MPU-6050 Accelerometer + Gyro

Introduction

The InvenSense MPU-6050 sensor contains a MEMS accelerometer and a MEMS gyro in a single chip. It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefor it captures the x, y, and z channel at the same time.

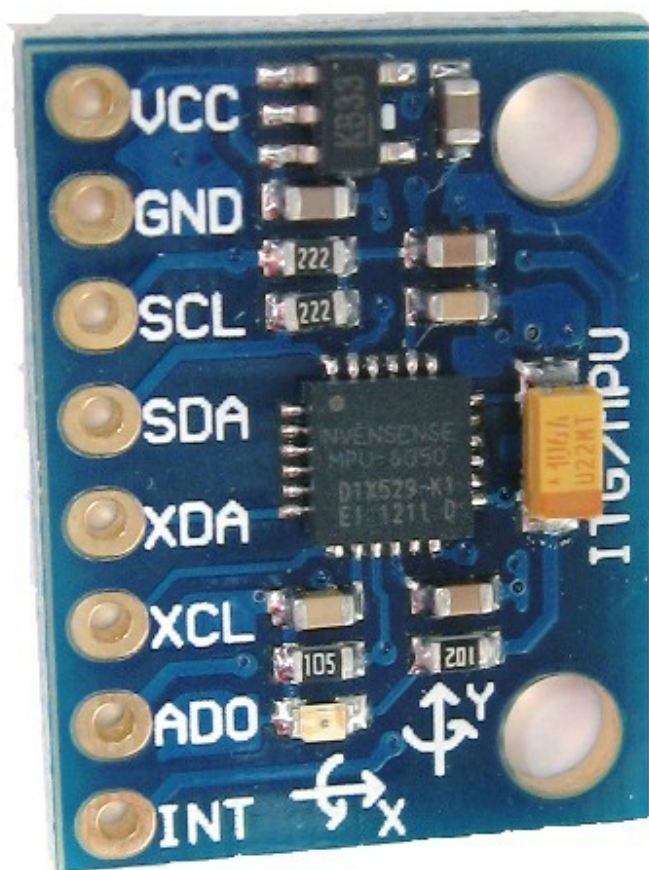The MPU-6050 is not expensive, especially given the fact that it combines both an accelerometer and a gyro.

Photo: GY-521 breakout board

Also note that Invensense has combined the MPU-6050 with a magnetometer (compass) in a single chip called MPU-9150.

Reading raw values is easy, the rest is not

Reading the raw values for the accelerometer and gyro is easy. The sleep mode has to be disabled, and then the registers for the accelerometer and gyro can be read.

But the sensor also contains a 1024 byte FIFO buffer. The sensor values can be programmed to be placed in the FIFO buffer. And the buffer can be read by the Arduino.

The FIFO buffer is used together with the interrupt signal. If the MPU-6050 places data in the FIFO buffer, it signals the Arduino with the interrupt signal so the Arduino knows that there is data in the FIFO buffer waiting to be read.

A little more complicated is the ability to control a second I2C-device.
The MPU-6050 always acts as a slave to the Arduino with the SDA and SCL pins connected to the I2C-bus.
But beside the normal I2C-bus, it has it's own I2C controller to be a master on a second (sub)-I2C-bus. It uses the pins AUX_DA and AUX_CL for that second (sub)-I2C-bus.
It can control, for example, a magnetometer. The values of the magnetometer can be passed on to the Arduino.

Things get really complex with the "DMP".
The sensor has a "Digital Motion Processor" (DMP), also called a "Digital Motion Processing Unit". This DMP can be programmed with firmware and is able to do complex calculations with the sensor values.
For this DMP, InvenSense has a discouragement policy, by not supplying enough information how to program the DMP. However, some have used reverse engineering to capture firmware.

The DMP ("Digital Motion Processor") can do fast calculations directly on the chip. This reduces the load for the microcontroller (like the Arduino). The DMP is even able to do calculations with the sensor values of another chip, for example a magnetometer connected to the second (sub)-I2C-bus.

Code

The accelerometer and gyro values are called the "raw" values. This is just as with other accelerometer and gyro sensors. A more sophisticated application is using the DMP to retrieve specific computed values from the sensor.

The example code on this page is just a simple and basic sketch to read the raw values.

For more serious use of the MPU-6050, Jeff Rowberg has done an excellent job.
See his I2C lib: http://www.i2cdevlib.com/devices/mpu6050
Or the latest code on github: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

The FreeIMU library includes the MPU-6050 code from Jeff Rowberg.
The FreeIMU library: http://www.varesano.net/projects/hardware/FreeIMU

To start with the MPU-6050, see the page of InvenSense: http://www.invensense.com/mems/gyro/mpu6050.html

For other programs and sensors, see the Degrees Of Freedom, 6DOF, 9DOF, 10DOF, 11DOF-section in the Playground index.

GY-521

This sensor board has a voltage regulator. When using 3.3V to the VCC the resulting voltage (after the onboard voltage regulator) might be too low for a good working I2C bus. It is preferred to apply 5V to the VCC pin of the sensor board. The board has pull-up resistors on the I2C-bus. The value of those pull-up resistors are sometimes 10k and sometimes 2k2. The 2k2 is rather low. If it is combined with other sensor board which have also pull-up resistors, the total pull-up impedance might be too low.
This schematic is hard to find, so here is a copy: http://playground.arduino.cc/uploads/Main/MPU6050-V1-SCH.jpg

Flyduino MPU6050 Break Out onboard 3.3V reg

http://flyduino.net/MPU6050-Break-Out-onboard-33V-reg_1
This sensor board contains a voltage regulator, so it can also be used with 5V. The pull-up resistors of the I2C-bus are 4k7. It is actually a GY-52 breakout board.

Flyduino 10DOF IMU GY-86 MPU6050+HMC5883l+MS5611

http://flyduino.net/10DOF-IMU-GY-86-MPU6050-HMC5883l-MS5611_1
A sensor board with the MPU-6050 and a magnetometer and barometer. This schematic is hard to find, so here is a copy: http://4.bp.blogspot.com/-dIgYZJIn7E/UxfwYegZwoI/AAAAAAAAo_8/szYT2_AkNvk/s1600/GY-86-Schematic.jpg. The sensor board contains a voltage regulator, so it can be used with 5V. There seems to be also a level shifter on the board for the I2C-bus. The pull-up resistors for the I2C-bus seems to be 2k2, which is rather low.

Measurements

The raw values raises questions in the forums, since the raw values might seem unstable. Below are the raw values of the sensor that I measured, so you can compare them with your own raw values.

The raw values changes a lot due to a number of reasons. The default sensitivity is high, and the sensor returns 16 bits, but the actual valid number of bits is less than 16 bits. Since they are 16 bits, a variation of 50 is just a very small variation.

The next measurement were done in these conditions:

- The sensor was placed as horizontal as possible.
- It was placed on concreet, not a wooden table.
- During the measurements, there was no traffic in the street.
- An battery of 12V was used, not the less stable voltage from the USB bus. I used a battery instead of an adapter to avoid any mains noise.
- The circuit was on for 15 minutes, to stabalize any temperature influence.
- The room temperature was 25 degrees Celcius.

```
MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 184, -484, 14992
temperature: 29.635 degrees Celsius
gyro x,y,z : 367, 220, -812,

MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 116, -364, 15056
temperature: 29.635 degrees Celsius
gyro x,y,z : 373, 226, -766,

MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 232, -432, 15100
temperature: 29.682 degrees Celsius
gyro x,y,z : 382, 232, -790,

MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 280, -468, 15136
temperature: 29.635 degrees Celsius
gyro x,y,z : 368, 211, -820,
```

```
MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 140, -432, 15108
temperature: 29.588 degrees Celsius
gyro x,y,z : 388, 203, -806,

MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 220, -464, 14920
temperature: 29.541 degrees Celsius
gyro x,y,z : 374, 196, -774,

MPU-6050
Read accel, temp and gyro, error = 0
accel x,y,z: 172, -440, 15100
temperature: 29.588 degrees Celsius
gyro x,y,z : 363, 200, -769,
```

Example sketch

The sketch below is the code made with Arduino 1.0.1 and it uses I2C-bus communication.

The code uses the Arduino functions as much as possible. It is just a simple and basic sketch to get the MPU-6050 working.

The I2C-address depends on the AD0 pin of the sensor. If it is connected to ground, the address is 0x68. If it is connected to VLOGIC (+3.3V) it is 0x69. There are a few sensor boards with the MPU-6050 sensor already soldered on it. Some of those boards have a pull-down resistor at AD0 (address = 0x68), others have a pull-up resistor (address = 0x69).

Search in the sketch for "MPU6050_I2C_ADDRESS" and set that to your own I2C address.
A i2c_scanner can be used to check if the device is connected to the i2c bus.

The acceleration and gyro values of the sketch are raw values, which are not yet compensated for offset. The very first acceleration and gyro values after power up are sometimes not valid.

The sketch is about 7kbyte, and will fit in a ATmega8.

1. // MPU-6050 Accelerometer + Gyro
2. // ----------------------------
3. //
4. // By arduino.cc user "Krodal".
5. //
6. // June 2012
7. //   first version
8. // July 2013
9. //   The 'int' in the union for the x,y,z
10. //   changed into int16_t to be compatible
11. //   with Arduino Due.
12. //
13. // Open Source / Public Domain
14. //
15. // Using Arduino 1.0.1
16. // It will not work with an older version,

```
17. // since Wire.endTransmission() uses a parameter
18. // to hold or release the I2C bus.
19. //
20. // Documentation:
21. // - The InvenSense documents:
22. //   - "MPU-6000 and MPU-6050 Product Specification",
23. //     PS-MPU-6000A.pdf
24. //   - "MPU-6000 and MPU-6050 Register Map and Descriptions",
25. //     RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
26. //   - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
27. //     AN-MPU-6000EVB.pdf
28. //
29. // The accuracy is 16-bits.
30. //
31. // Temperature sensor from -40 to +85 degrees Celsius
32. //   340 per degrees, -512 at 35 degrees.
33. //
34. // At power-up, all registers are zero, except these two:
35. //     Register 0x6B (PWR_MGMT_2) = 0x40  (I read zero).
36. //     Register 0x75 (WHO_AM_I)   = 0x68.
37. //
38.
39. #include <Wire.h>
40.
41.
42. // The name of the sensor is "MPU-6050".
43. // For program code, I omit the '-',
44. // therefor I use the name "MPU6050....".
45.
46.
47. // Register names according to the datasheet.
48. // According to the InvenSense document
49. // "MPU-6000 and MPU-6050 Register Map
50. // and Descriptions Revision 3.2", there are no registers
51. // at 0x02 ... 0x18, but according other information
52. // the registers in that unknown area are for gain
53. // and offsets.
54. //
55. #define MPU6050_AUX_VDDIO       0x01   // R/W
56. #define MPU6050_SMPLRT_DIV      0x19   // R/W
57. #define MPU6050_CONFIG          0x1A   // R/W
58. #define MPU6050_GYRO_CONFIG     0x1B   // R/W
59. #define MPU6050_ACCEL_CONFIG    0x1C   // R/W
60. #define MPU6050_FF_THR          0x1D   // R/W
61. #define MPU6050_FF_DUR          0x1E   // R/W
62. #define MPU6050_MOT_THR         0x1F   // R/W
63. #define MPU6050_MOT_DUR         0x20   // R/W
64. #define MPU6050_ZRMOT_THR       0x21   // R/W
65. #define MPU6050_ZRMOT_DUR       0x22   // R/W
66. #define MPU6050_FIFO_EN         0x23   // R/W
67. #define MPU6050_I2C_MST_CTRL    0x24   // R/W
68. #define MPU6050_I2C_SLV0_ADDR   0x25   // R/W
69. #define MPU6050_I2C_SLV0_REG    0x26   // R/W
70. #define MPU6050_I2C_SLV0_CTRL   0x27   // R/W
```

```
71. #define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
72. #define MPU6050_I2C_SLV1_REG       0x29   // R/W
73. #define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
74. #define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
75. #define MPU6050_I2C_SLV2_REG       0x2C   // R/W
76. #define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
77. #define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
78. #define MPU6050_I2C_SLV3_REG       0x2F   // R/W
79. #define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
80. #define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
81. #define MPU6050_I2C_SLV4_REG       0x32   // R/W
82. #define MPU6050_I2C_SLV4_DO        0x33   // R/W
83. #define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
84. #define MPU6050_I2C_SLV4_DI        0x35   // R
85. #define MPU6050_I2C_MST_STATUS     0x36   // R
86. #define MPU6050_INT_PIN_CFG        0x37   // R/W
87. #define MPU6050_INT_ENABLE         0x38   // R/W
88. #define MPU6050_INT_STATUS         0x3A   // R
89. #define MPU6050_ACCEL_XOUT_H       0x3B   // R
90. #define MPU6050_ACCEL_XOUT_L       0x3C   // R
91. #define MPU6050_ACCEL_YOUT_H       0x3D   // R
92. #define MPU6050_ACCEL_YOUT_L       0x3E   // R
93. #define MPU6050_ACCEL_ZOUT_H       0x3F   // R
94. #define MPU6050_ACCEL_ZOUT_L       0x40   // R
95. #define MPU6050_TEMP_OUT_H         0x41   // R
96. #define MPU6050_TEMP_OUT_L         0x42   // R
97. #define MPU6050_GYRO_XOUT_H        0x43   // R
98. #define MPU6050_GYRO_XOUT_L        0x44   // R
99. #define MPU6050_GYRO_YOUT_H        0x45   // R
100.      #define MPU6050_GYRO_YOUT_L        0x46   // R
101.      #define MPU6050_GYRO_ZOUT_H        0x47   // R
102.      #define MPU6050_GYRO_ZOUT_L        0x48   // R
103.      #define MPU6050_EXT_SENS_DATA_00   0x49   // R
104.      #define MPU6050_EXT_SENS_DATA_01   0x4A   // R
105.      #define MPU6050_EXT_SENS_DATA_02   0x4B   // R
106.      #define MPU6050_EXT_SENS_DATA_03   0x4C   // R
107.      #define MPU6050_EXT_SENS_DATA_04   0x4D   // R
108.      #define MPU6050_EXT_SENS_DATA_05   0x4E   // R
109.      #define MPU6050_EXT_SENS_DATA_06   0x4F   // R
110.      #define MPU6050_EXT_SENS_DATA_07   0x50   // R
111.      #define MPU6050_EXT_SENS_DATA_08   0x51   // R
112.      #define MPU6050_EXT_SENS_DATA_09   0x52   // R
113.      #define MPU6050_EXT_SENS_DATA_10   0x53   // R
114.      #define MPU6050_EXT_SENS_DATA_11   0x54   // R
115.      #define MPU6050_EXT_SENS_DATA_12   0x55   // R
116.      #define MPU6050_EXT_SENS_DATA_13   0x56   // R
117.      #define MPU6050_EXT_SENS_DATA_14   0x57   // R
118.      #define MPU6050_EXT_SENS_DATA_15   0x58   // R
119.      #define MPU6050_EXT_SENS_DATA_16   0x59   // R
120.      #define MPU6050_EXT_SENS_DATA_17   0x5A   // R
121.      #define MPU6050_EXT_SENS_DATA_18   0x5B   // R
122.      #define MPU6050_EXT_SENS_DATA_19   0x5C   // R
123.      #define MPU6050_EXT_SENS_DATA_20   0x5D   // R
124.      #define MPU6050_EXT_SENS_DATA_21   0x5E   // R
```

```
125.      #define MPU6050_EXT_SENS_DATA_22  0x5F   // R
126.      #define MPU6050_EXT_SENS_DATA_23  0x60   // R
127.      #define MPU6050_MOT_DETECT_STATUS  0x61   // R
128.      #define MPU6050_I2C_SLV0_DO       0x63   // R/W
129.      #define MPU6050_I2C_SLV1_DO       0x64   // R/W
130.      #define MPU6050_I2C_SLV2_DO       0x65   // R/W
131.      #define MPU6050_I2C_SLV3_DO       0x66   // R/W
132.      #define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
133.      #define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
134.      #define MPU6050_MOT_DETECT_CTRL   0x69   // R/W
135.      #define MPU6050_USER_CTRL         0x6A   // R/W
136.      #define MPU6050_PWR_MGMT_1        0x6B   // R/W
137.      #define MPU6050_PWR_MGMT_2        0x6C   // R/W
138.      #define MPU6050_FIFO_COUNTH       0x72   // R/W
139.      #define MPU6050_FIFO_COUNTL       0x73   // R/W
140.      #define MPU6050_FIFO_R_W          0x74   // R/W
141.      #define MPU6050_WHO_AM_I          0x75   // R
142.
143.
144.      // Defines for the bits, to be able to change
145.      // between bit number and binary definition.
146.      // By using the bit number, programming the sensor
147.      // is like programming the AVR microcontroller.
148.      // But instead of using "(1<<X)", or "_BV(X)",
149.      // the Arduino "bit(X)" is used.
150.      #define MPU6050_D0 0
151.      #define MPU6050_D1 1
152.      #define MPU6050_D2 2
153.      #define MPU6050_D3 3
154.      #define MPU6050_D4 4
155.      #define MPU6050_D5 5
156.      #define MPU6050_D6 6
157.      #define MPU6050_D7 7
158.
159.      // AUX_VDDIO Register
160.      #define MPU6050_AUX_VDDIO MPU6050_D7  // I2C high: 1=VDD, 0=VLOGIC
161.
162.      // CONFIG Register
163.      // DLPF is Digital Low Pass Filter for both gyro and accelerometers.
164.      // These are the names for the bits.
165.      // Use these only with the bit() macro.
166.      #define MPU6050_DLPF_CFG0    MPU6050_D0
167.      #define MPU6050_DLPF_CFG1    MPU6050_D1
168.      #define MPU6050_DLPF_CFG2    MPU6050_D2
169.      #define MPU6050_EXT_SYNC_SET0 MPU6050_D3
170.      #define MPU6050_EXT_SYNC_SET1 MPU6050_D4
171.      #define MPU6050_EXT_SYNC_SET2 MPU6050_D5
172.
173.      // Combined definitions for the EXT_SYNC_SET values
174.      #define MPU6050_EXT_SYNC_SET_0 (0)
175.      #define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
176.      #define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
177.      #define MPU6050_EXT_SYNC_SET_3
   (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
```

178.   #define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
179.   #define MPU6050_EXT_SYNC_SET_5
    (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))
180.   #define MPU6050_EXT_SYNC_SET_6
    (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))
181.   #define MPU6050_EXT_SYNC_SET_7
    (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
182.
183.   // Alternative names for the combined definitions.
184.   #define MPU6050_EXT_SYNC_DISABLED   MPU6050_EXT_SYNC_SET_0
185.   #define MPU6050_EXT_SYNC_TEMP_OUT_L MPU6050_EXT_SYNC_SET_1
186.   #define MPU6050_EXT_SYNC_GYRO_XOUT_L MPU6050_EXT_SYNC_SET_2
187.   #define MPU6050_EXT_SYNC_GYRO_YOUT_L MPU6050_EXT_SYNC_SET_3
188.   #define MPU6050_EXT_SYNC_GYRO_ZOUT_L MPU6050_EXT_SYNC_SET_4
189.   #define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5
190.   #define MPU6050_EXT_SYNC_ACCEL_YOUT_L MPU6050_EXT_SYNC_SET_6
191.   #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L MPU6050_EXT_SYNC_SET_7
192.
193.   // Combined definitions for the DLPF_CFG values
194.   #define MPU6050_DLPF_CFG_0 (0)
195.   #define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
196.   #define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
197.   #define MPU6050_DLPF_CFG_3 (bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
198.   #define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
199.   #define MPU6050_DLPF_CFG_5 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0))
200.   #define MPU6050_DLPF_CFG_6 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))
201.   #define MPU6050_DLPF_CFG_7
    (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
202.
203.   // Alternative names for the combined definitions
204.   // This name uses the bandwidth (Hz) for the accelometer,
205.   // for the gyro the bandwidth is almost the same.
206.   #define MPU6050_DLPF_260HZ   MPU6050_DLPF_CFG_0
207.   #define MPU6050_DLPF_184HZ   MPU6050_DLPF_CFG_1
208.   #define MPU6050_DLPF_94HZ   MPU6050_DLPF_CFG_2
209.   #define MPU6050_DLPF_44HZ   MPU6050_DLPF_CFG_3
210.   #define MPU6050_DLPF_21HZ   MPU6050_DLPF_CFG_4
211.   #define MPU6050_DLPF_10HZ   MPU6050_DLPF_CFG_5
212.   #define MPU6050_DLPF_5HZ   MPU6050_DLPF_CFG_6
213.   #define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7
214.
215.   // GYRO_CONFIG Register
216.   // The XG_ST, YG_ST, ZG_ST are bits for selftest.
217.   // The FS_SEL sets the range for the gyro.
218.   // These are the names for the bits.
219.   // Use these only with the bit() macro.
220.   #define MPU6050_FS_SEL0 MPU6050_D3
221.   #define MPU6050_FS_SEL1 MPU6050_D4
222.   #define MPU6050_ZG_ST MPU6050_D5
223.   #define MPU6050_YG_ST MPU6050_D6
224.   #define MPU6050_XG_ST MPU6050_D7
225.
226.   // Combined definitions for the FS_SEL values
227.   #define MPU6050_FS_SEL_0 (0)

```
228.        #define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
229.        #define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
230.        #define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))
231.
232.        // Alternative names for the combined definitions
233.        // The name uses the range in degrees per second.
234.        #define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
235.        #define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
236.        #define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
237.        #define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3
238.
239.        // ACCEL_CONFIG Register
240.        // The XA_ST, YA_ST, ZA_ST are bits for selftest.
241.        // The AFS_SEL sets the range for the accelerometer.
242.        // These are the names for the bits.
243.        // Use these only with the bit() macro.
244.        #define MPU6050_ACCEL_HPF0 MPU6050_D0
245.        #define MPU6050_ACCEL_HPF1 MPU6050_D1
246.        #define MPU6050_ACCEL_HPF2 MPU6050_D2
247.        #define MPU6050_AFS_SEL0   MPU6050_D3
248.        #define MPU6050_AFS_SEL1   MPU6050_D4
249.        #define MPU6050_ZA_ST      MPU6050_D5
250.        #define MPU6050_YA_ST      MPU6050_D6
251.        #define MPU6050_XA_ST      MPU6050_D7
252.
253.        // Combined definitions for the ACCEL_HPF values
254.        #define MPU6050_ACCEL_HPF_0 (0)
255.        #define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
256.        #define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
257.        #define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
258.        #define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
259.        #define MPU6050_ACCEL_HPF_7
    (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
260.
261.        // Alternative names for the combined definitions
262.        // The name uses the Cut-off frequency.
263.        #define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
264.        #define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
265.        #define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
266.        #define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
267.        #define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
268.        #define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7
269.
270.        // Combined definitions for the AFS_SEL values
271.        #define MPU6050_AFS_SEL_0 (0)
272.        #define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
273.        #define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
274.        #define MPU6050_AFS_SEL_3 (bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))
275.
276.        // Alternative names for the combined definitions
277.        // The name uses the full scale range for the accelerometer.
278.        #define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
279.        #define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
280.        #define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
```

```
281.     #define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3
282.
283.     // FIFO_EN Register
284.     // These are the names for the bits.
285.     // Use these only with the bit() macro.
286.     #define MPU6050_SLV0_FIFO_EN  MPU6050_D0
287.     #define MPU6050_SLV1_FIFO_EN  MPU6050_D1
288.     #define MPU6050_SLV2_FIFO_EN  MPU6050_D2
289.     #define MPU6050_ACCEL_FIFO_EN MPU6050_D3
290.     #define MPU6050_ZG_FIFO_EN    MPU6050_D4
291.     #define MPU6050_YG_FIFO_EN    MPU6050_D5
292.     #define MPU6050_XG_FIFO_EN    MPU6050_D6
293.     #define MPU6050_TEMP_FIFO_EN  MPU6050_D7
294.
295.     // I2C_MST_CTRL Register
296.     // These are the names for the bits.
297.     // Use these only with the bit() macro.
298.     #define MPU6050_I2C_MST_CLK0  MPU6050_D0
299.     #define MPU6050_I2C_MST_CLK1  MPU6050_D1
300.     #define MPU6050_I2C_MST_CLK2  MPU6050_D2
301.     #define MPU6050_I2C_MST_CLK3  MPU6050_D3
302.     #define MPU6050_I2C_MST_P_NSR MPU6050_D4
303.     #define MPU6050_SLV_3_FIFO_EN MPU6050_D5
304.     #define MPU6050_WAIT_FOR_ES   MPU6050_D6
305.     #define MPU6050_MULT_MST_EN   MPU6050_D7
306.
307.     // Combined definitions for the I2C_MST_CLK
308.     #define MPU6050_I2C_MST_CLK_0 (0)
309.     #define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
310.     #define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
311.     #define MPU6050_I2C_MST_CLK_3
   (bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
312.     #define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
313.     #define MPU6050_I2C_MST_CLK_5
   (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
314.     #define MPU6050_I2C_MST_CLK_6
   (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
315.     #define MPU6050_I2C_MST_CLK_7
   (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
316.     #define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
317.     #define MPU6050_I2C_MST_CLK_9
   (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))
318.     #define MPU6050_I2C_MST_CLK_10
   (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))
319.     #define MPU6050_I2C_MST_CLK_11
   (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
320.     #define MPU6050_I2C_MST_CLK_12
   (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))
321.     #define MPU6050_I2C_MST_CLK_13
   (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
322.     #define MPU6050_I2C_MST_CLK_14
   (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
```

323.   #define MPU6050_I2C_MST_CLK_15
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))

324.
325.   // Alternative names for the combined definitions
326.   // The names uses I2C Master Clock Speed in kHz.
327.   #define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
328.   #define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
329.   #define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
330.   #define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
331.   #define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
332.   #define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
333.   #define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
334.   #define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
335.   #define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
336.   #define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
337.   #define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
338.   #define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
339.   #define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
340.   #define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
341.   #define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
342.   #define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15
343.
344.   // I2C_SLV0_ADDR Register
345.   // These are the names for the bits.
346.   // Use these only with the bit() macro.
347.   #define MPU6050_I2C_SLV0_RW MPU6050_D7
348.
349.   // I2C_SLV0_CTRL Register
350.   // These are the names for the bits.
351.   // Use these only with the bit() macro.
352.   #define MPU6050_I2C_SLV0_LEN0   MPU6050_D0
353.   #define MPU6050_I2C_SLV0_LEN1   MPU6050_D1
354.   #define MPU6050_I2C_SLV0_LEN2   MPU6050_D2
355.   #define MPU6050_I2C_SLV0_LEN3   MPU6050_D3
356.   #define MPU6050_I2C_SLV0_GRP   MPU6050_D4
357.   #define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
358.   #define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
359.   #define MPU6050_I2C_SLV0_EN   MPU6050_D7
360.
361.   // A mask for the length
362.   #define MPU6050_I2C_SLV0_LEN_MASK 0x0F
363.
364.   // I2C_SLV1_ADDR Register
365.   // These are the names for the bits.
366.   // Use these only with the bit() macro.
367.   #define MPU6050_I2C_SLV1_RW MPU6050_D7
368.
369.   // I2C_SLV1_CTRL Register
370.   // These are the names for the bits.
371.   // Use these only with the bit() macro.
372.   #define MPU6050_I2C_SLV1_LEN0   MPU6050_D0
373.   #define MPU6050_I2C_SLV1_LEN1   MPU6050_D1
374.   #define MPU6050_I2C_SLV1_LEN2   MPU6050_D2

```c
375.        #define MPU6050_I2C_SLV1_LEN3   MPU6050_D3
376.        #define MPU6050_I2C_SLV1_GRP     MPU6050_D4
377.        #define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
378.        #define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
379.        #define MPU6050_I2C_SLV1_EN      MPU6050_D7
380.
381.        // A mask for the length
382.        #define MPU6050_I2C_SLV1_LEN_MASK 0x0F
383.
384.        // I2C_SLV2_ADDR Register
385.        // These are the names for the bits.
386.        // Use these only with the bit() macro.
387.        #define MPU6050_I2C_SLV2_RW MPU6050_D7
388.
389.        // I2C_SLV2_CTRL Register
390.        // These are the names for the bits.
391.        // Use these only with the bit() macro.
392.        #define MPU6050_I2C_SLV2_LEN0   MPU6050_D0
393.        #define MPU6050_I2C_SLV2_LEN1   MPU6050_D1
394.        #define MPU6050_I2C_SLV2_LEN2   MPU6050_D2
395.        #define MPU6050_I2C_SLV2_LEN3   MPU6050_D3
396.        #define MPU6050_I2C_SLV2_GRP     MPU6050_D4
397.        #define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
398.        #define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
399.        #define MPU6050_I2C_SLV2_EN      MPU6050_D7
400.
401.        // A mask for the length
402.        #define MPU6050_I2C_SLV2_LEN_MASK 0x0F
403.
404.        // I2C_SLV3_ADDR Register
405.        // These are the names for the bits.
406.        // Use these only with the bit() macro.
407.        #define MPU6050_I2C_SLV3_RW MPU6050_D7
408.
409.        // I2C_SLV3_CTRL Register
410.        // These are the names for the bits.
411.        // Use these only with the bit() macro.
412.        #define MPU6050_I2C_SLV3_LEN0   MPU6050_D0
413.        #define MPU6050_I2C_SLV3_LEN1   MPU6050_D1
414.        #define MPU6050_I2C_SLV3_LEN2   MPU6050_D2
415.        #define MPU6050_I2C_SLV3_LEN3   MPU6050_D3
416.        #define MPU6050_I2C_SLV3_GRP     MPU6050_D4
417.        #define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
418.        #define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
419.        #define MPU6050_I2C_SLV3_EN      MPU6050_D7
420.
421.        // A mask for the length
422.        #define MPU6050_I2C_SLV3_LEN_MASK 0x0F
423.
424.        // I2C_SLV4_ADDR Register
425.        // These are the names for the bits.
426.        // Use these only with the bit() macro.
427.        #define MPU6050_I2C_SLV4_RW MPU6050_D7
428.
```

```
429.        // I2C_SLV4_CTRL Register
430.        // These are the names for the bits.
431.        // Use these only with the bit() macro.
432.        #define MPU6050_I2C_MST_DLY0     MPU6050_D0
433.        #define MPU6050_I2C_MST_DLY1     MPU6050_D1
434.        #define MPU6050_I2C_MST_DLY2     MPU6050_D2
435.        #define MPU6050_I2C_MST_DLY3     MPU6050_D3
436.        #define MPU6050_I2C_MST_DLY4     MPU6050_D4
437.        #define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
438.        #define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
439.        #define MPU6050_I2C_SLV4_EN      MPU6050_D7
440.
441.        // A mask for the delay
442.        #define MPU6050_I2C_MST_DLY_MASK 0x1F
443.
444.        // I2C_MST_STATUS Register
445.        // These are the names for the bits.
446.        // Use these only with the bit() macro.
447.        #define MPU6050_I2C_SLV0_NACK MPU6050_D0
448.        #define MPU6050_I2C_SLV1_NACK MPU6050_D1
449.        #define MPU6050_I2C_SLV2_NACK MPU6050_D2
450.        #define MPU6050_I2C_SLV3_NACK MPU6050_D3
451.        #define MPU6050_I2C_SLV4_NACK MPU6050_D4
452.        #define MPU6050_I2C_LOST_ARB  MPU6050_D5
453.        #define MPU6050_I2C_SLV4_DONE MPU6050_D6
454.        #define MPU6050_PASS_THROUGH  MPU6050_D7
455.
456.        // I2C_PIN_CFG Register
457.        // These are the names for the bits.
458.        // Use these only with the bit() macro.
459.        #define MPU6050_CLKOUT_EN       MPU6050_D0
460.        #define MPU6050_I2C_BYPASS_EN   MPU6050_D1
461.        #define MPU6050_FSYNC_INT_EN    MPU6050_D2
462.        #define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
463.        #define MPU6050_INT_RD_CLEAR    MPU6050_D4
464.        #define MPU6050_LATCH_INT_EN    MPU6050_D5
465.        #define MPU6050_INT_OPEN        MPU6050_D6
466.        #define MPU6050_INT_LEVEL       MPU6050_D7
467.
468.        // INT_ENABLE Register
469.        // These are the names for the bits.
470.        // Use these only with the bit() macro.
471.        #define MPU6050_DATA_RDY_EN    MPU6050_D0
472.        #define MPU6050_I2C_MST_INT_EN MPU6050_D3
473.        #define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
474.        #define MPU6050_ZMOT_EN        MPU6050_D5
475.        #define MPU6050_MOT_EN         MPU6050_D6
476.        #define MPU6050_FF_EN          MPU6050_D7
477.
478.        // INT_STATUS Register
479.        // These are the names for the bits.
480.        // Use these only with the bit() macro.
481.        #define MPU6050_DATA_RDY_INT   MPU6050_D0
482.        #define MPU6050_I2C_MST_INT    MPU6050_D3
```

```
483.    #define MPU6050_FIFO_OFLOW_INT MPU6050_D4
484.    #define MPU6050_ZMOT_INT       MPU6050_D5
485.    #define MPU6050_MOT_INT        MPU6050_D6
486.    #define MPU6050_FF_INT         MPU6050_D7
487.
488.    // MOT_DETECT_STATUS Register
489.    // These are the names for the bits.
490.    // Use these only with the bit() macro.
491.    #define MPU6050_MOT_ZRMOT MPU6050_D0
492.    #define MPU6050_MOT_ZPOS  MPU6050_D2
493.    #define MPU6050_MOT_ZNEG  MPU6050_D3
494.    #define MPU6050_MOT_YPOS  MPU6050_D4
495.    #define MPU6050_MOT_YNEG  MPU6050_D5
496.    #define MPU6050_MOT_XPOS  MPU6050_D6
497.    #define MPU6050_MOT_XNEG  MPU6050_D7
498.
499.    // IC2_MST_DELAY_CTRL Register
500.    // These are the names for the bits.
501.    // Use these only with the bit() macro.
502.    #define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
503.    #define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
504.    #define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
505.    #define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
506.    #define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
507.    #define MPU6050_DELAY_ES_SHADOW MPU6050_D7
508.
509.    // SIGNAL_PATH_RESET Register
510.    // These are the names for the bits.
511.    // Use these only with the bit() macro.
512.    #define MPU6050_TEMP_RESET  MPU6050_D0
513.    #define MPU6050_ACCEL_RESET MPU6050_D1
514.    #define MPU6050_GYRO_RESET  MPU6050_D2
515.
516.    // MOT_DETECT_CTRL Register
517.    // These are the names for the bits.
518.    // Use these only with the bit() macro.
519.    #define MPU6050_MOT_COUNT0     MPU6050_D0
520.    #define MPU6050_MOT_COUNT1     MPU6050_D1
521.    #define MPU6050_FF_COUNT0      MPU6050_D2
522.    #define MPU6050_FF_COUNT1      MPU6050_D3
523.    #define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
524.    #define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5
525.
526.    // Combined definitions for the MOT_COUNT
527.    #define MPU6050_MOT_COUNT_0 (0)
528.    #define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
529.    #define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
530.    #define MPU6050_MOT_COUNT_3
    (bit(MPU6050_MOT_COUNT1)|bit(MPU6050_MOT_COUNT0))
531.
532.    // Alternative names for the combined definitions
533.    #define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0
534.
535.    // Combined definitions for the FF_COUNT
```

```
536.        #define MPU6050_FF_COUNT_0 (0)
537.        #define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
538.        #define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
539.        #define MPU6050_FF_COUNT_3 (bit(MPU6050_FF_COUNT1)|bit(MPU6050_FF_COUNT0))
540.
541.        // Alternative names for the combined definitions
542.        #define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0
543.
544.        // Combined definitions for the ACCEL_ON_DELAY
545.        #define MPU6050_ACCEL_ON_DELAY_0 (0)
546.        #define MPU6050_ACCEL_ON_DELAY_1 (bit(MPU6050_ACCEL_ON_DELAY0))
547.        #define MPU6050_ACCEL_ON_DELAY_2 (bit(MPU6050_ACCEL_ON_DELAY1))
548.        #define MPU6050_ACCEL_ON_DELAY_3
   (bit(MPU6050_ACCEL_ON_DELAY1)|bit(MPU6050_ACCEL_ON_DELAY0))
549.
550.        // Alternative names for the ACCEL_ON_DELAY
551.        #define MPU6050_ACCEL_ON_DELAY_0MS MPU6050_ACCEL_ON_DELAY_0
552.        #define MPU6050_ACCEL_ON_DELAY_1MS MPU6050_ACCEL_ON_DELAY_1
553.        #define MPU6050_ACCEL_ON_DELAY_2MS MPU6050_ACCEL_ON_DELAY_2
554.        #define MPU6050_ACCEL_ON_DELAY_3MS MPU6050_ACCEL_ON_DELAY_3
555.
556.        // USER_CTRL Register
557.        // These are the names for the bits.
558.        // Use these only with the bit() macro.
559.        #define MPU6050_SIG_COND_RESET MPU6050_D0
560.        #define MPU6050_I2C_MST_RESET  MPU6050_D1
561.        #define MPU6050_FIFO_RESET     MPU6050_D2
562.        #define MPU6050_I2C_IF_DIS     MPU6050_D4   // must be 0 for MPU-6050
563.        #define MPU6050_I2C_MST_EN     MPU6050_D5
564.        #define MPU6050_FIFO_EN        MPU6050_D6
565.
566.        // PWR_MGMT_1 Register
567.        // These are the names for the bits.
568.        // Use these only with the bit() macro.
569.        #define MPU6050_CLKSEL0      MPU6050_D0
570.        #define MPU6050_CLKSEL1      MPU6050_D1
571.        #define MPU6050_CLKSEL2      MPU6050_D2
572.        #define MPU6050_TEMP_DIS     MPU6050_D3   // 1: disable temperature sensor
573.        #define MPU6050_CYCLE        MPU6050_D5   // 1: sample and sleep
574.        #define MPU6050_SLEEP        MPU6050_D6   // 1: sleep mode
575.        #define MPU6050_DEVICE_RESET MPU6050_D7   // 1: reset to default values
576.
577.        // Combined definitions for the CLKSEL
578.        #define MPU6050_CLKSEL_0 (0)
579.        #define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
580.        #define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
581.        #define MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
582.        #define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
583.        #define MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL0))
584.        #define MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1))
585.        #define MPU6050_CLKSEL_7
   (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
586.
587.        // Alternative names for the combined definitions
```

```
588.     #define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
589.     #define MPU6050_CLKSEL_X           MPU6050_CLKSEL_1
590.     #define MPU6050_CLKSEL_Y           MPU6050_CLKSEL_2
591.     #define MPU6050_CLKSEL_Z           MPU6050_CLKSEL_3
592.     #define MPU6050_CLKSEL_EXT_32KHZ   MPU6050_CLKSEL_4
593.     #define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
594.     #define MPU6050_CLKSEL_RESERVED    MPU6050_CLKSEL_6
595.     #define MPU6050_CLKSEL_STOP        MPU6050_CLKSEL_7
596.
597.     // PWR_MGMT_2 Register
598.     // These are the names for the bits.
599.     // Use these only with the bit() macro.
600.     #define MPU6050_STBY_ZG      MPU6050_D0
601.     #define MPU6050_STBY_YG      MPU6050_D1
602.     #define MPU6050_STBY_XG      MPU6050_D2
603.     #define MPU6050_STBY_ZA      MPU6050_D3
604.     #define MPU6050_STBY_YA      MPU6050_D4
605.     #define MPU6050_STBY_XA      MPU6050_D5
606.     #define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
607.     #define MPU6050_LP_WAKE_CTRL1 MPU6050_D7
608.
609.     // Combined definitions for the LP_WAKE_CTRL
610.     #define MPU6050_LP_WAKE_CTRL_0 (0)
611.     #define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
612.     #define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
613.     #define MPU6050_LP_WAKE_CTRL_3
    (bit(MPU6050_LP_WAKE_CTRL1)|bit(MPU6050_LP_WAKE_CTRL0))
614.
615.     // Alternative names for the combined definitions
616.     // The names uses the Wake-up Frequency.
617.     #define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
618.     #define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
619.     #define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
620.     #define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3
621.
622.
623.     // Default I2C address for the MPU-6050 is 0x68.
624.     // But only if the AD0 pin is low.
625.     // Some sensor boards have AD0 high, and the
626.     // I2C address thus becomes 0x69.
627.     #define MPU6050_I2C_ADDRESS 0x68
628.
629.
630.     // Declaring an union for the registers and the axis values.
631.     // The byte order does not match the byte order of
632.     // the compiler and AVR chip.
633.     // The AVR chip (on the Arduino board) has the Low Byte
634.     // at the lower address.
635.     // But the MPU-6050 has a different order: High Byte at
636.     // lower address, so that has to be corrected.
637.     // The register part "reg" is only used internally,
638.     // and are swapped in code.
639.     typedef union accel_t_gyro_union
640.     {
```

```
641.        struct
642.        {
643.          uint8_t x_accel_h;
644.          uint8_t x_accel_l;
645.          uint8_t y_accel_h;
646.          uint8_t y_accel_l;
647.          uint8_t z_accel_h;
648.          uint8_t z_accel_l;
649.          uint8_t t_h;
650.          uint8_t t_l;
651.          uint8_t x_gyro_h;
652.          uint8_t x_gyro_l;
653.          uint8_t y_gyro_h;
654.          uint8_t y_gyro_l;
655.          uint8_t z_gyro_h;
656.          uint8_t z_gyro_l;
657.        } reg;
658.        struct
659.        {
660.          int16_t x_accel;
661.          int16_t y_accel;
662.          int16_t z_accel;
663.          int16_t temperature;
664.          int16_t x_gyro;
665.          int16_t y_gyro;
666.          int16_t z_gyro;
667.        } value;
668.      };
669.
670.
671.      void setup()
672.      {
673.        int error;
674.        uint8_t c;
675.
676.
677.        Serial.begin(9600);
678.        Serial.println(F("InvenSense MPU-6050"));
679.        Serial.println(F("June 2012"));
680.
681.        // Initialize the 'Wire' class for the I2C-bus.
682.        Wire.begin();
683.
684.
685.        // default at power-up:
686.        //   Gyro at 250 degrees second
687.        //   Acceleration at 2g
688.        //   Clock source at internal 8MHz
689.        //   The device is in sleep mode.
690.        //
691.
692.        error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
693.        Serial.print(F("WHO_AM_I : "));
694.        Serial.print(c,HEX);
```

```
695.        Serial.print(F(", error = "));
696.        Serial.println(error,DEC);
697.
698.        // According to the datasheet, the 'sleep' bit
699.        // should read a '1'.
700.        // That bit has to be cleared, since the sensor
701.        // is in sleep mode at power-up.
702.        error = MPU6050_read (MPU6050_PWR_MGMT_1, &c, 1);
703.        Serial.print(F("PWR_MGMT_1 : "));
704.        Serial.print(c,HEX);
705.        Serial.print(F(", error = "));
706.        Serial.println(error,DEC);
707.
708.
709.         // Clear the 'sleep' bit to start the sensor.
710.         MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
711.       }
712.
713.
714.      void loop()
715.      {
716.       int error;
717.       double dT;
718.       accel_t_gyro_union accel_t_gyro;
719.
720.
721.        Serial.println(F(""));
722.        Serial.println(F("MPU-6050"));
723.
724.        // Read the raw values.
725.        // Read 14 bytes at once,
726.        // containing acceleration, temperature and gyro.
727.        // With the default settings of the MPU-6050,
728.        // there is no filter enabled, and the values
729.        // are not very stable.
730.        error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) &accel_t_gyro,
    sizeof(accel_t_gyro));
731.        Serial.print(F("Read accel, temp and gyro, error = "));
732.        Serial.println(error,DEC);
733.
734.
735.        // Swap all high and low bytes.
736.        // After this, the registers values are swapped,
737.        // so the structure name like x_accel_l does no
738.        // longer contain the lower byte.
739.        uint8_t swap;
740.        #define SWAP(x,y) swap = x; x = y; y = swap
741.
742.        SWAP (accel_t_gyro.reg.x_accel_h, accel_t_gyro.reg.x_accel_l);
743.        SWAP (accel_t_gyro.reg.y_accel_h, accel_t_gyro.reg.y_accel_l);
744.        SWAP (accel_t_gyro.reg.z_accel_h, accel_t_gyro.reg.z_accel_l);
745.        SWAP (accel_t_gyro.reg.t_h, accel_t_gyro.reg.t_l);
746.        SWAP (accel_t_gyro.reg.x_gyro_h, accel_t_gyro.reg.x_gyro_l);
747.        SWAP (accel_t_gyro.reg.y_gyro_h, accel_t_gyro.reg.y_gyro_l);
```

```
748.        SWAP (accel_t_gyro.reg.z_gyro_h, accel_t_gyro.reg.z_gyro_l);
749.

750.

751.        // Print the raw acceleration values
752.

753.        Serial.print(F("accel x,y,z: "));
754.        Serial.print(accel_t_gyro.value.x_accel, DEC);
755.        Serial.print(F(", "));
756.        Serial.print(accel_t_gyro.value.y_accel, DEC);
757.        Serial.print(F(", "));
758.        Serial.print(accel_t_gyro.value.z_accel, DEC);
759.        Serial.println(F(""));
760.

761.

762.        // The temperature sensor is -40 to +85 degrees Celsius.
763.        // It is a signed integer.
764.        // According to the datasheet:
765.        //   340 per degrees Celsius, -512 at 35 degrees.
766.        // At 0 degrees: -512 - (340 * 35) = -12412
767.

768.        Serial.print(F("temperature: "));
769.        dT = ( (double) accel_t_gyro.value.temperature + 12412.0) / 340.0;
770.        Serial.print(dT, 3);
771.        Serial.print(F(" degrees Celsius"));
772.        Serial.println(F(""));
773.

774.

775.        // Print the raw gyro values.
776.

777.        Serial.print(F("gyro x,y,z : "));
778.        Serial.print(accel_t_gyro.value.x_gyro, DEC);
779.        Serial.print(F(", "));
780.        Serial.print(accel_t_gyro.value.y_gyro, DEC);
781.        Serial.print(F(", "));
782.        Serial.print(accel_t_gyro.value.z_gyro, DEC);
783.        Serial.print(F(", "));
784.        Serial.println(F(""));
785.

786.        delay(1000);
787.      }
788.

789.

790.      // -------------------------------------------------------
791.      // MPU6050_read
792.      //
793.      // This is a common function to read multiple bytes
794.      // from an I2C device.
795.      //
796.      // It uses the boolean parameter for Wire.endTransMission()
797.      // to be able to hold or release the I2C-bus.
798.      // This is implemented in Arduino 1.0.1.
799.      //
800.      // Only this function is used to read.
801.      // There is no function for a single byte.
```

```
802.      //
803.      int MPU6050_read(int start, uint8_t *buffer, int size)
804.      {
805.        int i, n, error;
806.
807.        Wire.beginTransmission(MPU6050_I2C_ADDRESS);
808.        n = Wire.write(start);
809.        if (n != 1)
810.          return (-10);
811.
812.        n = Wire.endTransmission(false);    // hold the I2C-bus
813.        if (n != 0)
814.          return (n);
815.
816.        // Third parameter is true: relase I2C-bus after data is read.
817.        Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
818.        i = 0;
819.        while(Wire.available() && i<size)
820.        {
821.          buffer[i++]=Wire.read();
822.        }
823.        if ( i != size)
824.          return (-11);
825.
826.        return (0);  // return : no error
827.      }
828.
829.
830.      // ---------------------------------------------------------
831.      // MPU6050_write
832.      //
833.      // This is a common function to write multiple bytes to an I2C device.
834.      //
835.      // If only a single register is written,
836.      // use the function MPU_6050_write_reg().
837.      //
838.      // Parameters:
839.      //   start : Start address, use a define for the register
840.      //   pData : A pointer to the data to write.
841.      //   size  : The number of bytes to write.
842.      //
843.      // If only a single register is written, a pointer
844.      // to the data has to be used, and the size is
845.      // a single byte:
846.      //   int data = 0;        // the data to write
847.      //   MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
848.      //
849.      int MPU6050_write(int start, const uint8_t *pData, int size)
850.      {
851.        int n, error;
852.
853.        Wire.beginTransmission(MPU6050_I2C_ADDRESS);
854.        n = Wire.write(start);        // write the start address
855.        if (n != 1)
```

```
856.          return (-20);
857.
858.        n = Wire.write(pData, size);  // write data bytes
859.        if (n != size)
860.          return (-21);
861.
862.        error = Wire.endTransmission(true); // release the I2C-bus
863.        if (error != 0)
864.          return (error);
865.
866.        return (0);        // return : no error
867.      }
868.
869.      // --------------------------------------------------------
870.      // MPU6050_write_reg
871.      //
872.      // An extra function to write a single register.
873.      // It is just a wrapper around the MPU_6050_write()
874.      // function, and it is only a convenient function
875.      // to make it easier to write a single register.
876.      //
877.      int MPU6050_write_reg(int reg, uint8_t data)
878.      {
879.        int error;
880.
881.        error = MPU6050_write(reg, &data, 1);
882.
883.        return (error);
884.      }
```