Arduino Gyroscope Driver

Generated by Doxygen 1.8.11

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- **GyroscopeMPU9250::GYRO_CONFIGbits** 2
- **Gyroscope** 3
  - **GyroscopeMPU9250** 5
- **GyroscopeMPU9250::PWR_MGMT_1bits** 9
- **GyroscopeMPU9250::PWR_MGMT_2bits** 10
- RegisterBasedWiredDevice
  - **GyroscopeMPU9250** 5

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Class Documentation

4.1 GyroscopeMPU9250::GYRO_CONFIGbits Union Reference

```
#include <GyroscopeMPU9250.h>
```

Public Attributes

- struct {
  unsigned char FCHOICE_B:2
  unsigned char:1
  unsigned char GYRO_FS_SEL:2
  unsigned char ZGYRO_CTEN:1
  unsigned char YGYRO_CTEN:1
  unsigned char XGYRO_Cten:1
  };
- unsigned char value

4.1.1 Detailed Description

Gyroscope Configuration (GYRO_CONFIG 0x1b) Serial IF: R/W Reset value: 0x00.

Definition at line 99 of file GyroscopeMPU9250.h.

4.1.2 Member Data Documentation

4.1.2.1 struct { ... }

4.1.2.2 unsigned GyroscopeMPU9250::GYRO_CONFIGbits::char

Definition at line 102 of file GyroscopeMPU9250.h.

4.1.2.3 unsigned char GyroscopeMPU9250::GYRO_CONFIGbits::FCHOICE_B

Definition at line 101 of file GyroscopeMPU9250.h.

4.1.2.4 unsigned char GyroscopeMPU9250::GYRO_CONFIGbits::GYRO_FS_SEL

Definition at line 103 of file GyroscopeMPU9250.h.

4.1.2.5 unsigned char GyroscopeMPU9250::GYRO_CONFIGbits::value

Definition at line 108 of file GyroscopeMPU9250.h.

4.1.2.6 unsigned char GyroscopeMPU9250::GYRO_CONFIGbits::XGYRO_Cten

Definition at line 106 of file GyroscopeMPU9250.h.

4.1.2.7 unsigned char GyroscopeMPU9250::GYRO_CONFIGbits::YGYRO_CTEN

Definition at line 105 of file GyroscopeMPU9250.h.

4.1.2.8 unsigned char GyroscopeMPU9250::GYRO_CONFIGbits::ZGYRO_CTEN

Definition at line 104 of file GyroscopeMPU9250.h.

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

- GyroscopeMPU9250.h

4.2 Gyroscope Class Reference

```
#include <Gyroscope.h>
```

Inheritance diagram for Gyroscope:

Public Member Functions

- virtual ∼Gyroscope ()
- virtual float getRotationX ()=0
- virtual float getRotationY ()=0
- virtual float getRotationZ ()=0

4.2.1 Detailed Description

Arduino - Gyroscope Driver.

Abstract interface which all gyroscope should implement.

Author

Dalmir da Silva `dalmirdasilva@gmail.com`

Definition at line 12 of file Gyroscope.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Gyroscope::∼Gyroscope ( ) [virtual]

Definition at line 3 of file Gyroscope.cpp.

4.2.3 Member Function Documentation

4.2.3.1 virtual float Gyroscope::getRotationX ( ) [pure virtual]

Implemented in GyroscopeMPU9250.

4.2.3.2 virtual float Gyroscope::getRotationY ( ) [pure virtual]

Implemented in GyroscopeMPU9250.

4.2.3.3 virtual float Gyroscope::getRotationZ ( ) [pure virtual]

Implemented in GyroscopeMPU9250.

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

- Gyroscope.h
- Gyroscope.cpp

4.3 GyroscopeMPU9250 Class Reference

```
#include <GyroscopeMPU9250.h>
```

Inheritance diagram for GyroscopeMPU9250:

Collaboration diagram for GyroscopeMPU9250:

Classes

- union GYRO_CONFIGbits
- union PWR_MGMT_1bits
- union PWR_MGMT_2bits

Public Types

- enum Register {
PWR_MGMT_1 = 0x6b, PWR_MGMT_2 = 0x6c, GYRO_CONFIG = 0x1b, GYRO_XOUT_H = 0x43,
GYRO_XOUT_L = 0x44, GYRO_YOUT_H = 0x45, GYRO_YOUT_L = 0x46, GYRO_ZOUT_H = 0x47,
GYRO_ZOUT_L = 0x48 }
- enum FullScaleRange { FS_SEL_250DPS = 0x00, FS_SEL_500DPS = 0x08, FS_SEL_1000DPS = 0x10, FS_SEL_2000DPS = 0x18 }
- enum ClockSelection { INTERNAL_20MHZ_OSCILLATOR = 0x00, BEST_AVAILABLE_SOURCE = 0x01, STOPS_CLOCK_KEEPS_TIMING = 0x07 }
- enum Axis {
AXIS_NONE = 0x00, AXIS_X = 0x04, AXIS_Y = 0x02, AXIS_Z = 0x01,
AXIS_XY = AXIS_X | AXIS_Y, AXIS_XZ = AXIS_X | AXIS_Z, AXIS_YZ = AXIS_Y | AXIS_Z, AXIS_XYZ = AXIS_X | AXIS_Y | AXIS_Z }
- enum Mask {
GYRO_CONFIG_GYRO_FS_SEL = 0x18, PWR_MGMT_2_DISABLE_G = 0x07, PWR_MGMT_1_H_RESET = 0x80, PWR_MGMT_1_SLEEP = 0x40,
PWR_MGMT_1_CYCLE = 0x20, PWR_MGMT_1_GYRO_STANDBY = 0x10, PWR_MGMT_1_CLKSEL = 0x07 }

Public Member Functions

- GyroscopeMPU9250 (bool ad0)
- float getRotationX ()
- float getRotationY ()
- float getRotationZ ()
- unsigned char readXYZ (unsigned char ∗buf)
- float readAxisRotation (unsigned char axisRegister)
- void setFullScaleRange (FullScaleRange fsr)
- void selectClock (ClockSelection cs)
- void reset ()
- void sleep ()
- void awake ()
- void enableAxis (Axis axis)
- float convertToDegreePerSeconds (unsigned char buf[2])

Private Attributes

- GYRO_CONFIGbits config

4.3.1 Detailed Description

The MPU-9250 consists of three independent vibratory MEMS rate gyroscopes, which detect rotation about the X-, Y-, and Z- Axes.

When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to t he angular rat e. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis. The full-scale range of the gyro sensors may be digitally programmed to ±250, ±500, ±1000, or ±2000 degrees per second (dps). The ADC sample rate is programmable from 8,000 s amples per second, down to 3.9 samples per second, and us er-s electable low-pass filters enable a wide range of cut-off frequencies.

Definition at line 27 of file GyroscopeMPU9250.h.

4.3.2 Member Enumeration Documentation

4.3.2.1 enum GyroscopeMPU9250::Axis

Axis.

Enumerator

- ***AXIS_NONE***
- ***AXIS_X***
- ***AXIS_Y***
- ***AXIS_Z***
- ***AXIS_XY***
- ***AXIS_XZ***
- ***AXIS_YZ***
- ***AXIS_XYZ***

Definition at line 70 of file GyroscopeMPU9250.h.

4.3.2.2 enum GyroscopeMPU9250::ClockSelection

Enumerator

- ***INTERNAL_20MHZ_OSCILLATOR***
- ***BEST_AVAILABLE_SOURCE***
- ***STOPS_CLOCK_KEEPS_TIMING***

Definition at line 61 of file GyroscopeMPU9250.h.

4.3.2.3 enum GyroscopeMPU9250::FullScaleRange

Gyroscope Full Scale Select.

```
FS1 FS0  g Range
0    0   +250dps
0    1   +500dps
1    0   +1000dps
1    1   +2000dps
```

Enumerator

FS_SEL_250DPS

FS_SEL_500DPS

FS_SEL_1000DPS

FS_SEL_2000DPS

Definition at line 54 of file GyroscopeMPU9250.h.

4.3.2.4 enum GyroscopeMPU9250::Mask

Some useful masks.

Enumerator

GYRO_CONFIG_GYRO_FS_SEL

PWR_MGMT_2_DISABLE_G

PWR_MGMT_1_H_RESET

PWR_MGMT_1_SLEEP

PWR_MGMT_1_CYCLE

PWR_MGMT_1_GYRO_STANDBY

PWR_MGMT_1_CLKSEL

Definition at line 84 of file GyroscopeMPU9250.h.

4.3.2.5 enum GyroscopeMPU9250::Register

Enumerator

PWR_MGMT_1

PWR_MGMT_2

GYRO_CONFIG

GYRO_XOUT_H

GYRO_XOUT_L

GYRO_YOUT_H

GYRO_YOUT_L

GYRO_ZOUT_H

GYRO_ZOUT_L

Definition at line 31 of file GyroscopeMPU9250.h.

4.3.3 Constructor & Destructor Documentation

4.3.3.1 GyroscopeMPU9250::GyroscopeMPU9250 (bool *ad0*)

Public constructor.

Parameters

ad0	LSBit of the device address.

Definition at line 3 of file GyroscopeMPU9250.cpp.

4.3.4 Member Function Documentation

4.3.4.1 void GyroscopeMPU9250::awake ( )

Definition at line 49 of file GyroscopeMPU9250.cpp.

4.3.4.2 float GyroscopeMPU9250::convertToDegreePerSeconds (unsigned char *buf[2]*)

Definition at line 57 of file GyroscopeMPU9250.cpp.

4.3.4.3 void GyroscopeMPU9250::enableAxis (Axis *axis*)

Definition at line 53 of file GyroscopeMPU9250.cpp.

4.3.4.4 float GyroscopeMPU9250::getRotationX ( ) [virtual]

Implements Gyroscope.

Definition at line 8 of file GyroscopeMPU9250.cpp.

4.3.4.5 float GyroscopeMPU9250::getRotationY ( ) [virtual]

Implements Gyroscope.

Definition at line 12 of file GyroscopeMPU9250.cpp.

4.3.4.6 float GyroscopeMPU9250::getRotationZ ( ) [virtual]

Implements Gyroscope.

Definition at line 16 of file GyroscopeMPU9250.cpp.

4.3.4.7 float GyroscopeMPU9250::readAxisRotation (unsigned char *axisRegister*)

Definition at line 24 of file GyroscopeMPU9250.cpp.

4.3.4.8 unsigned char GyroscopeMPU9250::readXYZ (unsigned char ∗ *buf*)

Definition at line 20 of file GyroscopeMPU9250.cpp.

4.3.4.9 void GyroscopeMPU9250::reset ( )

Definition at line 41 of file GyroscopeMPU9250.cpp.

4.3.4.10 void GyroscopeMPU9250::selectClock (ClockSelection *cs*)

Definition at line 37 of file GyroscopeMPU9250.cpp.

4.3.4.11 void GyroscopeMPU9250::setFullScaleRange (FullScaleRange *fsr*)

Definition at line 32 of file GyroscopeMPU9250.cpp.

4.3.4.12 void GyroscopeMPU9250::sleep ( )

Definition at line 45 of file GyroscopeMPU9250.cpp.

4.3.5 Member Data Documentation

4.3.5.1 GYRO_CONFIGbits GyroscopeMPU9250::config [private]

Definition at line 184 of file GyroscopeMPU9250.h.

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

- GyroscopeMPU9250.h
- GyroscopeMPU9250.cpp

4.4 GyroscopeMPU9250::PWR_MGMT_1bits Union Reference

```
#include <GyroscopeMPU9250.h>
```

Public Attributes

- struct {
  unsigned char CLKSEL:3
  unsigned char PD_PTAT:1
  unsigned char GYRO_STANDBY:1
  unsigned char CYCLE:1
  unsigned char SLEEP:1
  unsigned char H_RESET:1
  };
- unsigned char value

4.4.1 Detailed Description

Power Management 1 Serial IF: R/W Reset value: 0x00.

Definition at line 116 of file GyroscopeMPU9250.h.

4.4.2 Member Data Documentation

4.4.2.1 struct { ... }

4.4.2.2 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::CLKSEL

Definition at line 118 of file GyroscopeMPU9250.h.

4.4.2.3 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::CYCLE

Definition at line 121 of file GyroscopeMPU9250.h.

4.4.2.4 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::GYRO_STANDBY

Definition at line 120 of file GyroscopeMPU9250.h.

4.4.2.5 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::H_RESET

Definition at line 123 of file GyroscopeMPU9250.h.

4.4.2.6 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::PD_PTAT

Definition at line 119 of file GyroscopeMPU9250.h.

4.4.2.7 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::SLEEP

Definition at line 122 of file GyroscopeMPU9250.h.

4.4.2.8 unsigned char GyroscopeMPU9250::PWR_MGMT_1bits::value

Definition at line 125 of file GyroscopeMPU9250.h.

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

- GyroscopeMPU9250.h

4.5 GyroscopeMPU9250::PWR_MGMT_2bits Union Reference

```
#include <GyroscopeMPU9250.h>
```

Public Attributes

- struct {
  unsigned char DISABLE_ZG:1
  unsigned char DISABLE_YG:1
  unsigned char DISABLE_XG:1
  unsigned char DISABLE_ZA:1
  unsigned char DISABLE_YA:1
  unsigned char DISABLE_XA:1
  unsigned char:2
  };
- struct {
  unsigned char DISABLE_G:3
  unsigned char DISABLE_A:3
  unsigned char:2
  };
- unsigned char value

4.5.1 Detailed Description

Power Management 2 Serial IF: R/W Reset value: 0x00.

Definition at line 133 of file GyroscopeMPU9250.h.

4.5.2 Member Data Documentation

4.5.2.1 struct { ... }

4.5.2.2 struct { ... }

4.5.2.3 unsigned GyroscopeMPU9250::PWR_MGMT_2bits::char

Definition at line 141 of file GyroscopeMPU9250.h.

4.5.2.4 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_A

Definition at line 145 of file GyroscopeMPU9250.h.

4.5.2.5 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_G

Definition at line 144 of file GyroscopeMPU9250.h.

4.5.2.6 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_XA

Definition at line 140 of file GyroscopeMPU9250.h.

4.5.2.7 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_XG

Definition at line 137 of file GyroscopeMPU9250.h.

4.5.2.8 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_YA

Definition at line 139 of file GyroscopeMPU9250.h.

4.5.2.9 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_YG

Definition at line 136 of file GyroscopeMPU9250.h.

4.5.2.10 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_ZA

Definition at line 138 of file GyroscopeMPU9250.h.

4.5.2.11 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::DISABLE_ZG

Definition at line 135 of file GyroscopeMPU9250.h.

4.5.2.12 unsigned char GyroscopeMPU9250::PWR_MGMT_2bits::value

Definition at line 148 of file GyroscopeMPU9250.h.

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

- GyroscopeMPU9250.h

5 File Documentation

5.1 Gyroscope.cpp File Reference

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

5.2 Gyroscope.cpp

```
00001 #include "Gyroscope.h"
00002
00003 Gyroscope::~Gyroscope() {
00004 }
```

5.3 Gyroscope.h File Reference

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

Classes

- class Gyroscope

5.4 Gyroscope.h

```

#ifndef __ARDUINO_DRIVER_GYROSCOPE_H__
#define __ARDUINO_DRIVER_GYROSCOPE_H__ 1

class Gyroscope {

public:

    virtual ~Gyroscope();

    virtual float getRotationX() = 0;

    virtual float getRotationY() = 0;

    virtual float getRotationZ() = 0;
};

#endif /* __ARDUINO_DRIVER_GYROSCOPE_H__ */
```

5.5 GyroscopeMPU6050.cpp File Reference

5.6 GyroscopeMPU6050.cpp

```
//
//
//I2Cdev device library code is placed under the MIT license
//Copyright (c) 2012 Jeff Rowberg
//
//Permission is hereby granted, free of charge, to any person obtaining a copy
//of this software and associated documentation files (the "Software"), to deal
//in the Software without restriction, including without limitation the rights
//to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
//copies of the Software, and to permit persons to whom the Software is
//furnished to do so, subject to the following conditions:
//
//The above copyright notice and this permission notice shall be included in
//all copies or substantial portions of the Software.
//
//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
//IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
//AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
//OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
//THE SOFTWARE.
//===============================================
//*/
//
//#include "MPU6050.h"
//
// * @see MPU6050_DEFAULT_ADDRESS
// */
//MPU6050::MPU6050() {
//    devAddr = MPU6050_DEFAULT_ADDRESS;
//}
//
// * @param address I2C address
// * @see MPU6050_DEFAULT_ADDRESS
// * @see MPU6050_ADDRESS_AD0_LOW
// * @see MPU6050_ADDRESS_AD0_HIGH
// */
//MPU6050::MPU6050(uint8_t address) {
//    devAddr = address;
//}
//
// * 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.
// */
//void MPU6050::initialize() {
//    setClockSource(MPU6050_CLOCK_PLL_XGYRO);
//    setFullScaleGyroRange(MPU6050_GYRO_FS_250);
//    setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
//    setSleepEnabled(false); // thanks to Jack Elston for pointing this one out!
//}
```

```
//
// * Make sure the device is connected and responds as expected.
// * @return True if connection is valid, false otherwise
// */
//bool MPU6050::testConnection() {
//    return getDeviceID() == 0x34;
//}
//
//
// * 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.
// * @return I2C supply voltage level (0=VLOGIC, 1=VDD)
// */
//uint8_t MPU6050::getAuxVDDIOLevel() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_YG_OFFS_TC, MPU6050_TC_PWR_MODE_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @param level I2C supply voltage level (0=VLOGIC, 1=VDD)
// */
//void MPU6050::setAuxVDDIOLevel(uint8_t level) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_YG_OFFS_TC, MPU6050_TC_PWR_MODE_BIT, level);
//}
//
//
// * The sensor register output, FIFO output, DMP sampling, Motion detection, Zero
// * Motion detection, and Free Fall detection are all based on the Sample Rate.
// * The Sample Rate is generated by dividing the gyroscope output rate by
// * SMPLRT_DIV:
// *
// * Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)
// *
// * where Gyroscope Output Rate = 8kHz when the DLPF is disabled (DLPF_CFG = 0 or
// * 7), and 1kHz when the DLPF is enabled (see Register 26).
// *
// * Note: The accelerometer output rate is 1kHz. This means that for a Sample
// * Rate greater than 1kHz, the same accelerometer sample may be output to the
// * FIFO, DMP, and sensor registers more than once.
// *
// * For a diagram of the gyroscope and accelerometer signal paths, see Section 8
// * of the MPU-6000/MPU-6050 Product Specification document.
// *
// * @return Current sample rate
// * @see MPU6050_RA_SMPLRT_DIV
// */
//uint8_t MPU6050::getRate() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_SMPLRT_DIV, buffer);
//    return buffer[0];
//}
// * @param rate New sample rate divider
// * @see getRate()
// * @see MPU6050_RA_SMPLRT_DIV
// */
//void MPU6050::setRate(uint8_t rate) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_SMPLRT_DIV, rate);
//}
//
//
// * 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.
// *
// * <pre>
// * 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]
// * </pre>
// *
// * @return FSYNC configuration value
// */
```

```
//uint8_t MPU6050::getExternalFrameSync() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG, MPU6050_CFG_EXT_SYNC_SET_BIT, MPU6050_CFG_EXT_SYNC_SET_LENGTH, buffer);
//    return buffer[0];
//}
// * @see getExternalFrameSync()
// * @see MPU6050_RA_CONFIG
// * @param sync New FSYNC configuration value
// */
//void MPU6050::setExternalFrameSync(uint8_t sync) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG, MPU6050_CFG_EXT_SYNC_SET_BIT, MPU6050_CFG_EXT_SYNC_SET_LENGTH, sync);
//}
// * 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.
// *
// * <pre>
// *          |   ACCELEROMETER    |           GYROSCOPE
// * DLPF_CFG | Bandwidth | Delay  | Bandwidth | Delay  | Sample Rate
// * ---------+-----------+--------+-----------+--------+-------------
// * 0        | 260Hz     | 0ms    | 256Hz     | 0.98ms | 8kHz
// * 1        | 184Hz     | 2.0ms  | 188Hz     | 1.9ms  | 1kHz
// * 2        | 94Hz      | 3.0ms  | 98Hz      | 2.8ms  | 1kHz
// * 3        | 44Hz      | 4.9ms  | 42Hz      | 4.8ms  | 1kHz
// * 4        | 21Hz      | 8.5ms  | 20Hz      | 8.3ms  | 1kHz
// * 5        | 10Hz      | 13.8ms | 10Hz      | 13.4ms | 1kHz
// * 6        | 5Hz       | 19.0ms | 5Hz       | 18.6ms | 1kHz
// * 7        |   -- Reserved --   |   -- Reserved --   | Reserved
// * </pre>
// *
// * @return DLFP configuration
// * @see MPU6050_RA_CONFIG
// * @see MPU6050_CFG_DLPF_CFG_BIT
// * @see MPU6050_CFG_DLPF_CFG_LENGTH
// */
//uint8_t MPU6050::getDLPFMode() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG, MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH, buffer);
//    return buffer[0];
//}
// * @param mode New DLFP configuration setting
// * @see getDLPFBandwidth()
// * @see MPU6050_DLPF_BW_256
// * @see MPU6050_RA_CONFIG
// * @see MPU6050_CFG_DLPF_CFG_BIT
// * @see MPU6050_CFG_DLPF_CFG_LENGTH
// */
//void MPU6050::setDLPFMode(uint8_t mode) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG, MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH, mode);
//}
//
//
// * The FS_SEL parameter allows setting the full-scale range of the gyro sensors,
// * as described in the table below.
// *
// * <pre>
// * 0 = +/- 250 degrees/sec
// * 1 = +/- 500 degrees/sec
// * 2 = +/- 1000 degrees/sec
// * 3 = +/- 2000 degrees/sec
// * </pre>
// *
// * @return Current full-scale gyroscope range setting
// * @see MPU6050_GYRO_FS_250
// * @see MPU6050_RA_GYRO_CONFIG
// * @see MPU6050_GCONFIG_FS_SEL_BIT
// * @see MPU6050_GCONFIG_FS_SEL_LENGTH
// */
//uint8_t MPU6050::getFullScaleGyroRange() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_GYRO_CONFIG, MPU6050_GCONFIG_FS_SEL_BIT, MPU6050_GCONFIG_FS_SEL_LENGTH, buffer);
//    return buffer[0];
//}
// * @param range New full-scale gyroscope range value
// * @see getFullScaleRange()
// * @see MPU6050_GYRO_FS_250
// * @see MPU6050_RA_GYRO_CONFIG
// * @see MPU6050_GCONFIG_FS_SEL_BIT
// * @see MPU6050_GCONFIG_FS_SEL_LENGTH
// */
//void MPU6050::setFullScaleGyroRange(uint8_t range) {
```

```
//    I2Cdev::writeBits(devAddr, MPU6050_RA_GYRO_CONFIG, MPU6050_GCONFIG_FS_SEL_BIT, MPU6050_GCONFIG_FS_SEL_LENGTH, range);
//}
//
//
// * @return Self-test enabled value
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//bool MPU6050::getAccelXSelfTest() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_XA_ST_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled Self-test enabled value
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//void MPU6050::setAccelXSelfTest(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_XA_ST_BIT, enabled);
//}
// * @return Self-test enabled value
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//bool MPU6050::getAccelYSelfTest() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_YA_ST_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled Self-test enabled value
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//void MPU6050::setAccelYSelfTest(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_YA_ST_BIT, enabled);
//}
// * @return Self-test enabled value
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//bool MPU6050::getAccelZSelfTest() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_ZA_ST_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled Self-test enabled value
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//void MPU6050::setAccelZSelfTest(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_ZA_ST_BIT, enabled);
//}
// * The FS_SEL parameter allows setting the full-scale range of the accelerometer
// * sensors, as described in the table below.
// *
// * <pre>
// * 0 = +/- 2g
// * 1 = +/- 4g
// * 2 = +/- 8g
// * 3 = +/- 16g
// * </pre>
// *
// * @return Current full-scale accelerometer range setting
// * @see MPU6050_ACCEL_FS_2
// * @see MPU6050_RA_ACCEL_CONFIG
// * @see MPU6050_ACONFIG_AFS_SEL_BIT
// * @see MPU6050_ACONFIG_AFS_SEL_LENGTH
// */
//uint8_t MPU6050::getFullScaleAccelRange() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_AFS_SEL_BIT, MPU6050_ACONFIG_AFS_SEL_LENGTH, buffer);
//    return buffer[0];
//}
// * @param range New full-scale accelerometer range setting
// * @see getFullScaleAccelRange()
// */
//void MPU6050::setFullScaleAccelRange(uint8_t range) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_AFS_SEL_BIT, MPU6050_ACONFIG_AFS_SEL_LENGTH, range);
//}
// * 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:
// *
// * <pre>
// *    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.
// * </pre>
// *
// * <pre>
// * 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
// * </pre>
// *
// * @return Current high-pass filter configuration
// * @see MPU6050_DHPF_RESET
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//uint8_t MPU6050::getDHPFMode() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_ACCEL_HPF_BIT,
 MPU6050_ACONFIG_ACCEL_HPF_LENGTH, buffer);
//    return buffer[0];
//}
// * @param bandwidth New high-pass filter configuration
// * @see setDHPFMode()
// * @see MPU6050_DHPF_RESET
// * @see MPU6050_RA_ACCEL_CONFIG
// */
//void MPU6050::setDHPFMode(uint8_t bandwidth) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_ACCEL_HPF_BIT,
 MPU6050_ACONFIG_ACCEL_HPF_LENGTH, bandwidth);
//}
//
//
// * 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.
// *
// * @return Current free-fall acceleration threshold value (LSB = 2mg)
// * @see MPU6050_RA_FF_THR
// */
//uint8_t MPU6050::getFreefallDetectionThreshold() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_FF_THR, buffer);
//    return buffer[0];
//}
// * @param threshold New free-fall acceleration threshold value (LSB = 2mg)
// * @see getFreefallDetectionThreshold()
// * @see MPU6050_RA_FF_THR
// */
//void MPU6050::setFreefallDetectionThreshold(uint8_t threshold) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_FF_THR, 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.
// *
// * @return Current free-fall duration threshold value (LSB = 1ms)
// * @see MPU6050_RA_FF_DUR
// */
//uint8_t MPU6050::getFreefallDetectionDuration() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_FF_DUR, buffer);
//    return buffer[0];
//}
// * @param duration New free-fall duration threshold value (LSB = 1ms)
// * @see getFreefallDetectionDuration()
// * @see MPU6050_RA_FF_DUR
// */
//void MPU6050::setFreefallDetectionDuration(uint8_t duration) {
```

```
//    I2Cdev::writeByte(devAddr, MPU6050_RA_FF_DUR, duration);
//}
//
//
// * 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.
// *
// * @return Current motion detection acceleration threshold value (LSB = 2mg)
// * @see MPU6050_RA_MOT_THR
// */
//uint8_t MPU6050::getMotionDetectionThreshold() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_MOT_THR, buffer);
//    return buffer[0];
//}
// * @param threshold New motion detection acceleration threshold value (LSB = 2mg)
// * @see getMotionDetectionThreshold()
// * @see MPU6050_RA_MOT_THR
// */
//void MPU6050::setMotionDetectionThreshold(uint8_t threshold) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_MOT_THR, 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.
// *
// * @return Current motion detection duration threshold value (LSB = 1ms)
// * @see MPU6050_RA_MOT_DUR
// */
//uint8_t MPU6050::getMotionDetectionDuration() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_MOT_DUR, buffer);
//    return buffer[0];
//}
// * @param duration New motion detection duration threshold value (LSB = 1ms)
// * @see getMotionDetectionDuration()
// * @see MPU6050_RA_MOT_DUR
// */
//void MPU6050::setMotionDetectionDuration(uint8_t duration) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_MOT_DUR, duration);
//}
//
//
// * 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.
// *
// * @return Current zero motion detection acceleration threshold value (LSB = 2mg)
// * @see MPU6050_RA_ZRMOT_THR
// */
//uint8_t MPU6050::getZeroMotionDetectionThreshold() {
```

```
//    I2Cdev::readByte(devAddr, MPU6050_RA_ZRMOT_THR, buffer);
//    return buffer[0];
//}
// * @param threshold New zero motion detection acceleration threshold value (LSB = 2mg)
// * @see getZeroMotionDetectionThreshold()
// * @see MPU6050_RA_ZRMOT_THR
// */
//void MPU6050::setZeroMotionDetectionThreshold(uint8_t threshold) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_ZRMOT_THR, 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.
// *
// * @return Current zero motion detection duration threshold value (LSB = 64ms)
// * @see MPU6050_RA_ZRMOT_DUR
// */
//uint8_t MPU6050::getZeroMotionDetectionDuration() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_ZRMOT_DUR, buffer);
//    return buffer[0];
//}
// * @param duration New zero motion detection duration threshold value (LSB = 1ms)
// * @see getZeroMotionDetectionDuration()
// * @see MPU6050_RA_ZRMOT_DUR
// */
//void MPU6050::setZeroMotionDetectionDuration(uint8_t duration) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_ZRMOT_DUR, duration);
//}
//
//
// * 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.
// * @return Current temperature FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getTempFIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_TEMP_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New temperature FIFO enabled value
// * @see getTempFIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setTempFIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_TEMP_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current gyroscope X-axis FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getXGyroFIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_XG_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New gyroscope X-axis FIFO enabled value
// * @see getXGyroFIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setXGyroFIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_XG_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current gyroscope Y-axis FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getYGyroFIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_YG_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New gyroscope Y-axis FIFO enabled value
// * @see getYGyroFIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setYGyroFIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_YG_FIFO_EN_BIT, enabled);
```

```
//}
// * 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.
// * @return Current gyroscope Z-axis FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getZGyroFIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_ZG_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New gyroscope Z-axis FIFO enabled value
// * @see getZGyroFIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setZGyroFIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_ZG_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current accelerometer FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getAccelFIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_ACCEL_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New accelerometer FIFO enabled value
// * @see getAccelFIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setAccelFIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_ACCEL_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current Slave 2 FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getSlave2FIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV2_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New Slave 2 FIFO enabled value
// * @see getSlave2FIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setSlave2FIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV2_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current Slave 1 FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getSlave1FIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV1_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New Slave 1 FIFO enabled value
// * @see getSlave1FIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setSlave1FIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV1_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current Slave 0 FIFO enabled value
// * @see MPU6050_RA_FIFO_EN
// */
//bool MPU6050::getSlave0FIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV0_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New Slave 0 FIFO enabled value
// * @see getSlave0FIFOEnabled()
// * @see MPU6050_RA_FIFO_EN
// */
//void MPU6050::setSlave0FIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV0_FIFO_EN_BIT, enabled);
//}
//
//
// * 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.
// *
// * @return Current multi-master enabled value
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//bool MPU6050::getMultiMasterEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_MULT_MST_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New multi-master enabled value
// * @see getMultiMasterEnabled()
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//void MPU6050::setMultiMasterEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_MULT_MST_EN_BIT, enabled);
//}
// * 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.
// *
// * @return Current wait-for-external-sensor-data enabled value
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//bool MPU6050::getWaitForExternalSensorEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_WAIT_FOR_ES_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New wait-for-external-sensor-data enabled value
// * @see getWaitForExternalSensorEnabled()
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//void MPU6050::setWaitForExternalSensorEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_WAIT_FOR_ES_BIT, enabled);
//}
// * 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.
// * @return Current Slave 3 FIFO enabled value
// * @see MPU6050_RA_MST_CTRL
// */
//bool MPU6050::getSlave3FIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_SLV_3_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New Slave 3 FIFO enabled value
// * @see getSlave3FIFOEnabled()
// * @see MPU6050_RA_MST_CTRL
// */
//void MPU6050::setSlave3FIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_SLV_3_FIFO_EN_BIT, enabled);
//}
// * 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.
// *
// * @return Current slave read/write transition enabled value
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//bool MPU6050::getSlaveReadWriteTransitionEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_I2C_MST_P_NSR_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New slave read/write transition enabled value
// * @see getSlaveReadWriteTransitionEnabled()
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//void MPU6050::setSlaveReadWriteTransitionEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_I2C_MST_P_NSR_BIT, enabled);
//}
// * 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:
// *
// * <pre>
// * 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
// * </pre>
// *
// * @return Current I2C master clock speed
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//uint8_t MPU6050::getMasterClockSpeed() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_I2C_MST_CLK_BIT, MPU6050_I2C_MST_CLK_LENGTH, buffer);
//    return buffer[0];
//}
// * @reparam speed Current I2C master clock speed
// * @see MPU6050_RA_I2C_MST_CTRL
// */
//void MPU6050::setMasterClockSpeed(uint8_t speed) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_MST_CTRL, MPU6050_I2C_MST_CLK_BIT, MPU6050_I2C_MST_CLK_LENGTH, speed);
//}
//
//
// * 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.
// *
// * @param num Slave number (0-3)
// * @return Current address for specified slave
// * @see MPU6050_RA_I2C_SLV0_ADDR
// */
//uint8_t MPU6050::getSlaveAddress(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV0_ADDR + num*3, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param address New address for specified slave
// * @see getSlaveAddress()
// * @see MPU6050_RA_I2C_SLV0_ADDR
// */
//void MPU6050::setSlaveAddress(uint8_t num, uint8_t address) {
//    if (num > 3) return;
//    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_ADDR + num*3, address);
//}
// * 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.
// *
// * @param num Slave number (0-3)
// * @return Current active register for specified slave
// * @see MPU6050_RA_I2C_SLV0_REG
// */
//uint8_t MPU6050::getSlaveRegister(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV0_REG + num*3, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param reg New active register for specified slave
// * @see getSlaveRegister()
// * @see MPU6050_RA_I2C_SLV0_REG
// */
//void MPU6050::setSlaveRegister(uint8_t num, uint8_t reg) {
//    if (num > 3) return;
//    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_REG + num*3, reg);
//}
// * 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.
// * @param num Slave number (0-3)
// * @return Current enabled value for specified slave
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//bool MPU6050::getSlaveEnabled(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param enabled New enabled value for specified slave
// * @see getSlaveEnabled()
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//void MPU6050::setSlaveEnabled(uint8_t num, bool enabled) {
//    if (num > 3) return;
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_EN_BIT, enabled);
//}
// * 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.
// *
// * @param num Slave number (0-3)
// * @return Current word pair byte-swapping enabled value for specified slave
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//bool MPU6050::getSlaveWordByteSwap(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_BYTE_SW_BIT, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param enabled New word pair byte-swapping enabled value for specified slave
// * @see getSlaveWordByteSwap()
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//void MPU6050::setSlaveWordByteSwap(uint8_t num, bool enabled) {
//    if (num > 3) return;
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_BYTE_SW_BIT, enabled);
//}
// * 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.
// *
// * @param num Slave number (0-3)
// * @return Current write mode for specified slave (0 = register address + data, 1 = data only)
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//bool MPU6050::getSlaveWriteMode(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_REG_DIS_BIT, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param mode New write mode for specified slave (0 = register address + data, 1 = data only)
// * @see getSlaveWriteMode()
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//void MPU6050::setSlaveWriteMode(uint8_t num, bool mode) {
//    if (num > 3) return;
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_REG_DIS_BIT, mode);
//}
// * 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.
// *
// * @param num Slave number (0-3)
// * @return Current word pair grouping order offset for specified slave
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//bool MPU6050::getSlaveWordGroupOffset(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_GRP_BIT, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param enabled New word pair grouping order offset for specified slave
// * @see getSlaveWordGroupOffset()
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//void MPU6050::setSlaveWordGroupOffset(uint8_t num, bool enabled) {
//    if (num > 3) return;
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_GRP_BIT, enabled);
//}
// * 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.
// * @param num Slave number (0-3)
// * @return Number of bytes to read for specified slave
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//uint8_t MPU6050::getSlaveDataLength(uint8_t num) {
//    if (num > 3) return 0;
//    I2Cdev::readBits(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_LEN_BIT, MPU6050_I2C_SLV_LEN_LENGTH, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-3)
// * @param length Number of bytes to read for specified slave
// * @see getSlaveDataLength()
// * @see MPU6050_RA_I2C_SLV0_CTRL
// */
//void MPU6050::setSlaveDataLength(uint8_t num, uint8_t length) {
//    if (num > 3) return;
//    I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3, MPU6050_I2C_SLV_LEN_BIT, MPU6050_I2C_SLV_LEN_LENGTH, length);
//}
//
//
// * 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.
// *
// * @return Current address for Slave 4
// * @see getSlaveAddress()
// * @see MPU6050_RA_I2C_SLV4_ADDR
// */
//uint8_t MPU6050::getSlave4Address() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_ADDR, buffer);
//    return buffer[0];
//}
// * @param address New address for Slave 4
// * @see getSlave4Address()
// * @see MPU6050_RA_I2C_SLV4_ADDR
// */
//void MPU6050::setSlave4Address(uint8_t address) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_ADDR, address);
//}
// * Read/write operations for this slave will be done to whatever internal
// * register address is stored in this MPU register.
// *
// * @return Current active register for Slave 4
// * @see MPU6050_RA_I2C_SLV4_REG
// */
//uint8_t MPU6050::getSlave4Register() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_REG, buffer);
//    return buffer[0];
//}
// * @param reg New active register for Slave 4
// * @see getSlave4Register()
// * @see MPU6050_RA_I2C_SLV4_REG
// */
//void MPU6050::setSlave4Register(uint8_t reg) {
```

```
//    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_REG, reg);
//}
// * 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.
// * @param data New byte to write to Slave 4
// * @see MPU6050_RA_I2C_SLV4_DO
// */
//void MPU6050::setSlave4OutputByte(uint8_t data) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_DO, data);
//}
// * 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.
// * @return Current enabled value for Slave 4
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//bool MPU6050::getSlave4Enabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New enabled value for Slave 4
// * @see getSlave4Enabled()
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//void MPU6050::setSlave4Enabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_EN_BIT, enabled);
//}
// * 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.
// *
// * @return Current enabled value for Slave 4 transaction interrupts.
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//bool MPU6050::getSlave4InterruptEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_INT_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New enabled value for Slave 4 transaction interrupts.
// * @see getSlave4InterruptEnabled()
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//void MPU6050::setSlave4InterruptEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_INT_EN_BIT, enabled);
//}
// * 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.
// *
// * @return Current write mode for Slave 4 (0 = register address + data, 1 = data only)
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//bool MPU6050::getSlave4WriteMode() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_REG_DIS_BIT, buffer);
//    return buffer[0];
//}
// * @param mode New write mode for Slave 4 (0 = register address + data, 1 = data only)
// * @see getSlave4WriteMode()
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//void MPU6050::setSlave4WriteMode(bool mode) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_REG_DIS_BIT, mode);
//}
// * 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.
// *
// * @return Current Slave 4 master delay value
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
//uint8_t MPU6050::getSlave4MasterDelay() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_MST_DLY_BIT, MPU6050_I2C_SLV4_MST_DLY_LENGTH, buffer);
//    return buffer[0];
//}
// * @param delay New Slave 4 master delay value
// * @see getSlave4MasterDelay()
// * @see MPU6050_RA_I2C_SLV4_CTRL
// */
```

```
//void MPU6050::setSlave4MasterDelay(uint8_t delay) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_SLV4_CTRL, MPU6050_I2C_SLV4_MST_DLY_BIT, MPU6050_I2C_SLV4_MST_DLY_LENGTH, delay);
//}
// * This register stores the data read from Slave 4. This field is populated
// * after a read transaction.
// * @return Last available byte read from to Slave 4
// * @see MPU6050_RA_I2C_SLV4_DI
// */
//uint8_t MPU6050::getSlate4InputByte() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_DI, buffer);
//    return buffer[0];
//}
//
//
// * 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).
// * @return FSYNC interrupt status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getPassthroughStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_PASS_THROUGH_BIT, buffer);
//    return buffer[0];
//}
// * Automatically sets to 1 when a Slave 4 transaction has completed. This
// * triggers an interrupt if the I2C_MST_INT_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).
// * @return Slave 4 transaction done status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getSlave4IsDone() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_SLV4_DONE_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Master arbitration lost status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getLostArbitration() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_LOST_ARB_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Slave 4 NACK interrupt status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getSlave4Nack() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_SLV4_NACK_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Slave 3 NACK interrupt status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getSlave3Nack() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_SLV3_NACK_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Slave 2 NACK interrupt status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getSlave2Nack() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_SLV2_NACK_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Slave 1 NACK interrupt status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getSlave1Nack() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_SLV1_NACK_BIT, buffer);
//    return buffer[0];
```

```
//}
// * 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.
// * @return Slave 0 NACK interrupt status
// * @see MPU6050_RA_I2C_MST_STATUS
// */
//bool MPU6050::getSlave0Nack() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS, MPU6050_MST_I2C_SLV0_NACK_BIT, buffer);
//    return buffer[0];
//}
//
//
// * Will be set 0 for active-high, 1 for active-low.
// * @return Current interrupt mode (0=active-high, 1=active-low)
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_INT_LEVEL_BIT
// */
//bool MPU6050::getInterruptMode() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_INT_LEVEL_BIT, buffer);
//    return buffer[0];
//}
// * @param mode New interrupt mode (0=active-high, 1=active-low)
// * @see getInterruptMode()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_INT_LEVEL_BIT
// */
//void MPU6050::setInterruptMode(bool mode) {
//   I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_INT_LEVEL_BIT, mode);
//}
// * Will be set 0 for push-pull, 1 for open-drain.
// * @return Current interrupt drive mode (0=push-pull, 1=open-drain)
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_INT_OPEN_BIT
// */
//bool MPU6050::getInterruptDrive() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_INT_OPEN_BIT, buffer);
//    return buffer[0];
//}
// * @param drive New interrupt drive mode (0=push-pull, 1=open-drain)
// * @see getInterruptDrive()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_INT_OPEN_BIT
// */
//void MPU6050::setInterruptDrive(bool drive) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_INT_OPEN_BIT, drive);
//}
// * Will be set 0 for 50us-pulse, 1 for latch-until-int-cleared.
// * @return Current latch mode (0=50us-pulse, 1=latch-until-int-cleared)
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_LATCH_INT_EN_BIT
// */
//bool MPU6050::getInterruptLatch() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_LATCH_INT_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param latch New latch mode (0=50us-pulse, 1=latch-until-int-cleared)
// * @see getInterruptLatch()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_LATCH_INT_EN_BIT
// */
//void MPU6050::setInterruptLatch(bool latch) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_LATCH_INT_EN_BIT, latch);
//}
// * Will be set 0 for status-read-only, 1 for any-register-read.
// * @return Current latch clear mode (0=status-read-only, 1=any-register-read)
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_INT_RD_CLEAR_BIT
// */
//bool MPU6050::getInterruptLatchClear() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_INT_RD_CLEAR_BIT, buffer);
//    return buffer[0];
//}
// * @param clear New latch clear mode (0=status-read-only, 1=any-register-read)
// * @see getInterruptLatchClear()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_INT_RD_CLEAR_BIT
// */
//void MPU6050::setInterruptLatchClear(bool clear) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_INT_RD_CLEAR_BIT, clear);
//}
// * @return Current FSYNC interrupt mode (0=active-high, 1=active-low)
// * @see getFSyncInterruptMode()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT
// */
//bool MPU6050::getFSyncInterruptLevel() {
```

```
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT, buffer);
//    return buffer[0];
//}
// * @param mode New FSYNC interrupt mode (0=active-high, 1=active-low)
// * @see getFSyncInterruptMode()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT
// */
//void MPU6050::setFSyncInterruptLevel(bool level) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT, level);
//}
// * Will be set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled setting
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_FSYNC_INT_EN_BIT
// */
//bool MPU6050::getFSyncInterruptEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_FSYNC_INT_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New FSYNC pin interrupt enabled setting
// * @see getFSyncInterruptEnabled()
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_FSYNC_INT_EN_BIT
// */
//void MPU6050::setFSyncInterruptEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_FSYNC_INT_EN_BIT, enabled);
//}
// * 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]).
// * @return Current I2C bypass enabled status
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_I2C_BYPASS_EN_BIT
// */
//bool MPU6050::getI2CBypassEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_I2C_BYPASS_EN_BIT, buffer);
//    return buffer[0];
//}
// * 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]).
// * @param enabled New I2C bypass enabled status
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_I2C_BYPASS_EN_BIT
// */
//void MPU6050::setI2CBypassEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_I2C_BYPASS_EN_BIT, enabled);
//}
// * 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.
// * @return Current reference clock output enabled status
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_CLKOUT_EN_BIT
// */
//bool MPU6050::getClockOutputEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_CLKOUT_EN_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @param enabled New reference clock output enabled status
// * @see MPU6050_RA_INT_PIN_CFG
// * @see MPU6050_INTCFG_CLKOUT_EN_BIT
// */
//void MPU6050::setClockOutputEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG, MPU6050_INTCFG_CLKOUT_EN_BIT, enabled);
//}
//
//
// * Full register byte for all interrupts, for quick reading. Each bit will be
// * set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_FF_BIT
// **/
//uint8_t MPU6050::getIntEnabled() {
```

```
//    I2Cdev::readByte(devAddr, MPU6050_RA_INT_ENABLE, buffer);
//    return buffer[0];
//}
// * Full register byte for all interrupts, for quick reading. Each bit should be
// * set 0 for disabled, 1 for enabled.
// * @param enabled New interrupt enabled status
// * @see getIntFreefallEnabled()
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_FF_BIT
// **/
//void MPU6050::setIntEnabled(uint8_t enabled) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_INT_ENABLE, enabled);
//}
// * Will be set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_FF_BIT
// **/
//bool MPU6050::getIntFreefallEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_FF_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New interrupt enabled status
// * @see getIntFreefallEnabled()
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_FF_BIT
// **/
//void MPU6050::setIntFreefallEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_FF_BIT, enabled);
//}
// * Will be set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_MOT_BIT
// **/
//bool MPU6050::getIntMotionEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_MOT_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New interrupt enabled status
// * @see getIntMotionEnabled()
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_MOT_BIT
// **/
//void MPU6050::setIntMotionEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_MOT_BIT, enabled);
//}
// * Will be set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_ZMOT_BIT
// **/
//bool MPU6050::getIntZeroMotionEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_ZMOT_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New interrupt enabled status
// * @see getIntZeroMotionEnabled()
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_ZMOT_BIT
// **/
//void MPU6050::setIntZeroMotionEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_ZMOT_BIT, enabled);
//}
// * Will be set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_FIFO_OFLOW_BIT
// **/
//bool MPU6050::getIntFIFOBufferOverflowEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_FIFO_OFLOW_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New interrupt enabled status
// * @see getIntFIFOBufferOverflowEnabled()
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_FIFO_OFLOW_BIT
// **/
//void MPU6050::setIntFIFOBufferOverflowEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_FIFO_OFLOW_BIT, enabled);
//}
// * This enables any of the I2C Master interrupt sources to generate an
// * interrupt. Will be set 0 for disabled, 1 for enabled.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_I2C_MST_INT_BIT
// **/
```

```
//bool MPU6050::getIntI2CMasterEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_I2C_MST_INT_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New interrupt enabled status
// * @see getIntI2CMasterEnabled()
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_I2C_MST_INT_BIT
// **/
//void MPU6050::setIntI2CMasterEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_I2C_MST_INT_BIT, enabled);
//}
// * 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.
// * @return Current interrupt enabled status
// * @see MPU6050_RA_INT_ENABLE
// * @see MPU6050_INTERRUPT_DATA_RDY_BIT
// */
//bool MPU6050::getIntDataReadyEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_DATA_RDY_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New interrupt enabled status
// * @see getIntDataReadyEnabled()
// * @see MPU6050_RA_INT_CFG
// * @see MPU6050_INTERRUPT_DATA_RDY_BIT
// */
//void MPU6050::setIntDataReadyEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_DATA_RDY_BIT, enabled);
//}
//
//
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// */
//uint8_t MPU6050::getIntStatus() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_INT_STATUS, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// * @see MPU6050_INTERRUPT_FF_BIT
// */
//bool MPU6050::getIntFreefallStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_FF_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// * @see MPU6050_INTERRUPT_MOT_BIT
// */
//bool MPU6050::getIntMotionStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_MOT_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// * @see MPU6050_INTERRUPT_ZMOT_BIT
// */
//bool MPU6050::getIntZeroMotionStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_ZMOT_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// * @see MPU6050_INTERRUPT_FIFO_OFLOW_BIT
// */
//bool MPU6050::getIntFIFOBufferOverflowStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_FIFO_OFLOW_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// * @see MPU6050_INTERRUPT_I2C_MST_INT_BIT
// */
//bool MPU6050::getIntI2CMasterStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_I2C_MST_INT_BIT, buffer);
//    return buffer[0];
//}
// * 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.
// * @return Current interrupt status
// * @see MPU6050_RA_INT_STATUS
// * @see MPU6050_INTERRUPT_DATA_RDY_BIT
// */
//bool MPU6050::getIntDataReadyStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_DATA_RDY_BIT, buffer);
//    return buffer[0];
//}
//
//
// * FUNCTION NOT FULLY IMPLEMENTED YET.
// * @param ax 16-bit signed integer container for accelerometer X-axis value
// * @param ay 16-bit signed integer container for accelerometer Y-axis value
// * @param az 16-bit signed integer container for accelerometer Z-axis value
// * @param gx 16-bit signed integer container for gyroscope X-axis value
// * @param gy 16-bit signed integer container for gyroscope Y-axis value
// * @param gz 16-bit signed integer container for gyroscope Z-axis value
// * @param mx 16-bit signed integer container for magnetometer X-axis value
// * @param my 16-bit signed integer container for magnetometer Y-axis value
// * @param mz 16-bit signed integer container for magnetometer Z-axis value
// * @see getMotion6()
// * @see getAcceleration()
// * @see getRotation()
// * @see MPU6050_RA_ACCEL_XOUT_H
// */
//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) {
//    getMotion6(ax, ay, az, gx, gy, gz);
//    // TODO: magnetometer integration
//}
// * Retrieves all currently available motion sensor values.
// * @param ax 16-bit signed integer container for accelerometer X-axis value
// * @param ay 16-bit signed integer container for accelerometer Y-axis value
// * @param az 16-bit signed integer container for accelerometer Z-axis value
// * @param gx 16-bit signed integer container for gyroscope X-axis value
// * @param gy 16-bit signed integer container for gyroscope Y-axis value
// * @param gz 16-bit signed integer container for gyroscope Z-axis value
// * @see getAcceleration()
// * @see getRotation()
// * @see MPU6050_RA_ACCEL_XOUT_H
// */
//void MPU6050::getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy, int16_t* gz) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14, buffer);
//    *ax = (((int16_t)buffer[0]) << 8) | buffer[1];
//    *ay = (((int16_t)buffer[2]) << 8) | buffer[3];
//    *az = (((int16_t)buffer[4]) << 8) | buffer[5];
//    *gx = (((int16_t)buffer[8]) << 8) | buffer[9];
//    *gy = (((int16_t)buffer[10]) << 8) | buffer[11];
//    *gz = (((int16_t)buffer[12]) << 8) | buffer[13];
//}
// * 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:
// *
// * <pre>
// * 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
// * </pre>
// *
// * @param x 16-bit signed integer container for X-axis acceleration
// * @param y 16-bit signed integer container for Y-axis acceleration
// * @param z 16-bit signed integer container for Z-axis acceleration
// * @see MPU6050_RA_GYRO_XOUT_H
// */
//void MPU6050::getAcceleration(int16_t* x, int16_t* y, int16_t* z) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 6, buffer);
//    *x = (((int16_t)buffer[0]) << 8) | buffer[1];
//    *y = (((int16_t)buffer[2]) << 8) | buffer[3];
//    *z = (((int16_t)buffer[4]) << 8) | buffer[5];
//}
// * @return X-axis acceleration measurement in 16-bit 2's complement format
// * @see getMotion6()
// * @see MPU6050_RA_ACCEL_XOUT_H
// */
//int16_t MPU6050::getAccelerationX() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
// * @return Y-axis acceleration measurement in 16-bit 2's complement format
// * @see getMotion6()
// * @see MPU6050_RA_ACCEL_YOUT_H
// */
//int16_t MPU6050::getAccelerationY() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_YOUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
// * @return Z-axis acceleration measurement in 16-bit 2's complement format
// * @see getMotion6()
// * @see MPU6050_RA_ACCEL_ZOUT_H
// */
//int16_t MPU6050::getAccelerationZ() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_ZOUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//
//
// * @return Temperature reading in 16-bit 2's complement format
// * @see MPU6050_RA_TEMP_OUT_H
// */
//int16_t MPU6050::getTemperature() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_TEMP_OUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//
//
// * 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:
// *
// * <pre>
// * FS_SEL | Full Scale Range   | LSB Sensitivity
// * -------+--------------------+----------------
// * 0      | +/- 250 degrees/s  | 131 LSB/deg/s
// * 1      | +/- 500 degrees/s  | 65.5 LSB/deg/s
// * 2      | +/- 1000 degrees/s | 32.8 LSB/deg/s
// * 3      | +/- 2000 degrees/s | 16.4 LSB/deg/s
// * </pre>
// *
// * @param x 16-bit signed integer container for X-axis rotation
// * @param y 16-bit signed integer container for Y-axis rotation
// * @param z 16-bit signed integer container for Z-axis rotation
// * @see getMotion6()
// * @see MPU6050_RA_GYRO_XOUT_H
// */
//void MPU6050::getRotation(int16_t* x, int16_t* y, int16_t* z) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 6, buffer);
//    *x = (((int16_t)buffer[0]) << 8) | buffer[1];
//    *y = (((int16_t)buffer[2]) << 8) | buffer[3];
//    *z = (((int16_t)buffer[4]) << 8) | buffer[5];
//}
//
//void MPU6050::getRotationXY(int16_t* x, int16_t* y) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 4, buffer);
//    *x = (((int16_t)buffer[0]) << 8) | buffer[1];
//    *y = (((int16_t)buffer[2]) << 8) | buffer[3];
//}
//
// * @return X-axis rotation measurement in 16-bit 2's complement format
// * @see getMotion6()
// * @see MPU6050_RA_GYRO_XOUT_H
// */
//int16_t MPU6050::getRotationX() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
// * @return Y-axis rotation measurement in 16-bit 2's complement format
// * @see getMotion6()
// * @see MPU6050_RA_GYRO_YOUT_H
// */
//int16_t MPU6050::getRotationY() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_YOUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
// * @return Z-axis rotation measurement in 16-bit 2's complement format
// * @see getMotion6()
// * @see MPU6050_RA_GYRO_ZOUT_H
// */
//int16_t MPU6050::getRotationZ() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_ZOUT_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//
//
// * 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_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.
// *
// * @param position Starting position (0-23)
// * @return Byte read from register
// */
//uint8_t MPU6050::getExternalSensorByte(int position) {
//    I2Cdev::readByte(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, buffer);
//    return buffer[0];
//}
// * @param position Starting position (0-21)
// * @return Word read from register
// * @see getExternalSensorByte()
// */
//uint16_t MPU6050::getExternalSensorWord(int position) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, 2, buffer);
//    return (((uint16_t)buffer[0]) << 8) | buffer[1];
//}
// * @param position Starting position (0-20)
// * @return Double word read from registers
// * @see getExternalSensorByte()
// */
//uint32_t MPU6050::getExternalSensorDWord(int position) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, 4, buffer);
//    return (((uint32_t)buffer[0]) << 24) | (((uint32_t)buffer[1]) << 16) | (((uint16_t)buffer[2]) << 8) | buffer[3];
//}
//
//
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_XNEG_BIT
// */
//bool MPU6050::getXNegMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_XNEG_BIT, buffer);
//    return buffer[0];
//}
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_XPOS_BIT
// */
//bool MPU6050::getXPosMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_XPOS_BIT, buffer);
//    return buffer[0];
//}
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_YNEG_BIT
// */
//bool MPU6050::getYNegMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_YNEG_BIT, buffer);
//    return buffer[0];
//}
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_YPOS_BIT
// */
//bool MPU6050::getYPosMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_YPOS_BIT, buffer);
//    return buffer[0];
//}
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_ZNEG_BIT
// */
//bool MPU6050::getZNegMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_ZNEG_BIT, buffer);
//    return buffer[0];
//}
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_ZPOS_BIT
// */
//bool MPU6050::getZPosMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_ZPOS_BIT, buffer);
//    return buffer[0];
//}
```

```
// * @return Motion detection status
// * @see MPU6050_RA_MOT_DETECT_STATUS
// * @see MPU6050_MOTION_MOT_ZRMOT_BIT
// */
//bool MPU6050::getZeroMotionDetected() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS, MPU6050_MOTION_MOT_ZRMOT_BIT, buffer);
//    return buffer[0];
//}
//
//
// * 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.
// * @param num Slave number (0-3)
// * @param data Byte to write
// * @see MPU6050_RA_I2C_SLV0_DO
// */
//void MPU6050::setSlaveOutputByte(uint8_t num, uint8_t data) {
//    if (num > 3) return;
//    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_DO + num, data);
//}
//
//
// * 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.
// * @return Current external data shadow delay enabled status.
// * @see MPU6050_RA_I2C_MST_DELAY_CTRL
// * @see MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT
// */
//bool MPU6050::getExternalShadowDelayEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New external data shadow delay enabled status.
// * @see getExternalShadowDelayEnabled()
// * @see MPU6050_RA_I2C_MST_DELAY_CTRL
// * @see MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT
// */
//void MPU6050::setExternalShadowDelayEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT, enabled);
//}
// * When a particular slave delay is enabled, the rate of access for the that
// * slave device is reduced. When a slave's access rate is decreased relative to
// * the Sample Rate, the slave is accessed every:
// *
// *     1 / (1 + I2C_MST_DLY) Samples
// *
// * This base Sample Rate in turn is determined by SMPLRT_DIV (register  * 25)
// * and DLPF_CFG (register 26).
// *
// * For further information regarding I2C_MST_DLY, please refer to register 52.
// * For further information regarding the Sample Rate, please refer to register 25.
// *
// * @param num Slave number (0-4)
// * @return Current slave delay enabled status.
// * @see MPU6050_RA_I2C_MST_DELAY_CTRL
// * @see MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT
// */
//bool MPU6050::getSlaveDelayEnabled(uint8_t num) {
//    // MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT is 4, SLV3 is 3, etc.
//    if (num > 4) return 0;
//    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, num, buffer);
//    return buffer[0];
//}
// * @param num Slave number (0-4)
// * @param enabled New slave delay enabled status.
// * @see MPU6050_RA_I2C_MST_DELAY_CTRL
// * @see MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT
// */
//void MPU6050::setSlaveDelayEnabled(uint8_t num, bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, num, enabled);
//}
//
//
// * The reset will revert the signal path analog to digital converters and
// * filters to their power up configurations.
// * @see MPU6050_RA_SIGNAL_PATH_RESET
// * @see MPU6050_PATHRESET_GYRO_RESET_BIT
// */
//void MPU6050::resetGyroscopePath() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET, MPU6050_PATHRESET_GYRO_RESET_BIT, true);
//}
// * The reset will revert the signal path analog to digital converters and
// * filters to their power up configurations.
// * @see MPU6050_RA_SIGNAL_PATH_RESET
// * @see MPU6050_PATHRESET_ACCEL_RESET_BIT
// */
//void MPU6050::resetAccelerometerPath() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET, MPU6050_PATHRESET_ACCEL_RESET_BIT, true);
//}
// * The reset will revert the signal path analog to digital converters and
// * filters to their power up configurations.
// * @see MPU6050_RA_SIGNAL_PATH_RESET
// * @see MPU6050_PATHRESET_TEMP_RESET_BIT
// */
//void MPU6050::resetTemperaturePath() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET, MPU6050_PATHRESET_TEMP_RESET_BIT, true);
//}
//
//
// * 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.
// * @return Current accelerometer power-on delay
// * @see MPU6050_RA_MOT_DETECT_CTRL
// * @see MPU6050_DETECT_ACCEL_ON_DELAY_BIT
// */
//uint8_t MPU6050::getAccelerometerPowerOnDelay() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL, MPU6050_DETECT_ACCEL_ON_DELAY_BIT, MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH, buffer);
//    return buffer[0];
//}
// * @param delay New accelerometer power-on delay (0-3)
// * @see getAccelerometerPowerOnDelay()
// * @see MPU6050_RA_MOT_DETECT_CTRL
// * @see MPU6050_DETECT_ACCEL_ON_DELAY_BIT
// */
//void MPU6050::setAccelerometerPowerOnDelay(uint8_t delay) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL, MPU6050_DETECT_ACCEL_ON_DELAY_BIT, MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH, delay);
//}
// * 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:
// *
// * <pre>
// * FF_COUNT | Counter Decrement
// * ---------+------------------
// * 0        | Reset
// * 1        | 1
// * 2        | 2
// * 3        | 4
// * </pre>
// *
// * 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.
// *
// * @return Current decrement configuration
// * @see MPU6050_RA_MOT_DETECT_CTRL
// * @see MPU6050_DETECT_FF_COUNT_BIT
// */
//uint8_t MPU6050::getFreefallDetectionCounterDecrement() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL, MPU6050_DETECT_FF_COUNT_BIT, MPU6050_DETECT_FF_COUNT_LENGTH, buffer);
//    return buffer[0];
//}
// * @param decrement New decrement configuration value
// * @see getFreefallDetectionCounterDecrement()
// * @see MPU6050_RA_MOT_DETECT_CTRL
// * @see MPU6050_DETECT_FF_COUNT_BIT
// */
//void MPU6050::setFreefallDetectionCounterDecrement(uint8_t decrement) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL, MPU6050_DETECT_FF_COUNT_BIT, MPU6050_DETECT_FF_COUNT_LENGTH, decrement);
//}
// * 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:
// *
// * <pre>
// * MOT_COUNT | Counter Decrement
// * ----------+------------------
// * 0         | Reset
// * 1         | 1
// * 2         | 2
// * 3         | 4
// * </pre>
// *
// * 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.
// *
// */
//uint8_t MPU6050::getMotionDetectionCounterDecrement() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL, MPU6050_DETECT_MOT_COUNT_BIT, MPU6050_DETECT_MOT_COUNT_LENGTH, buffer);
//    return buffer[0];
//}
// * @param decrement New decrement configuration value
// * @see getMotionDetectionCounterDecrement()
// * @see MPU6050_RA_MOT_DETECT_CTRL
// * @see MPU6050_DETECT_MOT_COUNT_BIT
// */
//void MPU6050::setMotionDetectionCounterDecrement(uint8_t decrement) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL, MPU6050_DETECT_MOT_COUNT_BIT, MPU6050_DETECT_MOT_COUNT_LENGTH, decrement);
//}
//
//
// * 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.
// * @return Current FIFO enabled status
// * @see MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_FIFO_EN_BIT
// */
//bool MPU6050::getFIFOEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_FIFO_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New FIFO enabled status
// * @see getFIFOEnabled()
// * @see MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_FIFO_EN_BIT
// */
//void MPU6050::setFIFOEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_FIFO_EN_BIT, enabled);
//}
// * 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.
// * @return Current I2C Master Mode enabled status
// * @see MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_I2C_MST_EN_BIT
// */
//bool MPU6050::getI2CMasterModeEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_I2C_MST_EN_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New I2C Master Mode enabled status
// * @see getI2CMasterModeEnabled()
// * @see MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_I2C_MST_EN_BIT
// */
//void MPU6050::setI2CMasterModeEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_I2C_MST_EN_BIT, enabled);
//}
// * If this is set, the primary SPI interface will be enabled in place of the
// * disabled primary I2C interface.
// */
//void MPU6050::switchSPIEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_I2C_IF_DIS_BIT, enabled);
//}
// * 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 MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_FIFO_RESET_BIT
// */
//void MPU6050::resetFIFO() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_FIFO_RESET_BIT, true);
//}
// * 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 MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_I2C_MST_RESET_BIT
// */
//void MPU6050::resetI2CMaster() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_I2C_MST_RESET_BIT, true);
//}
// * 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 MPU6050_RA_USER_CTRL
// * @see MPU6050_USERCTRL_SIG_COND_RESET_BIT
// */
//void MPU6050::resetSensors() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_SIG_COND_RESET_BIT, true);
//}
//
//
// * A small delay of ~50ms may be desirable after triggering a reset.
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_DEVICE_RESET_BIT
// */
//void MPU6050::reset() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_DEVICE_RESET_BIT, true);
//}
// * 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.
// * @return Current sleep mode enabled status
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_SLEEP_BIT
// */
//bool MPU6050::getSleepEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_SLEEP_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New sleep mode enabled status
// * @see getSleepEnabled()
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_SLEEP_BIT
// */
//void MPU6050::setSleepEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_SLEEP_BIT, enabled);
//}
// * 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).
// * @return Current sleep mode enabled status
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_CYCLE_BIT
// */
//bool MPU6050::getWakeCycleEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_CYCLE_BIT, buffer);
//    return buffer[0];
//}
// * @param enabled New sleep mode enabled status
// * @see getWakeCycleEnabled()
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_CYCLE_BIT
// */
//void MPU6050::setWakeCycleEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_CYCLE_BIT, enabled);
//}
// * 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.
// *
// * @return Current temperature sensor enabled status
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_TEMP_DIS_BIT
// */
//bool MPU6050::getTempSensorEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_TEMP_DIS_BIT, buffer);
//    return buffer[0] == 0; // 1 is actually disabled here
//}
```

```
// * 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.
// *
// * @param enabled New temperature sensor enabled status
// * @see getTempSensorEnabled()
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_TEMP_DIS_BIT
// */
//void MPU6050::setTempSensorEnabled(bool enabled) {
//    // 1 is actually disabled here
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_TEMP_DIS_BIT, !enabled);
//}
// * @return Current clock source setting
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_CLKSEL_BIT
// * @see MPU6050_PWR1_CLKSEL_LENGTH
// */
//uint8_t MPU6050::getClockSource() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_CLKSEL_BIT, MPU6050_PWR1_CLKSEL_LENGTH, buffer);
//    return buffer[0];
//}
// * 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:
// *
// * <pre>
// * 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
// * </pre>
// *
// * @param source New clock source setting
// * @see getClockSource()
// * @see MPU6050_RA_PWR_MGMT_1
// * @see MPU6050_PWR1_CLKSEL_BIT
// * @see MPU6050_PWR1_CLKSEL_LENGTH
// */
//void MPU6050::setClockSource(uint8_t source) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_CLKSEL_BIT, MPU6050_PWR1_CLKSEL_LENGTH, source);
//}
//
//
// * The MPU-60X0 can be put into Accerlerometer Only Low Power Mode by setting
// * PWRSEL to 1 in the Power Management 1 register (Register 107). In this mode,
// * the device will power off all devices except for the primary I2C interface,
// * waking only the accelerometer at fixed intervals to take a single
// * measurement. The frequency of wake-ups can be configured with LP_WAKE_CTRL
// * as shown below:
// *
// * <pre>
// * LP_WAKE_CTRL | Wake-up Frequency
// * -------------+------------------
// * 0            | 1.25 Hz
// * 1            | 2.5 Hz
// * 2            | 5 Hz
// * 3            | 10 Hz
// * <pre>
// *
// * For further information regarding the MPU-60X0's power modes, please refer to
// * Register 107.
// *
// * @return Current wake frequency
// * @see MPU6050_RA_PWR_MGMT_2
// */
//uint8_t MPU6050::getWakeFrequency() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_LP_WAKE_CTRL_BIT, MPU6050_PWR2_LP_WAKE_CTRL_LENGTH, buffer);
//    return buffer[0];
//}
// * @param frequency New wake frequency
// * @see MPU6050_RA_PWR_MGMT_2
// */
//void MPU6050::setWakeFrequency(uint8_t frequency) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_LP_WAKE_CTRL_BIT, MPU6050_PWR2_LP_WAKE_CTRL_LENGTH, frequency);
//}
//
// * If enabled, the X-axis will not gather or report data (or use power).
// * @return Current X-axis standby enabled status
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_XA_BIT
// */
//bool MPU6050::getStandbyXAccelEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_XA_BIT, buffer);
//    return buffer[0];
//}
// * @param New X-axis standby enabled status
// * @see getStandbyXAccelEnabled()
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_XA_BIT
// */
//void MPU6050::setStandbyXAccelEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_XA_BIT, enabled);
//}
// * If enabled, the Y-axis will not gather or report data (or use power).
// * @return Current Y-axis standby enabled status
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_YA_BIT
// */
//bool MPU6050::getStandbyYAccelEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_YA_BIT, buffer);
//    return buffer[0];
//}
// * @param New Y-axis standby enabled status
// * @see getStandbyYAccelEnabled()
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_YA_BIT
// */
//void MPU6050::setStandbyYAccelEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_YA_BIT, enabled);
//}
// * If enabled, the Z-axis will not gather or report data (or use power).
// * @return Current Z-axis standby enabled status
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_ZA_BIT
// */
//bool MPU6050::getStandbyZAccelEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_ZA_BIT, buffer);
//    return buffer[0];
//}
// * @param New Z-axis standby enabled status
// * @see getStandbyZAccelEnabled()
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_ZA_BIT
// */
//void MPU6050::setStandbyZAccelEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_ZA_BIT, enabled);
//}
// * If enabled, the X-axis will not gather or report data (or use power).
// * @return Current X-axis standby enabled status
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_XG_BIT
// */
//bool MPU6050::getStandbyXGyroEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_XG_BIT, buffer);
//    return buffer[0];
//}
// * @param New X-axis standby enabled status
// * @see getStandbyXGyroEnabled()
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_XG_BIT
// */
//void MPU6050::setStandbyXGyroEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_XG_BIT, enabled);
//}
// * If enabled, the Y-axis will not gather or report data (or use power).
// * @return Current Y-axis standby enabled status
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_YG_BIT
// */
//bool MPU6050::getStandbyYGyroEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_YG_BIT, buffer);
//    return buffer[0];
//}
// * @param New Y-axis standby enabled status
// * @see getStandbyYGyroEnabled()
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_YG_BIT
// */
//void MPU6050::setStandbyYGyroEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_YG_BIT, enabled);
//}
// * If enabled, the Z-axis will not gather or report data (or use power).
// * @return Current Z-axis standby enabled status
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_ZG_BIT
// */
//bool MPU6050::getStandbyZGyroEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_ZG_BIT, buffer);
//    return buffer[0];
//}
// * @param New Z-axis standby enabled status
// * @see getStandbyZGyroEnabled()
// * @see MPU6050_RA_PWR_MGMT_2
// * @see MPU6050_PWR2_STBY_ZG_BIT
// */
//void MPU6050::setStandbyZGyroEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2, MPU6050_PWR2_STBY_ZG_BIT, enabled);
//}
//
//
// * This value indicates the number of bytes stored in the FIFO buffer. This
// * number is in turn the number of bytes that can be read from the FIFO buffer
// * and it is directly proportional to the number of samples available given the
// * set of sensor data bound to be stored in the FIFO (register 35 and 36).
// * @return Current FIFO buffer size
// */
//uint16_t MPU6050::getFIFOCount() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_COUNTH, 2, buffer);
//    return (((uint16_t)buffer[0]) << 8) | buffer[1];
//}
//
//
// * This register is used to read and write data from the FIFO buffer. Data is
// * written to the FIFO in order of register number (from lowest to highest). If
// * all the FIFO enable flags (see below) are enabled and all External Sensor
// * Data registers (Registers 73 to 96) are associated with a Slave device, the
// * contents of registers 59 through 96 will be written in order at the Sample
// * Rate.
// *
// * The contents of the sensor data registers (Registers 59 to 96) are written
// * into the FIFO buffer when their corresponding FIFO enable flags are set to 1
// * in FIFO_EN (Register 35). An additional flag for the sensor data registers
// * associated with I2C Slave 3 can be found in I2C_MST_CTRL (Register 36).
// *
// * If the FIFO buffer has overflowed, the status bit FIFO_OFLOW_INT is
// * automatically set to 1. This bit is located in INT_STATUS (Register 58).
// * When the FIFO buffer has overflowed, the oldest data will be lost and new
// * data will be written to the FIFO.
// *
// * If the FIFO buffer is empty, reading this register will return the last byte
// * that was previously read from the FIFO until new data is available. The user
// * should check FIFO_COUNT to ensure that the FIFO buffer is not read when
// * empty.
// *
// * @return Byte from FIFO buffer
// */
//uint8_t MPU6050::getFIFOByte() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_FIFO_R_W, buffer);
//    return buffer[0];
//}
//void MPU6050::getFIFOBytes(uint8_t *data, uint8_t length) {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_R_W, length, data);
//}
// * @see getFIFOByte()
// * @see MPU6050_RA_FIFO_R_W
// */
//void MPU6050::setFIFOByte(uint8_t data) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_FIFO_R_W, data);
//}
//
//
// * This register is used to verify the identity of the device (0b110100, 0x34).
// * @return Device ID (6 bits only! should be 0x34)
// * @see MPU6050_RA_WHO_AM_I
// * @see MPU6050_WHO_AM_I_BIT
// * @see MPU6050_WHO_AM_I_LENGTH
// */
//uint8_t MPU6050::getDeviceID() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_WHO_AM_I, MPU6050_WHO_AM_I_BIT, MPU6050_WHO_AM_I_LENGTH, buffer);
//    return buffer[0];
//}
// * Write a new ID into the WHO_AM_I register (no idea why this should ever be
// * necessary though).
// * @param id New device ID to set.
// * @see getDeviceID()
// * @see MPU6050_RA_WHO_AM_I
// * @see MPU6050_WHO_AM_I_BIT
// * @see MPU6050_WHO_AM_I_LENGTH
// */
//void MPU6050::setDeviceID(uint8_t id) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_WHO_AM_I, MPU6050_WHO_AM_I_BIT, MPU6050_WHO_AM_I_LENGTH, id);
//}
//
//
//
//uint8_t MPU6050::getOTPBankValid() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_XG_OFFS_TC, MPU6050_TC_OTP_BNK_VLD_BIT, buffer);
//    return buffer[0];
//}
//void MPU6050::setOTPBankValid(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_XG_OFFS_TC, MPU6050_TC_OTP_BNK_VLD_BIT, enabled);
//}
//int8_t MPU6050::getXGyroOffset() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_XG_OFFS_TC, MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, buffer);
//    return buffer[0];
//}
//void MPU6050::setXGyroOffset(int8_t offset) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_XG_OFFS_TC, MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
//}
//
//
//int8_t MPU6050::getYGyroOffset() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_YG_OFFS_TC, MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, buffer);
//    return buffer[0];
//}
//void MPU6050::setYGyroOffset(int8_t offset) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_YG_OFFS_TC, MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
//}
//
//
//int8_t MPU6050::getZGyroOffset() {
//    I2Cdev::readBits(devAddr, MPU6050_RA_ZG_OFFS_TC, MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, buffer);
//    return buffer[0];
//}
//void MPU6050::setZGyroOffset(int8_t offset) {
//    I2Cdev::writeBits(devAddr, MPU6050_RA_ZG_OFFS_TC, MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
//}
//
//
//int8_t MPU6050::getXFineGain() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_X_FINE_GAIN, buffer);
//    return buffer[0];
//}
//void MPU6050::setXFineGain(int8_t gain) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_X_FINE_GAIN, gain);
//}
//
//
//int8_t MPU6050::getYFineGain() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_Y_FINE_GAIN, buffer);
//    return buffer[0];
//}
//void MPU6050::setYFineGain(int8_t gain) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_Y_FINE_GAIN, gain);
//}
//
//
//int8_t MPU6050::getZFineGain() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_Z_FINE_GAIN, buffer);
//    return buffer[0];
//}
//void MPU6050::setZFineGain(int8_t gain) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_Z_FINE_GAIN, gain);
//}
//
//
//int16_t MPU6050::getXAccelOffset() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_XA_OFFS_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//void MPU6050::setXAccelOffset(int16_t offset) {
//    I2Cdev::writeWord(devAddr, MPU6050_RA_XA_OFFS_H, offset);
//}
//
```

```
//
//int16_t MPU6050::getYAccelOffset() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_YA_OFFS_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//void MPU6050::setYAccelOffset(int16_t offset) {
//    I2Cdev::writeWord(devAddr, MPU6050_RA_YA_OFFS_H, offset);
//}
//
//
//int16_t MPU6050::getZAccelOffset() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ZA_OFFS_H, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//void MPU6050::setZAccelOffset(int16_t offset) {
//    I2Cdev::writeWord(devAddr, MPU6050_RA_ZA_OFFS_H, offset);
//}
//
//
//int16_t MPU6050::getXGyroOffsetUser() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_XG_OFFS_USRH, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//void MPU6050::setXGyroOffsetUser(int16_t offset) {
//    I2Cdev::writeWord(devAddr, MPU6050_RA_XG_OFFS_USRH, offset);
//}
//
//
//int16_t MPU6050::getYGyroOffsetUser() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_YG_OFFS_USRH, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//void MPU6050::setYGyroOffsetUser(int16_t offset) {
//    I2Cdev::writeWord(devAddr, MPU6050_RA_YG_OFFS_USRH, offset);
//}
//
//
//int16_t MPU6050::getZGyroOffsetUser() {
//    I2Cdev::readBytes(devAddr, MPU6050_RA_ZG_OFFS_USRH, 2, buffer);
//    return (((int16_t)buffer[0]) << 8) | buffer[1];
//}
//void MPU6050::setZGyroOffsetUser(int16_t offset) {
//    I2Cdev::writeWord(devAddr, MPU6050_RA_ZG_OFFS_USRH, offset);
//}
//
//
//bool MPU6050::getIntPLLReadyEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
//    return buffer[0];
//}
//void MPU6050::setIntPLLReadyEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_PLL_RDY_INT_BIT, enabled);
//}
//bool MPU6050::getIntDMPEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
//    return buffer[0];
//}
//void MPU6050::setIntDMPEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE, MPU6050_INTERRUPT_DMP_INT_BIT, enabled);
//}
//
//
//bool MPU6050::getDMPInt5Status() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS, MPU6050_DMPINT_5_BIT, buffer);
//    return buffer[0];
//}
//bool MPU6050::getDMPInt4Status() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS, MPU6050_DMPINT_4_BIT, buffer);
//    return buffer[0];
//}
//bool MPU6050::getDMPInt3Status() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS, MPU6050_DMPINT_3_BIT, buffer);
//    return buffer[0];
//}
//bool MPU6050::getDMPInt2Status() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS, MPU6050_DMPINT_2_BIT, buffer);
//    return buffer[0];
//}
//bool MPU6050::getDMPInt1Status() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS, MPU6050_DMPINT_1_BIT, buffer);
//    return buffer[0];
//}
//bool MPU6050::getDMPInt0Status() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS, MPU6050_DMPINT_0_BIT, buffer);
//    return buffer[0];
//}
//
```

```
//
//bool MPU6050::getIntPLLReadyStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
//    return buffer[0];
//}
//bool MPU6050::getIntDMPStatus() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS, MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
//    return buffer[0];
//}
//
//
//bool MPU6050::getDMPEnabled() {
//    I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_DMP_EN_BIT, buffer);
//    return buffer[0];
//}
//void MPU6050::setDMPEnabled(bool enabled) {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_DMP_EN_BIT, enabled);
//}
//void MPU6050::resetDMP() {
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, MPU6050_USERCTRL_DMP_RESET_BIT, true);
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, 0x00, true);
//    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL, 0x80 | 0x40 | 0x08, true);
//}
//
//
//void MPU6050::setMemoryBank(uint8_t bank, bool prefetchEnabled, bool userBank) {
//    bank &= 0x1F;
//    if (userBank) bank |= 0x20;
//    if (prefetchEnabled) bank |= 0x40;
//    I2Cdev::writeByte(devAddr, MPU6050_RA_BANK_SEL, bank);
//}
//
//
//void MPU6050::setMemoryStartAddress(uint8_t address) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_START_ADDR, address);
//}
//
//
//uint8_t MPU6050::readMemoryByte() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_MEM_R_W, buffer);
//    return buffer[0];
//}
//void MPU6050::writeMemoryByte(uint8_t data) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_R_W, data);
//}
//void MPU6050::readMemoryBlock(uint8_t *data, uint16_t dataSize, uint8_t bank, uint8_t address) {
//    setMemoryBank(bank);
//    setMemoryStartAddress(address);
//    uint8_t chunkSize;
//    for (uint16_t i = 0; i < dataSize;) {
//        // determine correct chunk size according to bank position and data size
//        chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;
//
//        // make sure we don’t go past the data size
//        if (i + chunkSize > dataSize) chunkSize = dataSize - i;
//
//        // make sure this chunk doesn’t go past the bank boundary (256 bytes)
//        if (chunkSize > 256 - address) chunkSize = 256 - address;
//
//        // read the chunk of data as specified
//        I2Cdev::readBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, data + i);
//
//        // increase byte index by [chunkSize]
//        i += chunkSize;
//
//        // uint8_t automatically wraps to 0 at 256
//        address += chunkSize;
//
//        // if we aren’t done, update bank (if necessary) and address
//        if (i < dataSize) {
//            if (address == 0) bank++;
//            setMemoryBank(bank);
//            setMemoryStartAddress(address);
//        }
//    }
//}
//bool MPU6050::writeMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank, uint8_t address, bool verify, bool useProgMem) {
//    setMemoryBank(bank);
//    setMemoryStartAddress(address);
//    uint8_t chunkSize;
//    uint8_t *verifyBuffer;
//    uint8_t *progBuffer;
//    uint16_t i;
//    uint8_t j;
//    if (verify) verifyBuffer = (uint8_t *)malloc(MPU6050_DMP_MEMORY_CHUNK_SIZE);
//    if (useProgMem) progBuffer = (uint8_t *)malloc(MPU6050_DMP_MEMORY_CHUNK_SIZE);
//    for (i = 0; i < dataSize;) {
//        // determine correct chunk size according to bank position and data size
//        chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;
//
//        // make sure we don’t go past the data size
//        if (i + chunkSize > dataSize) chunkSize = dataSize - i;
//
//        // make sure this chunk doesn’t go past the bank boundary (256 bytes)
//        if (chunkSize > 256 - address) chunkSize = 256 - address;
//
//        if (useProgMem) {
//            // write the chunk of data as specified
//            for (j = 0; j < chunkSize; j++) progBuffer[j] = pgm_read_byte(data + i + j);
//        } else {
//            // write the chunk of data as specified
//            progBuffer = (uint8_t *)data + i;
//        }
//
//        I2Cdev::writeBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, progBuffer);
//
//        // verify data if needed
//        if (verify && verifyBuffer) {
//            setMemoryBank(bank);
//            setMemoryStartAddress(address);
//            I2Cdev::readBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, verifyBuffer);
//            if (memcmp(progBuffer, verifyBuffer, chunkSize) != 0) {
//                /*Serial.print("Block write verification error, bank ");
//                Serial.print(bank, DEC);
//                Serial.print(", address ");
//                Serial.print(address, DEC);
//                Serial.print("!\nExpected:");
//                for (j = 0; j < chunkSize; j++) {
//                    Serial.print(" 0x");
//                    if (progBuffer[j] < 16) Serial.print("0");
//                    Serial.print(progBuffer[j], HEX);
//                }
//                Serial.print("\nReceived:");
//                for (uint8_t j = 0; j < chunkSize; j++) {
//                    Serial.print(" 0x");
//                    if (verifyBuffer[i + j] < 16) Serial.print("0");
//                    Serial.print(verifyBuffer[i + j], HEX);
//                }
//                Serial.print("\n");*/
//                free(verifyBuffer);
//                if (useProgMem) free(progBuffer);
//                return false; // uh oh.
//            }
//        }
//
//        // increase byte index by [chunkSize]
//        i += chunkSize;
//
//        // uint8_t automatically wraps to 0 at 256
//        address += chunkSize;
//
//        // if we aren’t done, update bank (if necessary) and address
//        if (i < dataSize) {
//            if (address == 0) bank++;
//            setMemoryBank(bank);
//            setMemoryStartAddress(address);
//        }
//    }
//    if (verify) free(verifyBuffer);
//    if (useProgMem) free(progBuffer);
//    return true;
//}
//bool MPU6050::writeProgMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank, uint8_t address, bool verify) {
//    return writeMemoryBlock(data, dataSize, bank, address, verify, true);
//}
//bool MPU6050::writeDMPConfigurationSet(const uint8_t *data, uint16_t dataSize, bool useProgMem) {
//    uint8_t *progBuffer, success, special;
//    uint16_t i, j;
//    if (useProgMem) {
//        progBuffer = (uint8_t *)malloc(8); // assume 8-byte blocks, realloc later if necessary
//    }
//
//    // config set data is a long string of blocks with the following structure:
//    // [bank] [offset] [length] [byte[0], byte[1], ..., byte[length]]
//    uint8_t bank, offset, length;
//    for (i = 0; i < dataSize;) {
//        if (useProgMem) {
//            bank = pgm_read_byte(data + i++);
//            offset = pgm_read_byte(data + i++);
//            length = pgm_read_byte(data + i++);
//        } else {
//            bank = data[i++];
//            offset = data[i++];
//            length = data[i++];
//        }
//
//        // write data or perform special action
//        if (length > 0) {
//            // regular block of data to write
//            /*Serial.print("Writing config block to bank ");
//            Serial.print(bank);
//            Serial.print(", offset ");
//            Serial.print(offset);
//            Serial.print(", length=");
//            Serial.println(length);*/
//            if (useProgMem) {
//                if (sizeof(progBuffer) < length) progBuffer = (uint8_t *)realloc(progBuffer, length);
//                for (j = 0; j < length; j++) progBuffer[j] = pgm_read_byte(data + i + j);
//            } else {
//                progBuffer = (uint8_t *)data + i;
//            }
//            success = writeMemoryBlock(progBuffer, length, bank, offset, true);
//            i += length;
//        } else {
//            // special instruction
//            // NOTE: this kind of behavior (what and when to do certain things)
//            // is totally undocumented. This code is in here based on observed
//            // behavior only, and exactly why (or even whether) it has to be here
//            // is anybody's guess for now.
//            if (useProgMem) {
//                special = pgm_read_byte(data + i++);
//            } else {
//                special = data[i++];
//            }
//            /*Serial.print("Special command code ");
//            Serial.print(special, HEX);
//            Serial.println(" found...");*/
//            if (special == 0x01) {
//                // enable DMP-related interrupts
//
//                //setIntZeroMotionEnabled(true);
//                //setIntFIFOBufferOverflowEnabled(true);
//                //setIntDMPEnabled(true);
//                I2Cdev::writeByte(devAddr, MPU6050_RA_INT_ENABLE, 0x32);  // single operation
//
//                success = true;
//            } else {
//                // unknown special command
//                success = false;
//            }
//        }
//
//        if (!success) {
//            if (useProgMem) free(progBuffer);
//            return false; // uh oh
//        }
//    }
//    if (useProgMem) free(progBuffer);
//    return true;
//}
//bool MPU6050::writeProgDMPConfigurationSet(const uint8_t *data, uint16_t dataSize) {
//    return writeDMPConfigurationSet(data, dataSize, true);
//}
//
//
//uint8_t MPU6050::getDMPConfig1() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_1, buffer);
//    return buffer[0];
//}
//void MPU6050::setDMPConfig1(uint8_t config) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_1, config);
//}
//
//
//uint8_t MPU6050::getDMPConfig2() {
//    I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_2, buffer);
//    return buffer[0];
//}
//void MPU6050::setDMPConfig2(uint8_t config) {
//    I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_2, config);
//}
//
//
//
//
```

5.7 GyroscopeMPU6050.h File Reference

5.8 GyroscopeMPU6050.h

```
// * Arduino - MPU6050 Gyroscope Driver
// *
// * Concrete implementation of a gyroscope driver for MPU6050 sensor.
// *
// * @author Dalmir da Silva <dalmirdasilva@gmail.com>
// */
//
//#ifndef __ARDUINO_DRIVER_GYROSCOPE_MPU6050_H__
//#define __ARDUINO_DRIVER_GYROSCOPE_MPU6050_H__ 1
//
// *
// */
//class ArduinoGyroscopeMPU6050 {
//
//    /**
//     * Get X rotation.
//     */
//    int getRotationX();
//
//    /**
//     * Get Y rotation.
//     */
//    int getRotationY();
//
//    /**
//     * Get Z rotation.
//     */
//    int getRotationZ();
//};
//
//#endif /* __ARDUINO_DRIVER_GYROSCOPE_MPU6050_H__ */
//
//
//
//
//I2Cdev device library code is placed under the MIT license
//Copyright (c) 2012 Jeff Rowberg
//
//Permission is hereby granted, free of charge, to any person obtaining a copy
//of this software and associated documentation files (the "Software"), to deal
//in the Software without restriction, including without limitation the rights
//to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
//copies of the Software, and to permit persons to whom the Software is
//furnished to do so, subject to the following conditions:
//
//The above copyright notice and this permission notice shall be included in
//all copies or substantial portions of the Software.
//
//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
//IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
//AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
//OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
//THE SOFTWARE.
//===============================================
//*/
//
//#ifndef _MPU6050_H_
//#define _MPU6050_H_
//
//#include "I2Cdev.h"
//#include "definitions.h"
//#include <avr/pgmspace.h>
//
//
//
//#define MPU6050_RA_XG_OFFS_TC       0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
//#define MPU6050_RA_YG_OFFS_TC       0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
//#define MPU6050_RA_ZG_OFFS_TC       0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
//#define MPU6050_RA_X_FINE_GAIN      0x03 //[7:0] X_FINE_GAIN
//#define MPU6050_RA_Y_FINE_GAIN      0x04 //[7:0] Y_FINE_GAIN
//#define MPU6050_RA_Z_FINE_GAIN      0x05 //[7:0] Z_FINE_GAIN
//#define MPU6050_RA_XA_OFFS_H        0x06 //[15:0] XA_OFFS
//#define MPU6050_RA_XA_OFFS_L_TC     0x07
//#define MPU6050_RA_YA_OFFS_H        0x08 //[15:0] YA_OFFS
//#define MPU6050_RA_YA_OFFS_L_TC     0x09
//#define MPU6050_RA_ZA_OFFS_H        0x0A //[15:0] ZA_OFFS
//#define MPU6050_RA_ZA_OFFS_L_TC     0x0B
//#define MPU6050_RA_XG_OFFS_USRH     0x13 //[15:0] XG_OFFS_USR
```

```
//#define MPU6050_RA_XG_OFFS_USRL     0x14
//#define MPU6050_RA_YG_OFFS_USRH     0x15 //[15:0] YG_OFFS_USR
//#define MPU6050_RA_YG_OFFS_USRL     0x16
//#define MPU6050_RA_ZG_OFFS_USRH     0x17 //[15:0] ZG_OFFS_USR
//#define MPU6050_RA_ZG_OFFS_USRL     0x18
//#define MPU6050_RA_SMPLRT_DIV       0x19
//#define MPU6050_RA_CONFIG           0x1A
//#define MPU6050_RA_GYRO_CONFIG      0x1B
//#define MPU6050_RA_ACCEL_CONFIG     0x1C
//#define MPU6050_RA_FF_THR           0x1D
//#define MPU6050_RA_FF_DUR           0x1E
//#define MPU6050_RA_MOT_THR          0x1F
//#define MPU6050_RA_MOT_DUR          0x20
//#define MPU6050_RA_ZRMOT_THR        0x21
//#define MPU6050_RA_ZRMOT_DUR        0x22
//#define MPU6050_RA_FIFO_EN          0x23
//#define MPU6050_RA_I2C_MST_CTRL     0x24
//#define MPU6050_RA_I2C_SLV0_ADDR    0x25
//#define MPU6050_RA_I2C_SLV0_REG     0x26
//#define MPU6050_RA_I2C_SLV0_CTRL    0x27
//#define MPU6050_RA_I2C_SLV1_ADDR    0x28
//#define MPU6050_RA_I2C_SLV1_REG     0x29
//#define MPU6050_RA_I2C_SLV1_CTRL    0x2A
//#define MPU6050_RA_I2C_SLV2_ADDR    0x2B
//#define MPU6050_RA_I2C_SLV2_REG     0x2C
//#define MPU6050_RA_I2C_SLV2_CTRL    0x2D
//#define MPU6050_RA_I2C_SLV3_ADDR    0x2E
//#define MPU6050_RA_I2C_SLV3_REG     0x2F
//#define MPU6050_RA_I2C_SLV3_CTRL    0x30
//#define MPU6050_RA_I2C_SLV4_ADDR    0x31
//#define MPU6050_RA_I2C_SLV4_REG     0x32
//#define MPU6050_RA_I2C_SLV4_DO      0x33
//#define MPU6050_RA_I2C_SLV4_CTRL    0x34
//#define MPU6050_RA_I2C_SLV4_DI      0x35
//#define MPU6050_RA_I2C_MST_STATUS   0x36
//#define MPU6050_RA_INT_PIN_CFG      0x37
//#define MPU6050_RA_INT_ENABLE       0x38
//#define MPU6050_RA_DMP_INT_STATUS   0x39
//#define MPU6050_RA_INT_STATUS       0x3A
//#define MPU6050_RA_ACCEL_XOUT_H     0x3B
//#define MPU6050_RA_ACCEL_XOUT_L     0x3C
//#define MPU6050_RA_ACCEL_YOUT_H     0x3D
//#define MPU6050_RA_ACCEL_YOUT_L     0x3E
//#define MPU6050_RA_ACCEL_ZOUT_H     0x3F
//#define MPU6050_RA_ACCEL_ZOUT_L     0x40
//#define MPU6050_RA_TEMP_OUT_H       0x41
//#define MPU6050_RA_TEMP_OUT_L       0x42
//#define MPU6050_RA_GYRO_XOUT_H      0x43
//#define MPU6050_RA_GYRO_XOUT_L      0x44
//#define MPU6050_RA_GYRO_YOUT_H      0x45
//#define MPU6050_RA_GYRO_YOUT_L      0x46
//#define MPU6050_RA_GYRO_ZOUT_H      0x47
//#define MPU6050_RA_GYRO_ZOUT_L      0x48
//#define MPU6050_RA_EXT_SENS_DATA_00 0x49
//#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
//#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
//#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
//#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
//#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
//#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
//#define MPU6050_RA_EXT_SENS_DATA_07 0x50
//#define MPU6050_RA_EXT_SENS_DATA_08 0x51
//#define MPU6050_RA_EXT_SENS_DATA_09 0x52
//#define MPU6050_RA_EXT_SENS_DATA_10 0x53
//#define MPU6050_RA_EXT_SENS_DATA_11 0x54
//#define MPU6050_RA_EXT_SENS_DATA_12 0x55
//#define MPU6050_RA_EXT_SENS_DATA_13 0x56
//#define MPU6050_RA_EXT_SENS_DATA_14 0x57
//#define MPU6050_RA_EXT_SENS_DATA_15 0x58
//#define MPU6050_RA_EXT_SENS_DATA_16 0x59
//#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
//#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
//#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
//#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
//#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
//#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
//#define MPU6050_RA_EXT_SENS_DATA_23 0x60
//#define MPU6050_RA_MOT_DETECT_STATUS    0x61
//#define MPU6050_RA_I2C_SLV0_DO      0x63
//#define MPU6050_RA_I2C_SLV1_DO      0x64
//#define MPU6050_RA_I2C_SLV2_DO      0x65
//#define MPU6050_RA_I2C_SLV3_DO      0x66
//#define MPU6050_RA_I2C_MST_DELAY_CTRL   0x67
//#define MPU6050_RA_SIGNAL_PATH_RESET    0x68
//#define MPU6050_RA_MOT_DETECT_CTRL      0x69
//#define MPU6050_RA_USER_CTRL        0x6A
//#define MPU6050_RA_PWR_MGMT_1       0x6B
```

```
//#define MPU6050_RA_PWR_MGMT_2       0x6C
//#define MPU6050_RA_BANK_SEL         0x6D
//#define MPU6050_RA_MEM_START_ADDR   0x6E
//#define MPU6050_RA_MEM_R_W          0x6F
//#define MPU6050_RA_DMP_CFG_1        0x70
//#define MPU6050_RA_DMP_CFG_2        0x71
//#define MPU6050_RA_FIFO_COUNTH      0x72
//#define MPU6050_RA_FIFO_COUNTL      0x73
//#define MPU6050_RA_FIFO_R_W         0x74
//#define MPU6050_RA_WHO_AM_I         0x75
//
//#define MPU6050_TC_PWR_MODE_BIT     7
//#define MPU6050_TC_OFFSET_BIT       6
//#define MPU6050_TC_OFFSET_LENGTH    6
//#define MPU6050_TC_OTP_BNK_VLD_BIT  0
//
//#define MPU6050_VDDIO_LEVEL_VLOGIC  0
//#define MPU6050_VDDIO_LEVEL_VDD     1
//
//#define MPU6050_CFG_EXT_SYNC_SET_BIT    5
//#define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
//#define MPU6050_CFG_DLPF_CFG_BIT    2
//#define MPU6050_CFG_DLPF_CFG_LENGTH 3
//
//#define MPU6050_EXT_SYNC_DISABLED       0x0
//#define MPU6050_EXT_SYNC_TEMP_OUT_L     0x1
//#define MPU6050_EXT_SYNC_GYRO_XOUT_L    0x2
//#define MPU6050_EXT_SYNC_GYRO_YOUT_L    0x3
//#define MPU6050_EXT_SYNC_GYRO_ZOUT_L    0x4
//#define MPU6050_EXT_SYNC_ACCEL_XOUT_L   0x5
//#define MPU6050_EXT_SYNC_ACCEL_YOUT_L   0x6
//#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L   0x7
//
//#define MPU6050_DLPF_BW_256         0x00
//#define MPU6050_DLPF_BW_188         0x01
//#define MPU6050_DLPF_BW_98          0x02
//#define MPU6050_DLPF_BW_42          0x03
//#define MPU6050_DLPF_BW_20          0x04
//#define MPU6050_DLPF_BW_10          0x05
//#define MPU6050_DLPF_BW_5           0x06
//
//#define MPU6050_GCONFIG_FS_SEL_BIT      4
//#define MPU6050_GCONFIG_FS_SEL_LENGTH   2
//
//#define MPU6050_GYRO_FS_250         0x00
//#define MPU6050_GYRO_FS_500         0x01
//#define MPU6050_GYRO_FS_1000        0x02
//#define MPU6050_GYRO_FS_2000        0x03
//
//#define MPU6050_ACONFIG_XA_ST_BIT           7
//#define MPU6050_ACONFIG_YA_ST_BIT           6
//#define MPU6050_ACONFIG_ZA_ST_BIT           5
//#define MPU6050_ACONFIG_AFS_SEL_BIT         4
//#define MPU6050_ACONFIG_AFS_SEL_LENGTH      2
//#define MPU6050_ACONFIG_ACCEL_HPF_BIT       2
//#define MPU6050_ACONFIG_ACCEL_HPF_LENGTH    3
//
//#define MPU6050_ACCEL_FS_2          0x00
//#define MPU6050_ACCEL_FS_4          0x01
//#define MPU6050_ACCEL_FS_8          0x02
//#define MPU6050_ACCEL_FS_16         0x03
//
//#define MPU6050_DHPF_RESET          0x00
//#define MPU6050_DHPF_5              0x01
//#define MPU6050_DHPF_2P5            0x02
//#define MPU6050_DHPF_1P25           0x03
//#define MPU6050_DHPF_0P63           0x04
//#define MPU6050_DHPF_HOLD           0x07
//
//#define MPU6050_TEMP_FIFO_EN_BIT    7
//#define MPU6050_XG_FIFO_EN_BIT      6
//#define MPU6050_YG_FIFO_EN_BIT      5
//#define MPU6050_ZG_FIFO_EN_BIT      4
//#define MPU6050_ACCEL_FIFO_EN_BIT   3
//#define MPU6050_SLV2_FIFO_EN_BIT    2
//#define MPU6050_SLV1_FIFO_EN_BIT    1
//#define MPU6050_SLV0_FIFO_EN_BIT    0
//
//#define MPU6050_MULT_MST_EN_BIT     7
//#define MPU6050_WAIT_FOR_ES_BIT     6
//#define MPU6050_SLV_3_FIFO_EN_BIT   5
//#define MPU6050_I2C_MST_P_NSR_BIT   4
//#define MPU6050_I2C_MST_CLK_BIT     3
//#define MPU6050_I2C_MST_CLK_LENGTH  4
//
//#define MPU6050_CLOCK_DIV_348       0x0
//#define MPU6050_CLOCK_DIV_333       0x1
```

```
//#define MPU6050_CLOCK_DIV_320       0x2
//#define MPU6050_CLOCK_DIV_308       0x3
//#define MPU6050_CLOCK_DIV_296       0x4
//#define MPU6050_CLOCK_DIV_286       0x5
//#define MPU6050_CLOCK_DIV_276       0x6
//#define MPU6050_CLOCK_DIV_267       0x7
//#define MPU6050_CLOCK_DIV_258       0x8
//#define MPU6050_CLOCK_DIV_500       0x9
//#define MPU6050_CLOCK_DIV_471       0xA
//#define MPU6050_CLOCK_DIV_444       0xB
//#define MPU6050_CLOCK_DIV_421       0xC
//#define MPU6050_CLOCK_DIV_400       0xD
//#define MPU6050_CLOCK_DIV_381       0xE
//#define MPU6050_CLOCK_DIV_364       0xF
//
//#define MPU6050_I2C_SLV_RW_BIT      7
//#define MPU6050_I2C_SLV_ADDR_BIT    6
//#define MPU6050_I2C_SLV_ADDR_LENGTH 7
//#define MPU6050_I2C_SLV_EN_BIT      7
//#define MPU6050_I2C_SLV_BYTE_SW_BIT 6
//#define MPU6050_I2C_SLV_REG_DIS_BIT 5
//#define MPU6050_I2C_SLV_GRP_BIT     4
//#define MPU6050_I2C_SLV_LEN_BIT     3
//#define MPU6050_I2C_SLV_LEN_LENGTH  4
//
//#define MPU6050_I2C_SLV4_RW_BIT         7
//#define MPU6050_I2C_SLV4_ADDR_BIT       6
//#define MPU6050_I2C_SLV4_ADDR_LENGTH    7
//#define MPU6050_I2C_SLV4_EN_BIT         7
//#define MPU6050_I2C_SLV4_INT_EN_BIT     6
//#define MPU6050_I2C_SLV4_REG_DIS_BIT    5
//#define MPU6050_I2C_SLV4_MST_DLY_BIT    4
//#define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5
//
//#define MPU6050_MST_PASS_THROUGH_BIT    7
//#define MPU6050_MST_I2C_SLV4_DONE_BIT   6
//#define MPU6050_MST_I2C_LOST_ARB_BIT    5
//#define MPU6050_MST_I2C_SLV4_NACK_BIT   4
//#define MPU6050_MST_I2C_SLV3_NACK_BIT   3
//#define MPU6050_MST_I2C_SLV2_NACK_BIT   2
//#define MPU6050_MST_I2C_SLV1_NACK_BIT   1
//#define MPU6050_MST_I2C_SLV0_NACK_BIT   0
//
//#define MPU6050_INTCFG_INT_LEVEL_BIT        7
//#define MPU6050_INTCFG_INT_OPEN_BIT         6
//#define MPU6050_INTCFG_LATCH_INT_EN_BIT     5
//#define MPU6050_INTCFG_INT_RD_CLEAR_BIT     4
//#define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT  3
//#define MPU6050_INTCFG_FSYNC_INT_EN_BIT     2
//#define MPU6050_INTCFG_I2C_BYPASS_EN_BIT    1
//#define MPU6050_INTCFG_CLKOUT_EN_BIT        0
//
//#define MPU6050_INTMODE_ACTIVEHIGH  0x00
//#define MPU6050_INTMODE_ACTIVELOW   0x01
//
//#define MPU6050_INTDRV_PUSHPULL     0x00
//#define MPU6050_INTDRV_OPENDRAIN    0x01
//
//#define MPU6050_INTLATCH_50USPULSE  0x00
//#define MPU6050_INTLATCH_WAITCLEAR  0x01
//
//#define MPU6050_INTCLEAR_STATUSREAD 0x00
//#define MPU6050_INTCLEAR_ANYREAD    0x01
//
//#define MPU6050_INTERRUPT_FF_BIT            7
//#define MPU6050_INTERRUPT_MOT_BIT           6
//#define MPU6050_INTERRUPT_ZMOT_BIT          5
//#define MPU6050_INTERRUPT_FIFO_OFLOW_BIT    4
//#define MPU6050_INTERRUPT_I2C_MST_INT_BIT   3
//#define MPU6050_INTERRUPT_PLL_RDY_INT_BIT   2
//#define MPU6050_INTERRUPT_DMP_INT_BIT       1
//#define MPU6050_INTERRUPT_DATA_RDY_BIT      0
//
//#define MPU6050_DMPINT_5_BIT            5
//#define MPU6050_DMPINT_4_BIT            4
//#define MPU6050_DMPINT_3_BIT            3
//#define MPU6050_DMPINT_2_BIT            2
//#define MPU6050_DMPINT_1_BIT            1
//#define MPU6050_DMPINT_0_BIT            0
//
//#define MPU6050_MOTION_MOT_XNEG_BIT     7
//#define MPU6050_MOTION_MOT_XPOS_BIT     6
//#define MPU6050_MOTION_MOT_YNEG_BIT     5
//#define MPU6050_MOTION_MOT_YPOS_BIT     4
//#define MPU6050_MOTION_MOT_ZNEG_BIT     3
//#define MPU6050_MOTION_MOT_ZPOS_BIT     2
//#define MPU6050_MOTION_MOT_ZRMOT_BIT    0
```

```
//
//#define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT   7
//#define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT   4
//#define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT   3
//#define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT   2
//#define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT   1
//#define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT   0
//
//#define MPU6050_PATHRESET_GYRO_RESET_BIT    2
//#define MPU6050_PATHRESET_ACCEL_RESET_BIT   1
//#define MPU6050_PATHRESET_TEMP_RESET_BIT    0
//
//#define MPU6050_DETECT_ACCEL_ON_DELAY_BIT       5
//#define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH    2
//#define MPU6050_DETECT_FF_COUNT_BIT             3
//#define MPU6050_DETECT_FF_COUNT_LENGTH          2
//#define MPU6050_DETECT_MOT_COUNT_BIT            1
//#define MPU6050_DETECT_MOT_COUNT_LENGTH         2
//
//#define MPU6050_DETECT_DECREMENT_RESET  0x0
//#define MPU6050_DETECT_DECREMENT_1      0x1
//#define MPU6050_DETECT_DECREMENT_2      0x2
//#define MPU6050_DETECT_DECREMENT_4      0x3
//
//#define MPU6050_USERCTRL_DMP_EN_BIT             7
//#define MPU6050_USERCTRL_FIFO_EN_BIT            6
//#define MPU6050_USERCTRL_I2C_MST_EN_BIT         5
//#define MPU6050_USERCTRL_I2C_IF_DIS_BIT         4
//#define MPU6050_USERCTRL_DMP_RESET_BIT          3
//#define MPU6050_USERCTRL_FIFO_RESET_BIT         2
//#define MPU6050_USERCTRL_I2C_MST_RESET_BIT      1
//#define MPU6050_USERCTRL_SIG_COND_RESET_BIT     0
//
//#define MPU6050_PWR1_DEVICE_RESET_BIT   7
//#define MPU6050_PWR1_SLEEP_BIT          6
//#define MPU6050_PWR1_CYCLE_BIT          5
//#define MPU6050_PWR1_TEMP_DIS_BIT       3
//#define MPU6050_PWR1_CLKSEL_BIT         2
//#define MPU6050_PWR1_CLKSEL_LENGTH      3
//
//#define MPU6050_CLOCK_INTERNAL          0x00
//#define MPU6050_CLOCK_PLL_XGYRO         0x01
//#define MPU6050_CLOCK_PLL_YGYRO         0x02
//#define MPU6050_CLOCK_PLL_ZGYRO         0x03
//#define MPU6050_CLOCK_PLL_EXT32K        0x04
//#define MPU6050_CLOCK_PLL_EXT19M        0x05
//#define MPU6050_CLOCK_KEEP_RESET        0x07
//
//#define MPU6050_PWR2_LP_WAKE_CTRL_BIT       7
//#define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH    2
//#define MPU6050_PWR2_STBY_XA_BIT            5
//#define MPU6050_PWR2_STBY_YA_BIT            4
//#define MPU6050_PWR2_STBY_ZA_BIT            3
//#define MPU6050_PWR2_STBY_XG_BIT            2
//#define MPU6050_PWR2_STBY_YG_BIT            1
//#define MPU6050_PWR2_STBY_ZG_BIT            0
//
//#define MPU6050_WAKE_FREQ_1P25      0x0
//#define MPU6050_WAKE_FREQ_2P5       0x1
//#define MPU6050_WAKE_FREQ_5         0x2
//#define MPU6050_WAKE_FREQ_10        0x3
//
//#define MPU6050_BANKSEL_PRFTCH_EN_BIT       6
//#define MPU6050_BANKSEL_CFG_USER_BANK_BIT   5
//#define MPU6050_BANKSEL_MEM_SEL_BIT         4
//#define MPU6050_BANKSEL_MEM_SEL_LENGTH      5
//
//#define MPU6050_WHO_AM_I_BIT        6
//#define MPU6050_WHO_AM_I_LENGTH     6
//
//#define MPU6050_DMP_MEMORY_BANKS        8
//#define MPU6050_DMP_MEMORY_BANK_SIZE    256
//#define MPU6050_DMP_MEMORY_CHUNK_SIZE   16
//
//
//class MPU6050 {
//    public:
//        MPU6050();
//        MPU6050(uint8_t address);
//
//        void initialize();
//        bool testConnection();
//
//        // AUX_VDDIO register
//        uint8_t getAuxVDDIOLevel();
//        void setAuxVDDIOLevel(uint8_t level);
//
```

```
//        // SMPLRT_DIV register
//        uint8_t getRate();
//        void setRate(uint8_t rate);
//
//        // CONFIG register
//        uint8_t getExternalFrameSync();
//        void setExternalFrameSync(uint8_t sync);
//        uint8_t getDLPFMode();
//        void setDLPFMode(uint8_t bandwidth);
//
//        // GYRO_CONFIG register
//        uint8_t getFullScaleGyroRange();
//        void setFullScaleGyroRange(uint8_t range);
//
//        // ACCEL_CONFIG register
//        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);
//
//        // FF_THR register
//        uint8_t getFreefallDetectionThreshold();
//        void setFreefallDetectionThreshold(uint8_t threshold);
//
//        // FF_DUR register
//        uint8_t getFreefallDetectionDuration();
//        void setFreefallDetectionDuration(uint8_t duration);
//
//        // MOT_THR register
//        uint8_t getMotionDetectionThreshold();
//        void setMotionDetectionThreshold(uint8_t threshold);
//
//        // MOT_DUR register
//        uint8_t getMotionDetectionDuration();
//        void setMotionDetectionDuration(uint8_t duration);
//
//        // ZRMOT_THR register
//        uint8_t getZeroMotionDetectionThreshold();
//        void setZeroMotionDetectionThreshold(uint8_t threshold);
//
//        // ZRMOT_DUR register
//        uint8_t getZeroMotionDetectionDuration();
//        void setZeroMotionDetectionDuration(uint8_t duration);
//
//        // FIFO_EN register
//        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);
//
//        // I2C_MST_CTRL register
//        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);
//
//        // I2C_SLV* registers (Slave 0-3)
//        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);
//
//        // I2C_SLV* registers (Slave 4)
//        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 getSlate4InputByte();
//
//        // I2C_MST_STATUS register
//        bool getPassthroughStatus();
//        bool getSlave4IsDone();
//        bool getLostArbitration();
//        bool getSlave4Nack();
//        bool getSlave3Nack();
//        bool getSlave2Nack();
//        bool getSlave1Nack();
//        bool getSlave0Nack();
//
//        // INT_PIN_CFG register
//        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);
//
//        // INT_ENABLE register
//        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);
//
//        // INT_STATUS register
//        uint8_t getIntStatus();
//        bool getIntFreefallStatus();
//        bool getIntMotionStatus();
//        bool getIntZeroMotionStatus();
//        bool getIntFIFOBufferOverflowStatus();
//        bool getIntI2CMasterStatus();
//        bool getIntDataReadyStatus();
//
//        // ACCEL_*OUT_* registers
//        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();
//
```

```
//        // TEMP_OUT_* registers
//        int16_t getTemperature();
//
//        // GYRO_*OUT_* registers
//        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();
//
//        // EXT_SENS_DATA_* registers
//        uint8_t getExternalSensorByte(int position);
//        uint16_t getExternalSensorWord(int position);
//        uint32_t getExternalSensorDWord(int position);
//
//        // MOT_DETECT_STATUS register
//        bool getXNegMotionDetected();
//        bool getXPosMotionDetected();
//        bool getYNegMotionDetected();
//        bool getYPosMotionDetected();
//        bool getZNegMotionDetected();
//        bool getZPosMotionDetected();
//        bool getZeroMotionDetected();
//
//        // I2C_SLV*_DO register
//        void setSlaveOutputByte(uint8_t num, uint8_t data);
//
//        // I2C_MST_DELAY_CTRL register
//        bool getExternalShadowDelayEnabled();
//        void setExternalShadowDelayEnabled(bool enabled);
//        bool getSlaveDelayEnabled(uint8_t num);
//        void setSlaveDelayEnabled(uint8_t num, bool enabled);
//
//        // SIGNAL_PATH_RESET register
//        void resetGyroscopePath();
//        void resetAccelerometerPath();
//        void resetTemperaturePath();
//
//        // MOT_DETECT_CTRL register
//        uint8_t getAccelerometerPowerOnDelay();
//        void setAccelerometerPowerOnDelay(uint8_t delay);
//        uint8_t getFreefallDetectionCounterDecrement();
//        void setFreefallDetectionCounterDecrement(uint8_t decrement);
//        uint8_t getMotionDetectionCounterDecrement();
//        void setMotionDetectionCounterDecrement(uint8_t decrement);
//
//        // USER_CTRL register
//        bool getFIFOEnabled();
//        void setFIFOEnabled(bool enabled);
//        bool getI2CMasterModeEnabled();
//        void setI2CMasterModeEnabled(bool enabled);
//        void switchSPIEnabled(bool enabled);
//        void resetFIFO();
//        void resetI2CMaster();
//        void resetSensors();
//
//        // PWR_MGMT_1 register
//        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);
//
//        // PWR_MGMT_2 register
//        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);
//
//        // FIFO_COUNT_* registers
//        uint16_t getFIFOCount();
//
//        // FIFO_R_W register
//        uint8_t getFIFOByte();
//        void setFIFOByte(uint8_t data);
//        void getFIFOBytes(uint8_t *data, uint8_t length);
//
//        // WHO_AM_I register
//        uint8_t getDeviceID();
//        void setDeviceID(uint8_t id);
//
//        // ======== UNDOCUMENTED/DMP REGISTERS/METHODS ========
//
//        // XG_OFFS_TC register
//        uint8_t getOTPBankValid();
//        void setOTPBankValid(bool enabled);
//        int8_t getXGyroOffset();
//        void setXGyroOffset(int8_t offset);
//
//        // YG_OFFS_TC register
//        int8_t getYGyroOffset();
//        void setYGyroOffset(int8_t offset);
//
//        // ZG_OFFS_TC register
//        int8_t getZGyroOffset();
//        void setZGyroOffset(int8_t offset);
//
//        // X_FINE_GAIN register
//        int8_t getXFineGain();
//        void setXFineGain(int8_t gain);
//
//        // Y_FINE_GAIN register
//        int8_t getYFineGain();
//        void setYFineGain(int8_t gain);
//
//        // Z_FINE_GAIN register
//        int8_t getZFineGain();
//        void setZFineGain(int8_t gain);
//
//        // XA_OFFS_* registers
//        int16_t getXAccelOffset();
//        void setXAccelOffset(int16_t offset);
//
//        // YA_OFFS_* register
//        int16_t getYAccelOffset();
//        void setYAccelOffset(int16_t offset);
//
//        // ZA_OFFS_* register
//        int16_t getZAccelOffset();
//        void setZAccelOffset(int16_t offset);
//
//        // XG_OFFS_USR* registers
//        int16_t getXGyroOffsetUser();
//        void setXGyroOffsetUser(int16_t offset);
//
//        // YG_OFFS_USR* register
//        int16_t getYGyroOffsetUser();
//        void setYGyroOffsetUser(int16_t offset);
//
//        // ZG_OFFS_USR* register
//        int16_t getZGyroOffsetUser();
//        void setZGyroOffsetUser(int16_t offset);
//
//        // INT_ENABLE register (DMP functions)
//        bool getIntPLLReadyEnabled();
//        void setIntPLLReadyEnabled(bool enabled);
//        bool getIntDMPEnabled();
//        void setIntDMPEnabled(bool enabled);
//
//        // DMP_INT_STATUS
//        bool getDMPInt5Status();
//        bool getDMPInt4Status();
//        bool getDMPInt3Status();
//        bool getDMPInt2Status();
//        bool getDMPInt1Status();
//        bool getDMPInt0Status();
//
//        // INT_STATUS register (DMP functions)
//        bool getIntPLLReadyStatus();
//        bool getIntDMPStatus();
//
//        // USER_CTRL register (DMP functions)
//        bool getDMPEnabled();
//        void setDMPEnabled(bool enabled);
//        void resetDMP();
//
//        // BANK_SEL register
//        void setMemoryBank(uint8_t bank, bool prefetchEnabled=false, bool userBank=false);
//
//        // MEM_START_ADDR register
//        void setMemoryStartAddress(uint8_t address);
//
//        // MEM_R_W register
//        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);
//
//        // DMP_CFG_1 register
//        uint8_t getDMPConfig1();
//        void setDMPConfig1(uint8_t config);
//
//        // DMP_CFG_2 register
//        uint8_t getDMPConfig2();
//        void setDMPConfig2(uint8_t config);
//
//
//        // special methods for MotionApps 2.0 implementation
//        #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS20
//            uint8_t *dmpPacketBuffer;
//            uint16_t dmpPacketSize;
//
//            uint8_t dmpInitialize();
//            bool dmpPacketAvailable();
//
//            uint8_t dmpSetFIFORate(uint8_t fifoRate);
//            uint8_t dmpGetFIFORate();
//            uint8_t dmpGetSampleStepSizeMS();
//            uint8_t dmpGetSampleFrequency();
//            int32_t dmpDecodeTemperature(int8_t tempReg);
//
//            // Register callbacks after a packet of FIFO data is processed
//            //uint8_t dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
//            //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func func);
//            uint8_t dmpRunFIFORateProcesses();
//
//            // Setup FIFO for various output
//            uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
//            uint8_t dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
//            uint8_t dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);
//
//            // Get Fixed Point data from FIFO
//            uint8_t dmpGetAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternion(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternion(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t* packet=0);
//            uint8_t dmpGet6AxisQuaternion(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGet6AxisQuaternion(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGet6AxisQuaternion(Quaternion *q, const uint8_t* packet=0);
//            uint8_t dmpGetRelativeQuaternion(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetRelativeQuaternion(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetRelativeQuaternion(Quaternion *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyro(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyro(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpSetLinearAccelFilterCoefficient(float coef);
//            uint8_t dmpGetLinearAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw, VectorFloat *gravity);
//            uint8_t dmpGetLinearAccelInWorld(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccelInWorld(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal, Quaternion *q);
//            uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g, VectorInt16 *a, const uint8_t* packet=0);
//            uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroSensor(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetControlData(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetTemperature(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
//            uint8_t dmpGetUnquantizedAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetUnquantizedAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetQuantizedAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuantizedAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetExternalSensorData(int32_t *data, uint16_t size, const uint8_t* packet=0);
//            uint8_t dmpGetEIS(int32_t *data, const uint8_t* packet=0);
//
//            uint8_t dmpGetEuler(float *data, Quaternion *q);
//            uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat *gravity);
//
//            // Get Floating Point data from FIFO
//            uint8_t dmpGetAccelFloat(float *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternionFloat(float *data, const uint8_t* packet=0);
//
//            uint8_t dmpProcessFIFOPacket(const unsigned char *dmpData);
//            uint8_t dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t *processed=NULL);
//
//            uint8_t dmpSetFIFOProcessedCallback(void (*func) (void));
//
//            uint8_t dmpInitFIFOParam();
//            uint8_t dmpCloseFIFO();
//            uint8_t dmpSetGyroDataSource(uint8_t source);
//            uint8_t dmpDecodeQuantizedAccel();
//            uint32_t dmpGetGyroSumOfSquare();
//            uint32_t dmpGetAccelSumOfSquare();
//            void dmpOverrideQuaternion(long *q);
//            uint16_t dmpGetFIFOPacketSize();
//
//        #endif
//
//        // special methods for MotionApps 4.1 implementation
//        #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS41
//            uint8_t *dmpPacketBuffer;
//            uint16_t dmpPacketSize;
//
//            uint8_t dmpInitialize();
//            bool dmpPacketAvailable();
//
//            uint8_t dmpSetFIFORate(uint8_t fifoRate);
//            uint8_t dmpGetFIFORate();
//            uint8_t dmpGetSampleStepSizeMS();
//            uint8_t dmpGetSampleFrequency();
//            int32_t dmpDecodeTemperature(int8_t tempReg);
//
//            // Register callbacks after a packet of FIFO data is processed
//            //uint8_t dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
//            //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func func);
//            uint8_t dmpRunFIFORateProcesses();
//
//            // Setup FIFO for various output
//            uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
//            uint8_t dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
//            uint8_t dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t accuracy);
//            uint8_t dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);
//
//            // Get Fixed Point data from FIFO
//            uint8_t dmpGetAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternion(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternion(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t* packet=0);
//            uint8_t dmpGet6AxisQuaternion(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGet6AxisQuaternion(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGet6AxisQuaternion(Quaternion *q, const uint8_t* packet=0);
//            uint8_t dmpGetRelativeQuaternion(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetRelativeQuaternion(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetRelativeQuaternion(Quaternion *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyro(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyro(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetMag(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpSetLinearAccelFilterCoefficient(float coef);
//            uint8_t dmpGetLinearAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw, VectorFloat *gravity);
//            uint8_t dmpGetLinearAccelInWorld(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccelInWorld(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal, Quaternion *q);
//            uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g, VectorInt16 *a, const uint8_t* packet=0);
//            uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGyroSensor(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetControlData(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetTemperature(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
//            uint8_t dmpGetUnquantizedAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetUnquantizedAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetQuantizedAccel(int32_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuantizedAccel(int16_t *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
//            uint8_t dmpGetExternalSensorData(int32_t *data, uint16_t size, const uint8_t* packet=0);
//            uint8_t dmpGetEIS(int32_t *data, const uint8_t* packet=0);
//
//            uint8_t dmpGetEuler(float *data, Quaternion *q);
//            uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat *gravity);
//
//            // Get Floating Point data from FIFO
//            uint8_t dmpGetAccelFloat(float *data, const uint8_t* packet=0);
//            uint8_t dmpGetQuaternionFloat(float *data, const uint8_t* packet=0);
//
//            uint8_t dmpProcessFIFOPacket(const unsigned char *dmpData);
//            uint8_t dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t *processed=NULL);
//
//            uint8_t dmpSetFIFOProcessedCallback(void (*func) (void));
//
//            uint8_t dmpInitFIFOParam();
//            uint8_t dmpCloseFIFO();
//            uint8_t dmpSetGyroDataSource(uint8_t source);
//            uint8_t dmpDecodeQuantizedAccel();
//            uint32_t dmpGetGyroSumOfSquare();
//            uint32_t dmpGetAccelSumOfSquare();
//            void dmpOverrideQuaternion(long *q);
//            uint16_t dmpGetFIFOPacketSize();
//        #endif
//
//    private:
//        uint8_t devAddr;
//        uint8_t buffer[14];
//};
//
//#endif /* _MPU6050_H_ */
```

5.9 GyroscopeMPU9250.cpp File Reference

`#include "GyroscopeMPU9250.h"`

Include dependency graph for GyroscopeMPU9250.cpp:

5.10 GyroscopeMPU9250.cpp

```
#include "GyroscopeMPU9250.h"

GyroscopeMPU9250::GyroscopeMPU9250(bool ad0)
        : RegisterBasedWiredDevice(MPU9250_ADDRESS | (ad0 & 0x01)) {
    config.value = 0x00;
}

float GyroscopeMPU9250::getRotationX() {
    return readAxisRotation(GYRO_XOUT_H);
}
```

```

float GyroscopeMPU9250::getRotationY() {
    return readAxisRotation(GYRO_YOUT_H);
}

float GyroscopeMPU9250::getRotationZ() {
    return readAxisRotation(GYRO_ZOUT_H);
}

unsigned char GyroscopeMPU9250::readXYZ(unsigned char *buf) {
    return readRegisterBlock(GYRO_XOUT_H, buf, 6);
}

float GyroscopeMPU9250::readAxisRotation(unsigned char axisRegister) {
    unsigned char buf[2];
    if (readRegisterBlock(axisRegister, buf, 2) != 2) {
        return 0.0;
    }
    return convertToDegreePerSeconds(buf);
}

void GyroscopeMPU9250::setFullScaleRange(FullScaleRange fsr) {
    configureRegisterBits(GYRO_CONFIG, GYRO_CONFIG_GYRO_FS_SEL, (unsigned char) fsr);
    config.GYRO_FS_SEL = fsr >> 3;
}

void GyroscopeMPU9250::selectClock(ClockSelection cs) {
    configureRegisterBits(PWR_MGMT_1, PWR_MGMT_1_CLKSEL, (unsigned char) cs);
}

void GyroscopeMPU9250::reset() {
    configureRegisterBits(PWR_MGMT_1, PWR_MGMT_1_H_RESET, 0xff);
}

void GyroscopeMPU9250::sleep() {
    configureRegisterBits(PWR_MGMT_1, PWR_MGMT_1_SLEEP, 0xff);
}

void GyroscopeMPU9250::awake() {
    configureRegisterBits(PWR_MGMT_1, PWR_MGMT_1_SLEEP, 0x00);
}

void GyroscopeMPU9250::enableAxis(Axis axis) {
    configureRegisterBits(PWR_MGMT_2, PWR_MGMT_2_DISABLE_G, ~axis);
}

float GyroscopeMPU9250::convertToDegreePerSeconds(unsigned char buf[2]) {
    int raw = (buf[0] << 8) | buf[1];
    float counts[] = { 131.0, 65.5, 32.8, 16.4 };
    return raw / counts[config.GYRO_FS_SEL];
}
```

5.11 GyroscopeMPU9250.h File Reference

```
#include <Gyroscope.h>
#include <RegisterBasedWiredDevice.h>
```

Include dependency graph for GyroscopeMPU9250.h: This graph shows which files directly or indirectly include this file:

Classes

- class GyroscopeMPU9250
- union GyroscopeMPU9250::GYRO_CONFIGbits
- union GyroscopeMPU9250::PWR_MGMT_1bits
- union GyroscopeMPU9250::PWR_MGMT_2bits

Macros

- #define MPU9250_ADDRESS 0x68

5.11.1 Macro Definition Documentation

5.11.1.1 #define MPU9250_ADDRESS 0x68

Arduino - Gyroscope Driver.

Implementation for MPU9250.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 15 of file GyroscopeMPU9250.h.

5.12 GyroscopeMPU9250.h

```
00001
00009 #ifndef __ARDUINO_DRIVER_GYROSCOPE_MPU9250_H__
00010 #define __ARDUINO_DRIVER_GYROSCOPE_MPU9250_H__ 1
00011
00012 #include <Gyroscope.h>
00013 #include <RegisterBasedWiredDevice.h>
00014
00015 #define MPU9250_ADDRESS 0x68
00016
00027 class GyroscopeMPU9250 : public Gyroscope, public RegisterBasedWiredDevice {
00028
00029 public:
00030
00031     enum Register {
00032         PWR_MGMT_1 = 0x6b,
00033         PWR_MGMT_2 = 0x6c,
00034         GYRO_CONFIG = 0x1b,
00035         GYRO_XOUT_H = 0x43,
00036         GYRO_XOUT_L = 0x44,
00037         GYRO_YOUT_H = 0x45,
00038         GYRO_YOUT_L = 0x46,
00039         GYRO_ZOUT_H = 0x47,
00040         GYRO_ZOUT_L = 0x48,
00041     };
00042
00054     enum FullScaleRange {
00055         FS_SEL_250DPS = 0x00,
00056         FS_SEL_500DPS = 0x08,
00057         FS_SEL_1000DPS = 0x10,
00058         FS_SEL_2000DPS = 0x18
00059     };
00060
00061     enum ClockSelection {
00062         INTERNAL_20MHZ_OSCILLATOR = 0x00,
00063         BEST_AVAILABLE_SOURCE = 0x01,
00064         STOPS_CLOCK_KEEPS_TIMING = 0x07,
00065     };
00066
00070     enum Axis {
00071         AXIS_NONE = 0x00,
00072         AXIS_X = 0x04,
00073         AXIS_Y = 0x02,
00074         AXIS_Z = 0x01,
00075         AXIS_XY = AXIS_X | AXIS_Y,
00076         AXIS_XZ = AXIS_X | AXIS_Z,
00077         AXIS_YZ = AXIS_Y | AXIS_Z,
00078         AXIS_XYZ = AXIS_X | AXIS_Y | AXIS_Z
00079     };
00080
00084     enum Mask {
00085         GYRO_CONFIG_GYRO_FS_SEL = 0x18,
00086         PWR_MGMT_2_DISABLE_G = 0x07,
00087         PWR_MGMT_1_H_RESET = 0x80,
00088         PWR_MGMT_1_SLEEP = 0x40,
00089         PWR_MGMT_1_CYCLE = 0x20,
00090         PWR_MGMT_1_GYRO_STANDBY = 0x10,
00091         PWR_MGMT_1_CLKSEL = 0x07
00092     };
```

```

    union GYRO_CONFIGbits {
        struct {
            unsigned char FCHOICE_B :2;
            unsigned char :1;
            unsigned char GYRO_FS_SEL :2;
            unsigned char ZGYRO_CTEN :1;
            unsigned char YGYRO_CTEN :1;
            unsigned char XGYRO_Cten :1;
        };
        unsigned char value;
    };

    union PWR_MGMT_1bits {
        struct {
            unsigned char CLKSEL :3;
            unsigned char PD_PTAT :1;
            unsigned char GYRO_STANDBY :1;
            unsigned char CYCLE :1;
            unsigned char SLEEP :1;
            unsigned char H_RESET :1;
        };
        unsigned char value;
    };

    union PWR_MGMT_2bits {
        struct {
            unsigned char DISABLE_ZG :1;
            unsigned char DISABLE_YG :1;
            unsigned char DISABLE_XG :1;
            unsigned char DISABLE_ZA :1;
            unsigned char DISABLE_YA :1;
            unsigned char DISABLE_XA :1;
            unsigned char :2;
        };
        struct {
            unsigned char DISABLE_G :3;
            unsigned char DISABLE_A :3;
            unsigned char :2;
        };
        unsigned char value;
    };

    GyroscopeMPU9250(bool ad0);

    float getRotationX();

    float getRotationY();

    float getRotationZ();

    unsigned char readXYZ(unsigned char *buf);

    float readAxisRotation(unsigned char axisRegister);

    void setFullScaleRange(FullScaleRange fsr);

    void selectClock(ClockSelection cs);

    void reset();

    void sleep();

    void awake();

    void enableAxis(Axis axis);

    float convertToDegreePerSeconds(unsigned char buf[2]);

private:

    GYRO_CONFIGbits config;
};

#endif /* __ARDUINO_DRIVER_GYROSCOPE_MPU9250_H__ */
```

Index