MPU6050 Driver

1.0

Generated by Doxygen 1.9.5

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 MPU_Advanced.h File Reference

For changing the device's configuration.

```
#include "MPU_Shared.h"
```

Enumerations

- enum clockSource { intOsc , gyroX , gyroY , gyroZ }

  An enum for choosing the clock source.

Functions

- char setClockSource (enum clockSource CS)

  This function is used to set a clock source, by default the internal oscillator is used.
- char resetAccSignalPath ()

  This function will revert the ADC and DLPF settings for the accelerometers to their default values.
- char resetGyroSignalPath ()

  This function will revert the ADC and DLPF settings for the gyroscopes to their default values.
- char resetTempSignalPath ()

  This function will revert the ADC and DLPF settings for the temperature sensor to their default values.
- char dataRDY_IE ()

  This function will enable the data ready interrupt which is set everytime new readings are written to the sensor registers after passing by the ADC and the DLPF.
- char setDLPF (char DLPF_CFG)

  This function is used to configure the digital low-pass filter (DLPF) for both the gyroscope and accelerometer readings.
- char setSampleRate (int sampleRate)

  This function sets the sample rate divider of the MPU6050 according to the desired sample rate and the current DLPF settings.

2.1.1 Detailed Description

For changing the device's configuration.

This file includes all the function prototypes and macros for changing the configuration of the MPU6050 such as clock source, sample rate, enable data ready interrupt, etc....

2.1.2 Enumeration Type Documentation

2.1.2.1 clockSource

```
enum clockSource
```

An enum for choosing the clock source.

Chooses between internal oscillator or one of the gyros.

Enumerator

intOsc	Internal oscillator.
gyroX	X-axis gyroscope.
gyroY	Y-axis gyroscope.
gyroZ	Z-axis gyroscope.

2.1.3 Function Documentation

2.1.3.1 setClockSource()

```
char setClockSource (
            enum clockSource CS )
```

This function is used to set a clock source, by default the internal oscillator is used.

Parameters

CS	The enum of type clockSource used to set the clock source of the MPU6050.

Attention

It is recommended by the manufacturer to use one of the gyros for improved clock stability.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_AdvancedExample.cpp.

2.1.3.2 resetAccSignalPath()

```
char resetAccSignalPath ( )
```

This function will revert the ADC and DLPF settings for the accelerometers to their default values.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.1.3.3 resetGyroSignalPath()

```
char resetGyroSignalPath ( )
```

This function will revert the ADC and DLPF settings for the gyroscopes to their default values.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.1.3.4 resetTempSignalPath()

```
char resetTempSignalPath ( )
```

This function will revert the ADC and DLPF settings for the temperature sensor to their default values.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.1.3.5 dataRDY_IE()

```
char dataRDY_IE ( )
```

This function will enable the data ready interrupt which is set everytime new readings are written to the sensor registers after passing by the ADC and the DLPF.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.1.3.6 setDLPF()

```
char setDLPF (
            char DLPF_CFG )
```

This function is used to configure the digital low-pass filter (DLPF) for both the gyroscope and accelerometer readings.

Parameters

DLPF_CFG	This is the value written to the register and ranges from 0 to 6, the exact bandwidth and delay associated with each value can found in page 13 of the MPU6050's register map document.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_AdvancedExample.cpp.

2.1.3.7 setSampleRate()

```
char setSampleRate (
            int sampleRate )
```

This function sets the sample rate divider of the MPU6050 according to the desired sample rate and the current DLPF settings.

Parameters

sampleRate	The desired sampling rate.

Attention

It's advised to recall this function everytime setDLPF(char DLPF_CFG) is called because that could indirectly change the sampling rate.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_AdvancedExample.cpp.

2.2 MPU_Advanced.h

Go to the documentation of this file.

```
1
7 #ifndef ADVANCED_H
8 #define ADVANCED_H
9
10 #include "MPU_Shared.h"
11
15 enum clockSource
16 {
17     intOsc,
18     gyroX,
19     gyroY,
20     gyroZ
21 };
22
29 char setClockSource(enum clockSource CS);
30
35 char resetAccSignalPath();
36
41 char resetGyroSignalPath();
42
47 char resetTempSignalPath();
48
54 char dataRDY_IE();
55
63 char setDLPF(char DLPF_CFG);
64
73 char setSampleRate(int sampleRate);
74
79 #endif
```

2.3 MPU_Basic.h File Reference

For basic operation of the MPU6050.

```
#include "MPU_Shared.h"
```

Functions

- char wakeUp ()

  This function writes to MPU6050's power management register in order to get the device out of its default state at startup which is sleep mode.
- char deviceReset ()

  This function will reset all the MPU6050's registers to their default values.
- char setAccFS_Range (char FS_SELECT)

This function sets full scale range for the 3 accelerometers.

- char setGyroFS_Range (char FS_SELECT)

  This function sets full scale range for the 3 gyroscopes.
- char getAccReadings (int *data, char skipFromBeginning, char skipFromEnd)

  This function retrieves the raw accelerometer readings from the MPU6050.
- char getGyroReadings (int *data, char skipFromBeginning, char skipFromEnd)

  This function retrieves the raw gyroscope readings from the MPU6050.
- char getTempReadings (int *data)

  This function retrieves the raw temperature sensor reading from the MPU6050.

2.3.1 Detailed Description

For basic operation of the MPU6050.

This file includes all the prototypes for the basic functions of the MPU6050.

2.3.2 Function Documentation

2.3.2.1 wakeUp()

```
char wakeUp ( )
```

This function writes to MPU6050's power management register in order to get the device out of its default state at startup which is sleep mode.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_AdvancedExample.cpp, and MPU_SelfTestExample.cpp.

2.3.2.2 deviceReset()

```
char deviceReset ( )
```

This function will reset all the MPU6050's registers to their default values.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_AdvancedExample.cpp, and MPU_SelfTestExample.cpp.

2.3.2.3 setAccFS_Range()

```
char setAccFS_Range (
            char FS_SELECT )
```

This function sets full scale range for the 3 accelerometers.

Obviously as the range goes up, the sensor's resolution becomes worse.

Parameters

FS_SELECT	This parameter chooses the FS range and should be chosen as follows: • 1 : +- 4g • 2 : +- 8g • 3 : +- 16g • all other values : +- 2g

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_SelfTestExample.cpp.

2.3.2.4 setGyroFS_Range()

```
char setGyroFS_Range (
            char FS_SELECT )
```

This function sets full scale range for the 3 gyroscopes.

Obviously as the range goes up, the sensor's resolution becomes worse.

Parameters

FS_SELECT	This parameter chooses the FS range and should be chosen as follows: • 1 : +- 500dps (83.33rpm) • 2 : +- 1000dps (166.66rpm) • 3 : +- 2000dps (333.33 rpm) • all other values : +- 250dps (41.66rpm)

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_SelfTestExample.cpp.

2.3.2.5 getAccReadings()

```
char getAccReadings (
             int * data,
             char skipFromBeginning,
             char skipFromEnd )
```

This function retrieves the raw accelerometer readings from the MPU6050.

Parameters

data	A pointer to the start of a signed int array that will hold the values read from the IMU.
skipFromBeginning	This parameter can be used to retrieve only specific accelerometer readings and not all 3.
skipFromEnd	This parameter can be used to retrieve only specific accelerometer readings and not all 3.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_BasicExamples.cpp.

2.3.2.6 getGyroReadings()

```
char getGyroReadings (
             int * data,
             char skipFromBeginning,
             char skipFromEnd )
```

This function retrieves the raw gyroscope readings from the MPU6050.

Parameters

data	A pointer to the start of a signed int array that will hold the values read from the IMU.
skipFromBeginning	This parameter can be used to retrieve only specific gyroscope readings and not all 3.
skipFromEnd	This parameter can be used to retrieve only specific gyroscope readings and not all 3.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_BasicExamples.cpp, and MPU_SelfTestExample.cpp.

2.3.2.7 getTempReadings()

```
char getTempReadings (
             int * data )
```

This function retrieves the raw temperature sensor reading from the MPU6050.

Parameters

data	A pointer to a signed int that will hold the value read from the IMU.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_BasicExamples.cpp.

2.4 MPU_Basic.h

Go to the documentation of this file.

```
1
6 #ifndef BASIC_H
7 #define BASIC_H
8
9 #include "MPU_Shared.h"
10
16 char wakeUp();
17
22 char deviceReset();
23
34 char setAccFS_Range(char FS_SELECT);
35
46 char setGyroFS_Range(char FS_SELECT);
47
55 char getAccReadings(int* data, char skipFromBeginning, char skipFromEnd);
56
64 char getGyroReadings(int* data, char skipFromBeginning, char skipFromEnd);
65
71 char getTempReadings(int* data);
72
79 #endif
```

2.5 MPU_FIFO.h File Reference

For using the internal FIFO on the MPU6050.

```
#include "MPU_Shared.h"
```

Functions

- char FIFO_Reset ()

  This function resets the FIFO which will clear the FIFO buffer.
- char FIFO_Enable ()

  This function enables the FIFO, this needs to be called before enabling the FIFO for specific sensors (gyro, acc, temp).
- char FIFO_Disable ()

  This function disables the FIFO.
- char readFIFO_count (unsigned int *count)

  This function retreives the no of samples currently in the FIFO buffer (in bytes) that are available to be read.
- char readFIFO (char *value)

  This function reads the first byte that was written to the buffer.
- char FIFO_OVF_IE ()

  This function enables the FIFO overflow interrupt.
- char gyroFIFO_En ()

  This function enables the FIFO for the gyroscope outputs such that their values will be copied into the FIFO.
- char accFIFO_En ()

  This function enables the FIFO for the accelerometer outputs such that their values will be copied into the FIFO.
- char tempFIFO_EN ()

  This function enables the FIFO for the temperature sensor output such that its value will be copied into the FIFO.

2.5.1 Detailed Description

For using the internal FIFO on the MPU6050.

This file includes all the needed function prototypes macros to be able to use the internal FIFO for storing and readings the sensors' output values.

Attention

The size of the FIFO is 1024 bytes.

2.5.2 Function Documentation

2.5.2.1 FIFO_Reset()

```
char FIFO_Reset ( )
```

This function resets the FIFO which will clear the FIFO buffer.

Attention

The register automatically clears the bit to 0 after execution therefore there is no need to clear it manually.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.2 FIFO_Enable()

```
char FIFO_Enable ( )
```

This function enables the FIFO, this needs to be called before enabling the FIFO for specific sensors (gyro, acc, temp).

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.3 FIFO_Disable()

```
char FIFO_Disable ( )
```

This function disables the FIFO.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.4 readFIFO_count()

```
char readFIFO_count (
            unsigned int * count )
```

This function retreives the no of samples currently in the FIFO buffer (in bytes) that are available to be read.

This should be checked before reading from the FIFO buffer to insure that it's not empty.

Parameters

count	A pointer to an unsigned int that will store the current FIFO count.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.5 readFIFO()

```
char readFIFO (
            char * value )
```

This function reads the first byte that was written to the buffer.

Attention

The sensor values are written to the buffer in order from the lowest register address (59: AccX High) first to the highest register address (72: GyroZ Low).

Parameters

value	A pointer to a signed char that will store the value read from the buffer.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.6 FIFO_OVF_IE()

```
char FIFO_OVF_IE ( )
```

This function enables the FIFO overflow interrupt.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.7 gyroFIFO_En()

```
char gyroFIFO_En ( )
```

This function enables the FIFO for the gyroscope outputs such that their values will be copied into the FIFO.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.8 accFIFO_En()

```
char accFIFO_En ( )
```

This function enables the FIFO for the accelerometer outputs such that their values will be copied into the FIFO.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.5.2.9 tempFIFO_EN()

```
char tempFIFO_EN ( )
```

This function enables the FIFO for the temperature sensor output such that its value will be copied into the FIFO.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.6 MPU_FIFO.h

Go to the documentation of this file.

```
1
8 #ifndef FIFO_H
9 #define FIFO_H
10
11 #include "MPU_Shared.h"
12
19 char FIFO_Reset();
20
25 char FIFO_Enable();
26
31 char FIFO_Disable();
32
39 char readFIFO_count(unsigned int* count);
40
48 char readFIFO(char* value);
49
54 char FIFO_OVF_IE();
55
60 char gyroFIFO_En();
61
66 char accFIFO_En();
67
72 char tempFIFO_EN();
73
74 #endif
```

2.7 MPU_SelfTest.h File Reference

For testing the relaibility of the IMU's readings.

```
#include "MPU_Shared.h"
#include <math.h>
```

Functions

- char accSelfTestEn ()

  This function enables the self-test feature for the accelerometers which uses the on-board electronics to actuate the on-board sensors as well as setting the appropriate full-scale range for testing.
- char gyroSelfTestEn ()

  This function enables the self-test feature for the gyroscopes which uses the on-board electronics to actuate the on-board sensors as well as setting the appropriate full-scale range for testing.
- char getFactoryTrimGyro (int *data)

  This function will retrieve the factory trim values for the gyroscopes.
- char getFactoryTrimAcc (int *data)

  This function will retrieve the factory trim values for the accelerometers.

2.7.1 Detailed Description

For testing the relaibility of the IMU's readings.

This file includes all the function prototypes and macros for using the MPU6050's self-test feature which can be used to insure that the device can still deliver relaible readings.

Attention

According to the manufacturers's datasheet, the FS range for testing the gyros and accelerometers should be +-250 dps and +-8g respectively.

The Change from the factory trim is calculated as follows:

$$\text{Change from the factory trim} = \frac{STR - FT}{FT}$$

Where STR is the self-test response and equals to: $STR = (\text{Output with self test enabled} - \text{Output with self test disabled})$

The acceptable change is within +- 2%.

2.7.2 Function Documentation

2.7.2.1 accSelfTestEn()

```
char accSelfTestEn ( )
```

This function enables the self-test feature for the accelerometers which uses the on-board electronics to actuate the on-board sensors as well as setting the appropriate full-scale range for testing.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.7.2.2 gyroSelfTestEn()

```
char gyroSelfTestEn ( )
```

This function enables the self-test feature for the gyroscopes which uses the on-board electronics to actuate the on-board sensors as well as setting the appropriate full-scale range for testing.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_SelfTestExample.cpp.

2.7.2.3 getFactoryTrimGyro()

```
char getFactoryTrimGyro (
            int * data )
```

This function will retrieve the factory trim values for the gyroscopes.

Parameters

data	A pointer to the start of an array of signed int that will store the FTs for each gyroscope.

Returns

Will return 1 upon successful completion or 0 otherwise.

Examples

MPU_SelfTestExample.cpp.

2.7.2.4 getFactoryTrimAcc()

```
char getFactoryTrimAcc (
            int * data )
```

This function will retrieve the factory trim values for the accelerometers.

Parameters

data	A pointer to the start of an array of signed int that will store the FTs for each accelerometer.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.8 MPU_SelfTest.h

Go to the documentation of this file.

```
1
17 #ifndef SELF_TEST_H
18 #define SELF_TEST_H
19
20 #include "MPU_Shared.h"
21 #include <math.h>
22
28 char accSelfTestEn();
29
35 char gyroSelfTestEn();
36
42 char getFactoryTrimGyro(int* data);
43
49 char getFactoryTrimAcc(int* data);
50
56 #endif
```

2.9 MPU_Shared.h File Reference

Shared functions and macros.

```
#include "I2C_1.h"
```

Macros

- #define **MPU6050_ADDRESS** 0x34

  The 6 bit I2C address of the MPU6050 which excludes AD0.

Functions

- void setAD0 (char pinValue)

  This function sets the value used for the AD0 bit in the MPU6050's I2C 7 bit address and its value should match that of the pin i.e.
- char writeToReg (char regAddr, char data)

  This function writes a byte into a register on the MPU6050.
- char readFromReg (char regAddr, char ∗data, char n)

  This function will read from a register/s on the MPU6050.
- char readInterruptStatus (char ∗value)

  This function reads the status of all 3 possible interrupt sources (FIFO OVF, I2C Master INT, Data RDY INT) on the MPU6050 to determine the source which triggered the interrupt.
- char disableInterrupts ()

  This function disables the all interrupts on the MPU6050.

2.9.1 Detailed Description

Shared functions and macros.

This file includes all the function prototypes, macros, and needed header files for the MPU_Shared.c which has the definitions of functions used by all the other source files of the MPU6050 library. It does not need to be included as all other files include it.

2.9.2 Function Documentation

2.9.2.1 setAD0()

```
void setAD0 (
            char pinValue )
```

This function sets the value used for the AD0 bit in the MPU6050's I2C 7 bit address and its value should match that of the pin i.e.

1 (High) or 0 (Low).

Parameters

pinValue	The value of the AD0 pin on the MPU6050 IC. Note: all values other than 0 will be taken as 1.

Examples

MPU_AdvancedExample.cpp, and MPU_SelfTestExample.cpp.

2.9.2.2 writeToReg()

```
char writeToReg (
            char regAddr,
            char data )
```

This function writes a byte into a register on the MPU6050.

Parameters

regAddr	The address of the register in the MPU6050's register map.
data	The value that will be written to that register.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.9.2.3 readFromReg()

```
char readFromReg (
            char regAddr,
            char * data,
            char n )
```

This function will read from a register/s on the MPU6050.

Parameters

regAddr	The address of the first register to read from in the MPU6050's register map.
data	A pointer to the start of an array of bytes that the read data will be written to.
n	The number of registers to read from.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.9.2.4 readInterruptStatus()

```
char readInterruptStatus (
            char * value )
```

This function reads the status of all 3 possible interrupt sources (FIFO OVF, I2C Master INT, Data RDY INT) on the MPU6050 to determine the source which triggered the interrupt.

Parameters

value	pointer to char that will hold the read value.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.9.2.5 disableInterrupts()

```
char disableInterrupts ( )
```

This function disables the all interrupts on the MPU6050.

Returns

Will return 1 upon successful completion or 0 otherwise.

2.10 MPU_Shared.h

Go to the documentation of this file.

```
1
9 #ifndef SHARED_H
10 #define SHARED_H
11
12 #include "I2C_1.h" /* Driver for the atmega32 I2C peripheral.  */
13 #define MPU6050_ADDRESS 0x34
20 void setAD0(char pinValue);
21
28 static char initiateTransmissionWrite();
29
36 static char initiateTransmissionRead();
37
44 char writeToReg(char regAddr, char data);
45
53 char readFromReg(char regAddr, char* data, char n);
54
61 char readInterruptStatus(char* value);
62
67 char disableInterrupts();
68
69 #endif
```

Chapter 3

Example Documentation

3.1 MPU_BasicExamples.cpp

This is an example on how to use the functions, from the MPU_Basic.h file, for retrieving the on-board sensor readings.

This is an example on how to use the functions, from the MPU_Basic.h file, for retrieving the on-board sensor readings.You will find examples on reading from all 3 sensors (X,Y,Z) or only specific ones as well as retrieving the temperature sensor readings.

```
#include "MPU_Basic.h"
int main(void)
{
    /* Read all 3 accelerometers raw values */
    int accReadings[3];
    char status = getAccReadings(accReadings, 0, 0);
    if(status)
    {
        /* Operation was successful and accReadings now contains all 3 raw readings ordered from AccX at
      index 0 to AccZ at index 2.  */
    }
    else
    {
        /* Operation failed and appropriate action should be taken.  */
    }

    /* Read raw reading for the X-axis accelerometer only */
    int accX;
    status = getAccReadings(&accX, 0, 2);
    if(status)
    {
        /* Operation was successful and the value of the X-axis accelerometer has been loaded into the accX
      variable */
    }
    else
    {
        /* Operation failed and appropriate action should be taken.  */
    }


    /* Read raw reading for the Y-axis gyroscope only */
    int gyroY;
    status = getGyroReadings(&gyroY, 1, 1);
    if(status)
    {
        /* Operation was successful and the value of the Y-axis gyroscope has been loaded into the gyroY
      variable */
    }
    else
    {
        /* Operation failed and appropriate action should be taken.  */
    }


    /* Read raw reading for the temperature sensor */
    int temp;
    status = getTempReadings(&temp);
```

```
    if(status)
    {
        /* Operation was successful and the value of the temperature sensor has been loaded into the temp
      variable */
    }
    else
    {
        /* Operation failed and appropriate action should be taken.  */
    }

    /* Note:  The status checks are advised but obviously not mandatory at every step.  */
```

3.2 MPU_SelfTestExample.cpp

This examples shows how to use the MPU_SelfTest.h functions to calculate the percent change from the factory trim for the 3 gyroscopes.

This examples shows how to use the MPU_SelfTest.h functions to calculate the percent change from the factory trim for the 3 gyroscopes. The exact same procedure would be done for the accelerometers.

```
#include "MPU_Basic.h" /* For readings data from the IMU */
#include "MPU_SelfTest.h"
int main(void)
{
    /* Testing the 3 gyroscopes */

    int gyroReadingsRaw[3] = {0, 0, 0};         // Readings with self-test disabled.
    int gyroReadingsSelfTest[3] = {0, 0, 0};    // Readings with self-test enabled.
    int gyroFactoryTrim[3] = {0, 0, 0};         // Factory trim for each gyro.

    I2CInit(); // Initialize I2C peripheral.

    setAD0(0);
    deviceReset();                              // To erase any previous configurations.
    // It's advised to create some delay between reseting the IMU and waking it back up.

    wakeUp();                                    // Wake up from sleep mode.
    setGyroFS_Range(0); // +-250dps              // Required gyro FS range for testing.
    setAccFS_Range(2); //  +-8g                   // Required accelerometer FS range for testing.

    getFactoryTrimGyro(gyroFactoryTrim);      // Retrieve the factory trim data for the gyros.
    getGyroReadings(gyroReadingsRaw, 0, 0);      // Get raw readings before enabling the self-test.

    gyroSelfTestEn();                          // Enable the self-test feature for the gyros.
    getGyroReadings(gyroReadingsSelfTest,0,0);  // Get readings with self-test enabled.

    int gyroSelfTestResponse[3];
    for(unsigned char i = 0; i < 3; i++)        // Calculate the self-test response.
    {
        gyroSelfTestResponse[i] = gyroReadingsSelfTest[i] - gyroReadingsRaw[i];
    }

    float changeFromFactoryTrim[3];
    for(unsigned char i = 0; i < 3; i++)        // The % change should be in between +-2% for a functional
      IMU.
    {
        changeFromFactoryTrim[i] = ((float) gyroSelfTestResponse[i] - gyroFactoryTrim[i]) /
      gyroFactoryTrim[i];
        changeFromFactoryTrim[i] = 100 * changeFromFactoryTrim[i]; // Convert to percentage.
    }

    /* It's advised to take many samples while self-test is enabled and while it's disabled and to use
* the mean of these values to calculate the change from the factory trim to have more reliable results.
*/
}
```

3.3 MPU_AdvancedExample.cpp

This example shows how to change the MPU6050's configuration using the functions from MPU_Advanced.h.

This example shows how to change the MPU6050's configuration using the functions from MPU_Advanced.h.

```
#include "MPU_Advanced.h"
```

```
int main(void)
{
    I2CInit();

    setAD0(0);
    deviceReset();                  // To erase any previous configurations.
                                    // Insert some delay here.

    wakeUp();                        // Wake up from sleep mode.

    /* Configuration 1:  DLPF = 2, Sample Rate = 200Hz, Clock Source = Z-axis gyroscope */

    setClockSource(gyroZ);           // Set the clock source as Z-axis gyro.
    setDLPF(2);                      // Gyro Bandwidth = 98Hz and Accelerometer Bandwidth = 94Hz.
    setSampleRate(200);                // Set the sampling rate to 200Hz.

    /* Configuration 2:  DLPF = 0, Sample Rate = 500Hz, Clock Source = Internal Oscillator */

    setClockSource(intOsc);         // Set the clock source as the internal oscillator.
    setDLPF(0);                      // Gyro Bandwidth = 256Hz and Accelerometer Bandwidth = 260Hz.
    setSampleRate(500);                // Set the sampling rate to 500Hz.
}
```

Index