, [150](#)  
MPU6050\_RA\_FF\_THR, [151](#)  
MPU6050\_RA\_FIFO\_COUNTH, [151](#)  
MPU6050\_RA\_FIFO\_COUNTL, [151](#)  
MPU6050\_RA\_FIFO\_EN, [151](#)  
MPU6050\_RA\_FIFO\_R\_W, [151](#)  
MPU6050\_RA\_GYRO\_CONFIG, [151](#)  
MPU6050\_RA\_GYRO\_XOUT\_H, [151](#)  
MPU6050\_RA\_GYRO\_XOUT\_L, [151](#)  
MPU6050\_RA\_GYRO\_YOUT\_H, [151](#)  
MPU6050\_RA\_GYRO\_YOUT\_L, [151](#)  
MPU6050\_RA\_GYRO\_ZOUT\_H, [151](#)  
MPU6050\_RA\_GYRO\_ZOUT\_L, [151](#)  
MPU6050\_RA\_I2C\_MST\_CTRL, [151](#)  
MPU6050\_RA\_I2C\_MST\_DELAY\_CTRL, [151](#)  
MPU6050\_RA\_I2C\_MST\_STATUS, [151](#)  
MPU6050\_RA\_I2C\_SLV0\_ADDR, [152](#)  
MPU6050\_RA\_I2C\_SLV0\_CTRL, [152](#)  
MPU6050\_RA\_I2C\_SLV0\_DO, [152](#)  
MPU6050\_RA\_I2C\_SLV0\_REG, [152](#)  
MPU6050\_RA\_I2C\_SLV1\_ADDR, [152](#)  
MPU6050\_RA\_I2C\_SLV1\_CTRL, [152](#)  
MPU6050\_RA\_I2C\_SLV1\_DO, [152](#)  
MPU6050\_RA\_I2C\_SLV1\_REG, [152](#)  
MPU6050\_RA\_I2C\_SLV2\_ADDR, [152](#)  
MPU6050\_RA\_I2C\_SLV2\_CTRL, [152](#)  
MPU6050\_RA\_I2C\_SLV2\_DO, [152](#)  
MPU6050\_RA\_I2C\_SLV2\_REG, [152](#)  
MPU6050\_RA\_I2C\_SLV3\_ADDR, [152](#)  
MPU6050\_RA\_I2C\_SLV3\_CTRL, [152](#)  
MPU6050\_RA\_I2C\_SLV3\_DO, [152](#)  
MPU6050\_RA\_I2C\_SLV3\_REG, [153](#)  
MPU6050\_RA\_I2C\_SLV4\_ADDR, [153](#)  
MPU6050\_RA\_I2C\_SLV4\_CTRL, [153](#)  
MPU6050\_RA\_I2C\_SLV4\_DI, [153](#)  
MPU6050\_RA\_I2C\_SLV4\_DO, [153](#)  
MPU6050\_RA\_I2C\_SLV4\_REG, [153](#)  
MPU6050\_RA\_INT\_ENABLE, [153](#)  
MPU6050\_RA\_INT\_PIN\_CFG, [153](#)  
MPU6050\_RA\_INT\_STATUS, [153](#)  
MPU6050\_RA\_MEM\_R\_W, [153](#)  
MPU6050\_RA\_MEM\_START\_ADDR, [153](#)  
MPU6050\_RA\_MOT\_DETECT\_CTRL, [153](#)  
MPU6050\_RA\_MOT\_DETECT\_STATUS, [153](#)  
MPU6050\_RA\_MOT\_DUR, [153](#)  
MPU6050\_RA\_MOT\_THR, [153](#)  
MPU6050\_RA\_PWR\_MGMT\_1, [154](#)  
MPU6050\_RA\_PWR\_MGMT\_2, [154](#)  
MPU6050\_RA\_SIGNAL\_PATH\_RESET, [154](#)  
MPU6050\_RA\_SPLRT\_DIV, [154](#)  
MPU6050\_RA\_TEMP\_OUT\_H, [154](#)  
MPU6050\_RA\_TEMP\_OUT\_L, [154](#)  
MPU6050\_RA\_USER\_CTRL, [154](#)  
MPU6050\_RA\_WHO\_AM\_I, [154](#)  
MPU6050\_RA\_X\_FINE\_GAIN, [154](#)  
MPU6050\_RA\_XA\_OFFS\_H, [154](#)  
MPU6050\_RA\_XA\_OFFS\_L\_TC, [154](#)  
MPU6050\_RA\_XG\_OFFS\_TC, [154](#)  
MPU6050\_RA\_XG\_OFFS\_USRH, [154](#)  
MPU6050\_RA\_XG\_OFFS\_USRL, [154](#)  
MPU6050\_RA\_Y\_FINE\_GAIN, [154](#)  
MPU6050\_RA\_YA\_OFFS\_H, [155](#)  
MPU6050\_RA\_YA\_OFFS\_L\_TC, [155](#)  
MPU6050\_RA\_YG\_OFFS\_TC, [155](#)  
MPU6050\_RA\_YG\_OFFS\_USRH, [155](#)  
MPU6050\_RA\_YG\_OFFS\_USRL, [155](#)  
MPU6050\_RA\_Z\_FINE\_GAIN, [155](#)  
MPU6050\_RA\_ZA\_OFFS\_H, [155](#)  
MPU6050\_RA\_ZA\_OFFS\_L\_TC, [155](#)  
MPU6050\_RA\_ZG\_OFFS\_TC, [155](#)  
MPU6050\_RA\_ZG\_OFFS\_USRH, [155](#)  
MPU6050\_RA\_ZG\_OFFS\_USRL, [155](#)  
MPU6050\_RA\_ZRMOT\_DUR, [155](#)  
MPU6050\_RA\_ZRMOT\_THR, [155](#)  
MPU6050\_SLV0\_FIFO\_EN\_BIT, [155](#)  
MPU6050\_SLV1\_FIFO\_EN\_BIT, [155](#)  
MPU6050\_SLV2\_FIFO\_EN\_BIT, [156](#)  
MPU6050\_SLV3\_FIFO\_EN\_BIT, [156](#)  
MPU6050\_TC\_OFFSET\_BIT, [156](#)  
MPU6050\_TC\_OFFSET\_LENGTH, [156](#)  
MPU6050\_TC\_OTP\_BNK\_VLD\_BIT, [156](#)  
MPU6050\_TC\_PWR\_MODE\_BIT, [156](#)

- MPU6050\_TEMP\_FIFO\_EN\_BIT, [156](#)
- MPU6050\_USERCTRL\_DMP\_EN\_BIT, [156](#)
- MPU6050\_USERCTRL\_DMP\_RESET\_BIT, [156](#)
- MPU6050\_USERCTRL\_FIFO\_EN\_BIT, [156](#)
- MPU6050\_USERCTRL\_FIFO\_RESET\_BIT, [156](#)
- MPU6050\_USERCTRL\_I2C\_IF\_DIS\_BIT, [156](#)
- MPU6050\_USERCTRL\_I2C\_MST\_EN\_BIT, [156](#)
- MPU6050\_USERCTRL\_I2C\_MST\_RESET\_BIT, [156](#)
- MPU6050\_USERCTRL\_SIG\_COND\_RESET\_BIT, [156](#)
- MPU6050\_VDDIO\_LEVEL\_VDD, [157](#)
- MPU6050\_VDDIO\_LEVEL\_VLOGIC, [157](#)
- MPU6050\_WAIT\_FOR\_ES\_BIT, [157](#)
- MPU6050\_WAKE\_FREQ\_10, [157](#)
- MPU6050\_WAKE\_FREQ\_1P25, [157](#)
- MPU6050\_WAKE\_FREQ\_2P5, [157](#)
- MPU6050\_WAKE\_FREQ\_5, [157](#)
- MPU6050\_WHO\_AM\_I\_BIT, [157](#)
- MPU6050\_WHO\_AM\_I\_LENGTH, [157](#)
- MPU6050\_XG\_FIFO\_EN\_BIT, [157](#)
- MPU6050\_YG\_FIFO\_EN\_BIT, [157](#)
- MPU6050\_ZG\_FIFO\_EN\_BIT, [157](#)
- MPU6050\_6Axis\_DMP.h, [169](#)
- MPU6050\_DMP\_CODE\_SIZE, [170](#)
- MPU6050\_DMP\_CONFIG\_SIZE, [170](#)
- MPU6050\_DMP\_UPDATES\_SIZE, [170](#)
- MPU6050\_INCLUDE\_DMP\_MOTIONAPPS20, [170](#)
- PROGMEM, [170](#)
- MPU6050\_ACCEL\_FIFO\_EN\_BIT
- MPU6050.h, [137](#)
- MPU6050\_ACCEL\_FS\_16
- MPU6050.h, [137](#)
- MPU6050\_ACCEL\_FS\_2
- MPU6050.h, [137](#)
- MPU6050\_ACCEL\_FS\_4
- MPU6050.h, [137](#)
- MPU6050\_ACCEL\_FS\_8
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_ACCEL\_HPF\_BIT
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_ACCEL\_HPF\_LENGTH
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_AFS\_SEL\_BIT
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_AFS\_SEL\_LENGTH
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_XA\_ST\_BIT
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_YA\_ST\_BIT
- MPU6050.h, [137](#)
- MPU6050\_ACONFIG\_ZA\_ST\_BIT
- MPU6050.h, [137](#)
- MPU6050\_ADDRESS\_AD0\_HIGH
- definitions.h, [85](#)
- MPU6050\_ADDRESS\_AD0\_LOW
- definitions.h, [85](#)
- MPU6050\_BANKSEL\_CFG\_USER\_BANK\_BIT
- MPU6050.h, [137](#)
- MPU6050\_BANKSEL\_MEM\_SEL\_BIT
- MPU6050.h, [137](#)
- MPU6050\_BANKSEL\_MEM\_SEL\_LENGTH
- MPU6050.h, [138](#)
- MPU6050\_BANKSEL\_PRFTCH\_EN\_BIT
- MPU6050.h, [138](#)
- MPU6050\_CFG\_DLPF\_CFG\_BIT
- MPU6050.h, [138](#)
- MPU6050\_CFG\_DLPF\_CFG\_LENGTH
- MPU6050.h, [138](#)
- MPU6050\_CFG\_EXT\_SYNC\_SET\_BIT
- MPU6050.h, [138](#)
- MPU6050\_CFG\_EXT\_SYNC\_SET\_LENGTH
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_258
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_267
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_276
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_286
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_296
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_308
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_320
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_333
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_348
- MPU6050.h, [138](#)
- MPU6050\_CLOCK\_DIV\_364
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_DIV\_381
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_DIV\_400
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_DIV\_421
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_DIV\_444
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_DIV\_471
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_DIV\_500
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_INTERNAL
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_KEEP\_RESET
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_PLL\_EXT19M
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_PLL\_EXT32K
- MPU6050.h, [139](#)
- MPU6050\_CLOCK\_PLL\_XGYRO
- MPU6050.h, [139](#)



MPU6050\_CLOCK\_PLL\_YGYRO  
MPU6050.h, [139](#)

MPU6050\_CLOCK\_PLL\_ZGYRO  
MPU6050.h, [139](#)

MPU6050\_DEFAULT\_ADDRESS  
definitions.h, [85](#)

MPU6050\_DELAYCTRL\_DELAY\_ES\_SHADOW\_BIT  
MPU6050.h, [139](#)

MPU6050\_DELAYCTRL\_I2C\_SLV0\_DLY\_EN\_BIT  
MPU6050.h, [140](#)

MPU6050\_DELAYCTRL\_I2C\_SLV1\_DLY\_EN\_BIT  
MPU6050.h, [140](#)

MPU6050\_DELAYCTRL\_I2C\_SLV2\_DLY\_EN\_BIT  
MPU6050.h, [140](#)

MPU6050\_DELAYCTRL\_I2C\_SLV3\_DLY\_EN\_BIT  
MPU6050.h, [140](#)

MPU6050\_DELAYCTRL\_I2C\_SLV4\_DLY\_EN\_BIT  
MPU6050.h, [140](#)

MPU6050\_DETECT\_ACCEL\_ON\_DELAY\_BIT  
MPU6050.h, [140](#)

MPU6050\_DETECT\_ACCEL\_ON\_DELAY\_LENGTH  
MPU6050.h, [140](#)

MPU6050\_DETECT\_DECREMENT\_1  
MPU6050.h, [140](#)

MPU6050\_DETECT\_DECREMENT\_2  
MPU6050.h, [140](#)

MPU6050\_DETECT\_DECREMENT\_4  
MPU6050.h, [140](#)

MPU6050\_DETECT\_DECREMENT\_RESET  
MPU6050.h, [140](#)

MPU6050\_DETECT\_FF\_COUNT\_BIT  
MPU6050.h, [140](#)

MPU6050\_DETECT\_FF\_COUNT\_LENGTH  
MPU6050.h, [140](#)

MPU6050\_DETECT\_MOT\_COUNT\_BIT  
MPU6050.h, [140](#)

MPU6050\_DETECT\_MOT\_COUNT\_LENGTH  
MPU6050.h, [140](#)

MPU6050\_DHPF\_0P63  
MPU6050.h, [141](#)

MPU6050\_DHPF\_1P25  
MPU6050.h, [141](#)

MPU6050\_DHPF\_2P5  
MPU6050.h, [141](#)

MPU6050\_DHPF\_5  
MPU6050.h, [141](#)

MPU6050\_DHPF\_HOLD  
MPU6050.h, [141](#)

MPU6050\_DHPF\_RESET  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW  
definitions.h, [85](#)

MPU6050\_DLPF\_BW\_10  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW\_188  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW\_20  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW\_256  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW\_42  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW\_5  
MPU6050.h, [141](#)

MPU6050\_DLPF\_BW\_98  
MPU6050.h, [141](#)

MPU6050\_DMP\_CODE\_SIZE  
MPU6050\_6Axis\_DMP.h, [170](#)

MPU6050\_DMP\_CONFIG\_SIZE  
MPU6050\_6Axis\_DMP.h, [170](#)

MPU6050\_DMP\_MEMORY\_BANK\_SIZE  
MPU6050.h, [141](#)

MPU6050\_DMP\_MEMORY\_BANKS  
MPU6050.h, [141](#)

MPU6050\_DMP\_MEMORY\_CHUNK\_SIZE  
MPU6050.h, [142](#)

MPU6050\_DMP\_UPDATES\_SIZE  
MPU6050\_6Axis\_DMP.h, [170](#)

MPU6050\_DMPINT\_0\_BIT  
MPU6050.h, [142](#)

MPU6050\_DMPINT\_1\_BIT  
MPU6050.h, [142](#)

MPU6050\_DMPINT\_2\_BIT  
MPU6050.h, [142](#)

MPU6050\_DMPINT\_3\_BIT  
MPU6050.h, [142](#)

MPU6050\_DMPINT\_4\_BIT  
MPU6050.h, [142](#)

MPU6050\_DMPINT\_5\_BIT  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_ACCEL\_XOUT\_L  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_ACCEL\_YOUT\_L  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_ACCEL\_ZOUT\_L  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_DISABLED  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_GYRO\_XOUT\_L  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_GYRO\_YOUT\_L  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_GYRO\_ZOUT\_L  
MPU6050.h, [142](#)

MPU6050\_EXT\_SYNC\_TEMP\_OUT\_L  
MPU6050.h, [142](#)

MPU6050\_GCONFIG\_FS\_SEL\_BIT  
MPU6050.h, [143](#)

MPU6050\_GCONFIG\_FS\_SEL\_LENGTH  
MPU6050.h, [143](#)

MPU6050\_GYRO\_FS  
definitions.h, [85](#)

MPU6050\_GYRO\_FS\_1000  
MPU6050.h, [143](#)

MPU6050\_GYRO\_FS\_2000  
MPU6050.h, [143](#)

|  |                                   |
|--|-----------------------------------|
| MPU6050_GYRO_FS_250                      | MPU6050_INTCFG_INT_RD_CLEAR_BIT   |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_GYRO_FS_500                      | MPU6050_INTCFG_LATCH_INT_EN_BIT   |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_MST_CLK_BIT                  | MPU6050_INTCLEAR_ANYREAD          |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_MST_CLK_LENGTH               | MPU6050_INTCLEAR_STATUSREAD       |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_MST_P_NSR_BIT                | MPU6050_INTDRV_OPENDRAIN          |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_ADDR_BIT                | MPU6050_INTDRV_PUSH_PULL          |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_ADDR_LENGTH             | MPU6050_INTERRUPT_DATA_RDY_BIT    |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_EN_BIT                  | MPU6050_INTERRUPT_DMP_INT_BIT     |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_INT_EN_BIT              | MPU6050_INTERRUPT_FF_BIT          |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_MST_DLY_BIT             | MPU6050_INTERRUPT_FIFO_OFLOW_BIT  |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_MST_DLY_LENGTH          | MPU6050_INTERRUPT_I2C_MST_INT_BIT |
| MPU6050.h, <a href="#">143</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_REG_DIS_BIT             | MPU6050_INTERRUPT_MOT_BIT         |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV4_RW_BIT                  | MPU6050_INTERRUPT_PLL_RDY_INT_BIT |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">145</a>    |
| MPU6050_I2C_SLV_ADDR_BIT                 | MPU6050_INTERRUPT_ZMOT_BIT        |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_ADDR_LENGTH              | MPU6050_INTLATCH_50USPULSE        |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_BYTE_SW_BIT              | MPU6050_INTLATCH_WAITCLEAR        |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_EN_BIT                   | MPU6050_INTMODE_ACTIVEHIGH        |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_GRP_BIT                  | MPU6050_INTMODE_ACTIVELOW         |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_LEN_BIT                  | MPU6050_MOTION_MOT_XNEG_BIT       |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_LEN_LENGTH               | MPU6050_MOTION_MOT_XPOS_BIT       |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_REG_DIS_BIT              | MPU6050_MOTION_MOT_YNEG_BIT       |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_I2C_SLV_RW_BIT                   | MPU6050_MOTION_MOT_YPOS_BIT       |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INCLUDE_DMP_MOTIONAPPS20         | MPU6050_MOTION_MOT_ZNEG_BIT       |
| MPU6050_6Axis_DMP.h, <a href="#">170</a> | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INTCFG_CLKOUT_EN_BIT             | MPU6050_MOTION_MOT_ZPOS_BIT       |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INTCFG_FSYNC_INT_EN_BIT          | MPU6050_MOTION_MOT_ZRMOT_BIT      |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT       | MPU6050_MST_I2C_LOST_ARB_BIT      |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INTCFG_I2C_BYPASS_EN_BIT         | MPU6050_MST_I2C_SLV0_NACK_BIT     |
| MPU6050.h, <a href="#">144</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INTCFG_INT_LEVEL_BIT             | MPU6050_MST_I2C_SLV1_NACK_BIT     |
| MPU6050.h, <a href="#">145</a>           | MPU6050.h, <a href="#">146</a>    |
| MPU6050_INTCFG_INT_OPEN_BIT              | MPU6050_MST_I2C_SLV2_NACK_BIT     |
| MPU6050.h, <a href="#">145</a>           | MPU6050.h, <a href="#">147</a>    |

MPU6050\_MST\_I2C\_SLV3\_NACK\_BIT  
MPU6050.h, [147](#)

MPU6050\_MST\_I2C\_SLV4\_DONE\_BIT  
MPU6050.h, [147](#)

MPU6050\_MST\_I2C\_SLV4\_NACK\_BIT  
MPU6050.h, [147](#)

MPU6050\_MST\_PASS\_THROUGH\_BIT  
MPU6050.h, [147](#)

MPU6050\_MULT\_MST\_EN\_BIT  
MPU6050.h, [147](#)

MPU6050\_PATHRESET\_ACCEL\_RESET\_BIT  
MPU6050.h, [147](#)

MPU6050\_PATHRESET\_GYRO\_RESET\_BIT  
MPU6050.h, [147](#)

MPU6050\_PATHRESET\_TEMP\_RESET\_BIT  
MPU6050.h, [147](#)

MPU6050\_PWR1\_CLKSEL\_BIT  
MPU6050.h, [147](#)

MPU6050\_PWR1\_CLKSEL\_LENGTH  
MPU6050.h, [147](#)

MPU6050\_PWR1\_CYCLE\_BIT  
MPU6050.h, [147](#)

MPU6050\_PWR1\_DEVICE\_RESET\_BIT  
MPU6050.h, [147](#)

MPU6050\_PWR1\_SLEEP\_BIT  
MPU6050.h, [147](#)

MPU6050\_PWR1\_TEMP\_DIS\_BIT  
MPU6050.h, [147](#)

MPU6050\_PWR2\_LP\_WAKE\_CTRL\_BIT  
MPU6050.h, [148](#)

MPU6050\_PWR2\_LP\_WAKE\_CTRL\_LENGTH  
MPU6050.h, [148](#)

MPU6050\_PWR2\_STBY\_XA\_BIT  
MPU6050.h, [148](#)

MPU6050\_PWR2\_STBY\_XG\_BIT  
MPU6050.h, [148](#)

MPU6050\_PWR2\_STBY\_YA\_BIT  
MPU6050.h, [148](#)

MPU6050\_PWR2\_STBY\_YG\_BIT  
MPU6050.h, [148](#)

MPU6050\_PWR2\_STBY\_ZA\_BIT  
MPU6050.h, [148](#)

MPU6050\_PWR2\_STBY\_ZG\_BIT  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_CONFIG  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_XOUT\_H  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_XOUT\_L  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_YOUT\_H  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_YOUT\_L  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_ZOUT\_H  
MPU6050.h, [148](#)

MPU6050\_RA\_ACCEL\_ZOUT\_L  
MPU6050.h, [148](#)

MPU6050\_RA\_BANK\_SEL  
MPU6050.h, [149](#)

MPU6050\_RA\_CONFIG  
MPU6050.h, [149](#)

MPU6050\_RA\_DMP\_CFG\_1  
MPU6050.h, [149](#)

MPU6050\_RA\_DMP\_CFG\_2  
MPU6050.h, [149](#)

MPU6050\_RA\_DMP\_INT\_STATUS  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_00  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_01  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_02  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_03  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_04  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_05  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_06  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_07  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_08  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_09  
MPU6050.h, [149](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_10  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_11  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_12  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_13  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_14  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_15  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_16  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_17  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_18  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_19  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_20  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_21  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_22  
MPU6050.h, [150](#)

MPU6050\_RA\_EXT\_SENS\_DATA\_23  
MPU6050.h, [150](#)



|                                |                                |
|--------------------------------|--------------------------------|
| MPU6050_RA_FF_DUR              | MPU6050_RA_I2C_SLV3_CTRL       |
| MPU6050.h, <a href="#">150</a> | MPU6050.h, <a href="#">152</a> |
| MPU6050_RA_FF_THR              | MPU6050_RA_I2C_SLV3_DO         |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">152</a> |
| MPU6050_RA_FIFO_COUNTH         | MPU6050_RA_I2C_SLV3_REG        |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_FIFO_COUNTL         | MPU6050_RA_I2C_SLV4_ADDR       |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_FIFO_EN             | MPU6050_RA_I2C_SLV4_CTRL       |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_FIFO_R_W            | MPU6050_RA_I2C_SLV4_DI         |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_CONFIG         | MPU6050_RA_I2C_SLV4_DO         |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_XOUT_H         | MPU6050_RA_I2C_SLV4_REG        |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_XOUT_L         | MPU6050_RA_INT_ENABLE          |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_YOUT_H         | MPU6050_RA_INT_PIN_CFG         |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_YOUT_L         | MPU6050_RA_INT_STATUS          |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_ZOUT_H         | MPU6050_RA_MEM_R_W             |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_GYRO_ZOUT_L         | MPU6050_RA_MEM_START_ADDR      |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_I2C_MST_CTRL        | MPU6050_RA_MOT_DETECT_CTRL     |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_I2C_MST_DELAY_CTRL  | MPU6050_RA_MOT_DETECT_STATUS   |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_I2C_MST_STATUS      | MPU6050_RA_MOT_DUR             |
| MPU6050.h, <a href="#">151</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_I2C_SLV0_ADDR       | MPU6050_RA_MOT_THR             |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">153</a> |
| MPU6050_RA_I2C_SLV0_CTRL       | MPU6050_RA_PWR_MGMT_1          |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV0_DO         | MPU6050_RA_PWR_MGMT_2          |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV0_REG        | MPU6050_RA_SIGNAL_PATH_RESET   |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV1_ADDR       | MPU6050_RA_SMPLRT_DIV          |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV1_CTRL       | MPU6050_RA_TEMP_OUT_H          |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV1_DO         | MPU6050_RA_TEMP_OUT_L          |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV1_REG        | MPU6050_RA_USER_CTRL           |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV2_ADDR       | MPU6050_RA_WHO_AM_I            |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV2_CTRL       | MPU6050_RA_X_FINE_GAIN         |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV2_DO         | MPU6050_RA_XA_OFFS_H           |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV2_REG        | MPU6050_RA_XA_OFFS_L_TC        |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |
| MPU6050_RA_I2C_SLV3_ADDR       | MPU6050_RA_XG_OFFS_TC          |
| MPU6050.h, <a href="#">152</a> | MPU6050.h, <a href="#">154</a> |

- MPU6050\_RA\_XG\_OFFS\_USRH  
MPU6050.h, [154](#)
- MPU6050\_RA\_XG\_OFFS\_USRL  
MPU6050.h, [154](#)
- MPU6050\_RA\_Y\_FINE\_GAIN  
MPU6050.h, [154](#)
- MPU6050\_RA\_YA\_OFFS\_H  
MPU6050.h, [155](#)
- MPU6050\_RA\_YA\_OFFS\_L\_TC  
MPU6050.h, [155](#)
- MPU6050\_RA\_YG\_OFFS\_TC  
MPU6050.h, [155](#)
- MPU6050\_RA\_YG\_OFFS\_USRH  
MPU6050.h, [155](#)
- MPU6050\_RA\_YG\_OFFS\_USRL  
MPU6050.h, [155](#)
- MPU6050\_RA\_Z\_FINE\_GAIN  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZA\_OFFS\_H  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZA\_OFFS\_L\_TC  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZG\_OFFS\_TC  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZG\_OFFS\_USRH  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZG\_OFFS\_USRL  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZRMOT\_DUR  
MPU6050.h, [155](#)
- MPU6050\_RA\_ZRMOT\_THR  
MPU6050.h, [155](#)
- MPU6050\_SLV0\_FIFO\_EN\_BIT  
MPU6050.h, [155](#)
- MPU6050\_SLV1\_FIFO\_EN\_BIT  
MPU6050.h, [155](#)
- MPU6050\_SLV2\_FIFO\_EN\_BIT  
MPU6050.h, [156](#)
- MPU6050\_SLV\_3\_FIFO\_EN\_BIT  
MPU6050.h, [156](#)
- MPU6050\_TC\_OFFSET\_BIT  
MPU6050.h, [156](#)
- MPU6050\_TC\_OFFSET\_LENGTH  
MPU6050.h, [156](#)
- MPU6050\_TC\_OTP\_BNK\_VLD\_BIT  
MPU6050.h, [156](#)
- MPU6050\_TC\_PWR\_MODE\_BIT  
MPU6050.h, [156](#)
- MPU6050\_TEMP\_FIFO\_EN\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_DMP\_EN\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_DMP\_RESET\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_FIFO\_EN\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_FIFO\_RESET\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_I2C\_IF\_DIS\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_I2C\_MST\_EN\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_I2C\_MST\_RESET\_BIT  
MPU6050.h, [156](#)
- MPU6050\_USERCTRL\_SIG\_COND\_RESET\_BIT  
MPU6050.h, [156](#)
- MPU6050\_VDDIO\_LEVEL\_VDD  
MPU6050.h, [157](#)
- MPU6050\_VDDIO\_LEVEL\_VLOGIC  
MPU6050.h, [157](#)
- MPU6050\_WAIT\_FOR\_ES\_BIT  
MPU6050.h, [157](#)
- MPU6050\_WAKE\_FREQ\_10  
MPU6050.h, [157](#)
- MPU6050\_WAKE\_FREQ\_1P25  
MPU6050.h, [157](#)
- MPU6050\_WAKE\_FREQ\_2P5  
MPU6050.h, [157](#)
- MPU6050\_WAKE\_FREQ\_5  
MPU6050.h, [157](#)
- MPU6050\_WHO\_AM\_I\_BIT  
MPU6050.h, [157](#)
- MPU6050\_WHO\_AM\_I\_LENGTH  
MPU6050.h, [157](#)
- MPU6050\_XG\_FIFO\_EN\_BIT  
MPU6050.h, [157](#)
- MPU6050\_YG\_FIFO\_EN\_BIT  
MPU6050.h, [157](#)
- MPU6050\_ZG\_FIFO\_EN\_BIT  
MPU6050.h, [157](#)
- mask  
PCintPort::PCintPin, [72](#)
- mode  
PCintPort::PCintPin, [72](#)
- N\_SIN  
definitions.h, [85](#)
- NO\_PORTA\_PINCHANGES  
PinChangeInt.h, [179](#)
- next  
PCintPort::PCintPin, [72](#)  
SerialCommand, [78](#)
- normalize  
Quaternion, [76](#)  
VectorFloat, [80](#)  
VectorInt16, [82](#)
- PCICRbit  
PCintPort, [75](#)
- PCINT\_VERSION  
PinChangeInt.h, [180](#)
- PCIntvoidFuncPtr  
PinChangeInt.h, [180](#)
- PCattachInterrupt  
PinChangeInt.h, [180](#)
- PCdetachInterrupt  
PinChangeInt.h, [180](#)

- PCgetArduinoPin
  - PinChangeInt.h, 180
- PCInt
  - PCIntPort, 74
- PCIntFunc
  - PCIntPort::PCIntPin, 72
- PCIntPin
  - PCIntPort::PCIntPin, 72
- PCIntPort, 73
  - addPin, 74
  - arduinoPin, 74
  - attachInterrupt, 74
  - curr, 74
  - detachInterrupt, 74
  - enable, 74
  - firstPin, 74
  - lastPinView, 75
  - PCICRbit, 75
  - PCInt, 74
  - PCIntPort, 74
  - pinState, 75
  - portFallingPins, 75
  - portInputReg, 75
  - portPCMask, 75
  - portRisingPins, 75
- PCIntPort::PCIntPin, 72
  - arduinoPin, 72
  - mask, 72
  - mode, 72
  - next, 72
  - PCIntFunc, 72
  - PCIntPin, 72
- PORTBVECT
  - PinChangeInt.h, 180
- PORTCVECT
  - PinChangeInt.h, 180
- PORTDVECT
  - PinChangeInt.h, 180
- PROGMEM
  - MPU6050\_6Axis\_DMP.h, 170
- PWM\_32KHZ\_PHASE
  - definitions.h, 85
- PWM\_A\_MOTOR0
  - definitions.h, 85
- PWM\_A\_MOTOR1
  - definitions.h, 85
- PWM\_B\_MOTOR0
  - definitions.h, 85
- PWM\_B\_MOTOR1
  - definitions.h, 85
- PWM\_C\_MOTOR0
  - definitions.h, 85
- PWM\_C\_MOTOR1
  - definitions.h, 85
- PinChangeInt.h, 178
  - INLINE\_PCINT, 179
  - ISR, 180
  - lookupPortNumToPort, 180
  - NO\_PORTA\_PINCHANGES, 179
  - PCINT\_VERSION, 180
  - PCIntvoidFuncPtr, 180
  - PCattachInterrupt, 180
  - PCdetachInterrupt, 180
  - PCgetArduinoPin, 180
  - PORTBVECT, 180
  - PORTCVECT, 180
  - PORTDVECT, 180
  - portB, 180
  - portC, 181
  - portD, 181
- pinState
  - PCIntPort, 75
- portB
  - PinChangeInt.h, 180
- portC
  - PinChangeInt.h, 181
- portD
  - PinChangeInt.h, 181
- portFallingPins
  - PCIntPort, 75
- portInputReg
  - PCIntPort, 75
- portPCMask
  - PCIntPort, 75
- portRisingPins
  - PCIntPort, 75
- Quaternion, 75
  - getConjugate, 76
  - getMagnitude, 76
  - getNormalized, 76
  - getProduct, 76
  - normalize, 76
  - Quaternion, 76
  - w, 76
  - x, 76
  - y, 76
  - z, 76
- RC\_DEADBAND
  - definitions.h, 85
- RC\_PIN\_PITCH
  - definitions.h, 85
- RC\_PIN\_ROLL
  - definitions.h, 86
- readBit
  - I2Cdev, 3
- readBitW
  - I2Cdev, 4
- readBits
  - I2Cdev, 3
- readBitsW
  - I2Cdev, 3
- readByte
  - I2Cdev, 4
- readBytes
  - I2Cdev, 4

- readMemoryBlock
  - MPU6050, [49](#)
- readMemoryByte
  - MPU6050, [49](#)
- readSerial
  - SerialCommand, [78](#)
- readTimeout
  - I2Cdev, [8](#)
- readWord
  - I2Cdev, [5](#)
- readWords
  - I2Cdev, [5](#)
- reset
  - MPU6050, [49](#)
- resetAccelerometerPath
  - MPU6050, [50](#)
- resetDMP
  - MPU6050, [50](#)
- resetFIFO
  - MPU6050, [50](#)
- resetGyroscopePath
  - MPU6050, [50](#)
- resetI2CMaster
  - MPU6050, [50](#)
- resetSensors
  - MPU6050, [50](#)
- resetTemperaturePath
  - MPU6050, [51](#)
- rotate
  - VectorFloat, [80](#)
  - VectorInt16, [82](#)
- SCALE\_ACC
  - definitions.h, [86](#)
- SCALE\_PID\_PARAMS
  - definitions.h, [86](#)
- SERIALCOMMAND\_BUFFER
  - SerialCommand.h, [191](#)
- SERIALCOMMAND\_MAXCOMMANDENGTH
  - SerialCommand.h, [192](#)
- SerialCommand, [77](#)
  - addCommand, [78](#)
  - bufPos, [79](#)
  - buffer, [78](#)
  - clearBuffer, [78](#)
  - commandCount, [79](#)
  - commandList, [79](#)
  - defaultHandler, [79](#)
  - delim, [79](#)
  - last, [79](#)
  - next, [78](#)
  - readSerial, [78](#)
  - SerialCommand, [78](#)
  - setDefaultHandler, [78](#)
  - term, [79](#)
- SerialCommand.cpp, [189](#)
- SerialCommand.h, [190](#)
  - SERIALCOMMAND\_BUFFER, [191](#)
  - SERIALCOMMAND\_MAXCOMMANDENGTH, [192](#)
- SerialCommand::SerialCommandCallback, [79](#)
  - command, [79](#)
  - function, [79](#)
- setAccelFIFOEnabled
  - MPU6050, [51](#)
- setAccelXSelfTest
  - MPU6050, [51](#)
- setAccelYSelfTest
  - MPU6050, [52](#)
- setAccelZSelfTest
  - MPU6050, [52](#)
- setAccelerometerPowerOnDelay
  - MPU6050, [51](#)
- setAuxVDDIOLevel
  - MPU6050, [52](#)
- setClockOutputEnabled
  - MPU6050, [52](#)
- setClockSource
  - MPU6050, [53](#)
- setDHPFMode
  - MPU6050, [53](#)
- setDLPFMode
  - MPU6050, [53](#)
- setDMPConfig1
  - MPU6050, [55](#)
- setDMPConfig2
  - MPU6050, [55](#)
- setDMPEEnabled
  - MPU6050, [55](#)
- setDefaultHandler
  - SerialCommand, [78](#)
- setDeviceID
  - MPU6050, [53](#)
- setExternalFrameSync
  - MPU6050, [55](#)
- setExternalShadowDelayEnabled
  - MPU6050, [55](#)
- setFIFOByte
  - MPU6050, [55](#)
- setFIFOEnabled
  - MPU6050, [55](#)
- setFSyncInterruptEnabled
  - MPU6050, [56](#)
- setFSyncInterruptLevel
  - MPU6050, [57](#)
- setFreefallDetectionCounterDecrement
  - MPU6050, [56](#)
- setFreefallDetectionDuration
  - MPU6050, [56](#)
- setFreefallDetectionThreshold
  - MPU6050, [56](#)
- setFullScaleAccelRange
  - MPU6050, [57](#)
- setFullScaleGyroRange
  - MPU6050, [57](#)
- setI2CBypassEnabled

MPU6050, [57](#)  
 setI2CMasterModeEnabled  
     MPU6050, [58](#)  
 setIntDMPEnabled  
     MPU6050, [58](#)  
 setIntDataReadyEnabled  
     MPU6050, [58](#)  
 setIntEnabled  
     MPU6050, [58](#)  
 setIntFIFOBufferOverflowEnabled  
     MPU6050, [60](#)  
 setIntFreefallEnabled  
     MPU6050, [60](#)  
 setIntI2CMasterEnabled  
     MPU6050, [60](#)  
 setIntMotionEnabled  
     MPU6050, [60](#)  
 setIntPLLReadyEnabled  
     MPU6050, [61](#)  
 setIntZeroMotionEnabled  
     MPU6050, [61](#)  
 setInterruptDrive  
     MPU6050, [59](#)  
 setInterruptLatch  
     MPU6050, [59](#)  
 setInterruptLatchClear  
     MPU6050, [59](#)  
 setInterruptMode  
     MPU6050, [59](#)  
 setMasterClockSpeed  
     MPU6050, [61](#)  
 setMemoryBank  
     MPU6050, [61](#)  
 setMemoryStartAddress  
     MPU6050, [61](#)  
 setMotionDetectionCounterDecrement  
     MPU6050, [61](#)  
 setMotionDetectionDuration  
     MPU6050, [61](#)  
 setMotionDetectionThreshold  
     MPU6050, [62](#)  
 setMultiMasterEnabled  
     MPU6050, [62](#)  
 setOTPBANKValid  
     MPU6050, [62](#)  
 setRate  
     MPU6050, [62](#)  
 setSlave0FIFOEnabled  
     MPU6050, [62](#)  
 setSlave1FIFOEnabled  
     MPU6050, [63](#)  
 setSlave2FIFOEnabled  
     MPU6050, [63](#)  
 setSlave3FIFOEnabled  
     MPU6050, [63](#)  
 setSlave4Address  
     MPU6050, [63](#)  
 setSlave4Enabled  
     MPU6050, [64](#)  
 setSlave4InterruptEnabled  
     MPU6050, [64](#)  
 setSlave4MasterDelay  
     MPU6050, [64](#)  
 setSlave4OutputByte  
     MPU6050, [64](#)  
 setSlave4Register  
     MPU6050, [65](#)  
 setSlave4WriteMode  
     MPU6050, [65](#)  
 setSlaveAddress  
     MPU6050, [65](#)  
 setSlaveDataLength  
     MPU6050, [65](#)  
 setSlaveDelayEnabled  
     MPU6050, [66](#)  
 setSlaveEnabled  
     MPU6050, [66](#)  
 setSlaveOutputByte  
     MPU6050, [66](#)  
 setSlaveReadWriteTransitionEnabled  
     MPU6050, [66](#)  
 setSlaveRegister  
     MPU6050, [67](#)  
 setSlaveWordByteSwap  
     MPU6050, [67](#)  
 setSlaveWordGroupOffset  
     MPU6050, [67](#)  
 setSlaveWriteMode  
     MPU6050, [67](#)  
 setSleepEnabled  
     MPU6050, [68](#)  
 setStandbyXAccelEnabled  
     MPU6050, [68](#)  
 setStandbyXGyroEnabled  
     MPU6050, [68](#)  
 setStandbyYAccelEnabled  
     MPU6050, [68](#)  
 setStandbyYGyroEnabled  
     MPU6050, [68](#)  
 setStandbyZAccelEnabled  
     MPU6050, [68](#)  
 setStandbyZGyroEnabled  
     MPU6050, [68](#)  
 setTempFIFOEnabled  
     MPU6050, [68](#)  
 setTempSensorEnabled  
     MPU6050, [68](#)  
 setWaitForExternalSensorEnabled  
     MPU6050, [69](#)  
 setWakeCycleEnabled  
     MPU6050, [69](#)  
 setWakeFrequency  
     MPU6050, [69](#)  
 setXAccelOffset  
     MPU6050, [69](#)  
 setXFineGain

- MPU6050, [69](#)
- setXGyroFIFOEnabled
  - MPU6050, [69](#)
- setXGyroOffset
  - MPU6050, [69](#)
- setXGyroOffsetUser
  - MPU6050, [70](#)
- setYAccelOffset
  - MPU6050, [70](#)
- setYFineGain
  - MPU6050, [70](#)
- setYGyroFIFOEnabled
  - MPU6050, [70](#)
- setYGyroOffset
  - MPU6050, [70](#)
- setYGyroOffsetUser
  - MPU6050, [70](#)
- setZAccelOffset
  - MPU6050, [70](#)
- setZFineGain
  - MPU6050, [70](#)
- setZGyroFIFOEnabled
  - MPU6050, [70](#)
- setZGyroOffset
  - MPU6050, [71](#)
- setZGyroOffsetUser
  - MPU6050, [71](#)
- setZeroMotionDetectionDuration
  - MPU6050, [70](#)
- setZeroMotionDetectionThreshold
  - MPU6050, [70](#)
- switchSPIEnabled
  - MPU6050, [71](#)
- term
  - SerialCommand, [79](#)
- testConnection
  - MPU6050, [71](#)
- VectorFloat, [80](#)
  - getMagnitude, [80](#)
  - getNormalized, [80](#)
  - getRotated, [80](#)
  - normalize, [80](#)
  - rotate, [80](#)
  - VectorFloat, [80](#)
  - x, [81](#)
  - y, [81](#)
  - z, [81](#)
- VectorInt16, [81](#)
  - getMagnitude, [82](#)
  - getNormalized, [82](#)
  - getRotated, [82](#)
  - normalize, [82](#)
  - rotate, [82](#)
  - VectorInt16, [81](#)
  - x, [82](#)
  - y, [82](#)
  - z, [82](#)
- w
  - Quaternion, [76](#)
- writeBit
  - I2Cdev, [5](#)
- writeBitW
  - I2Cdev, [6](#)
- writeBits
  - I2Cdev, [6](#)
- writeBitsW
  - I2Cdev, [6](#)
- writeByte
  - I2Cdev, [7](#)
- writeBytes
  - I2Cdev, [7](#)
- writeDMPCConfigurationSet
  - MPU6050, [71](#)
- writeMemoryBlock
  - MPU6050, [71](#)
- writeMemoryByte
  - MPU6050, [71](#)
- writeProgDMPCConfigurationSet
  - MPU6050, [71](#)
- writeProgMemoryBlock
  - MPU6050, [71](#)
- writeWord
  - I2Cdev, [7](#)
- writeWords
  - I2Cdev, [7](#)
- x
  - Quaternion, [76](#)
  - VectorFloat, [81](#)
  - VectorInt16, [82](#)
- y
  - Quaternion, [76](#)
  - VectorFloat, [81](#)
  - VectorInt16, [82](#)
- z
  - Quaternion, [76](#)
  - VectorFloat, [81](#)
  - VectorInt16, [82](#)