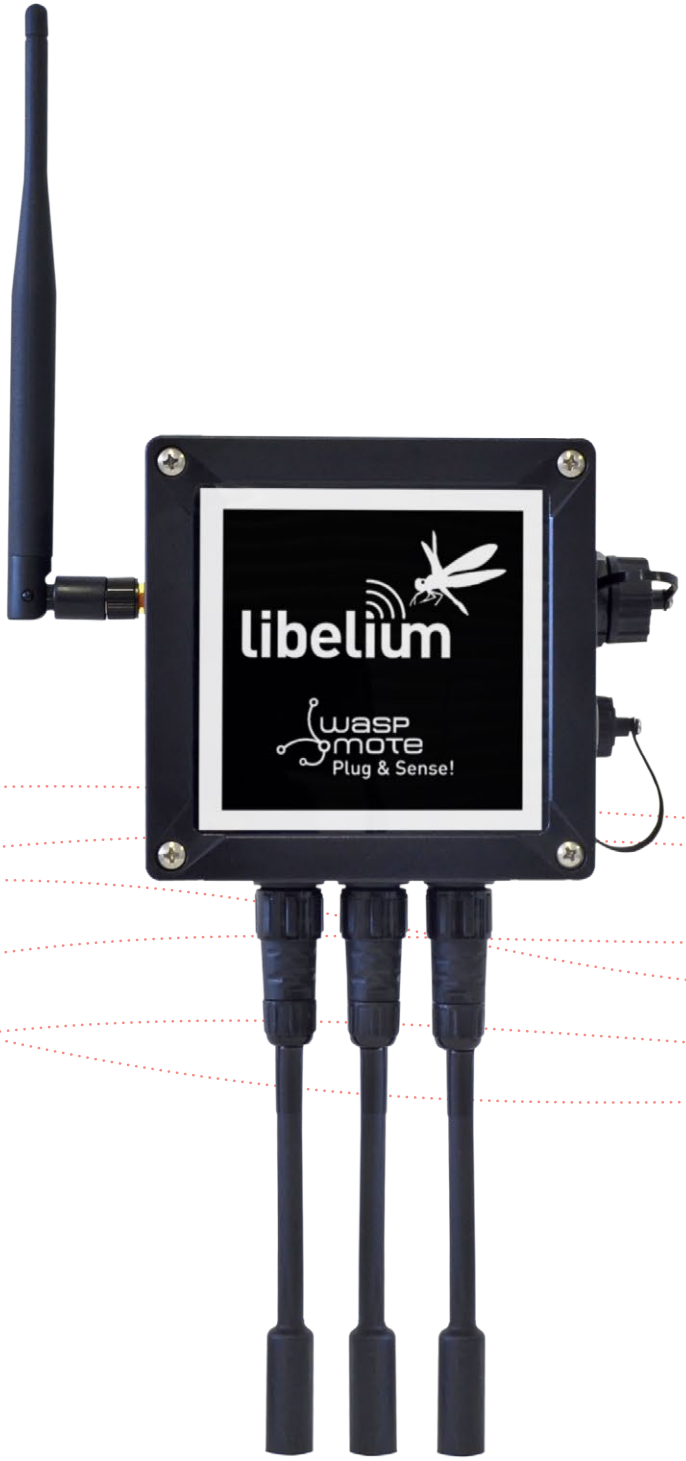
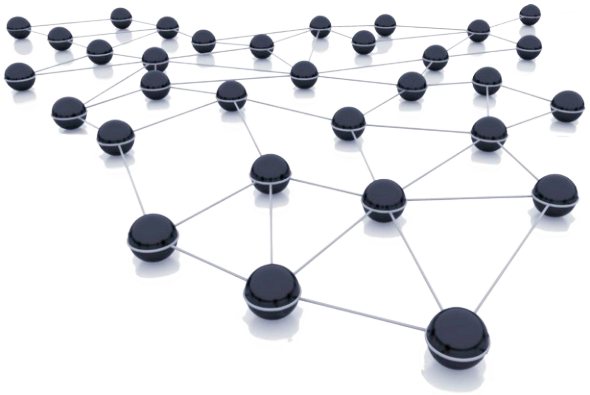
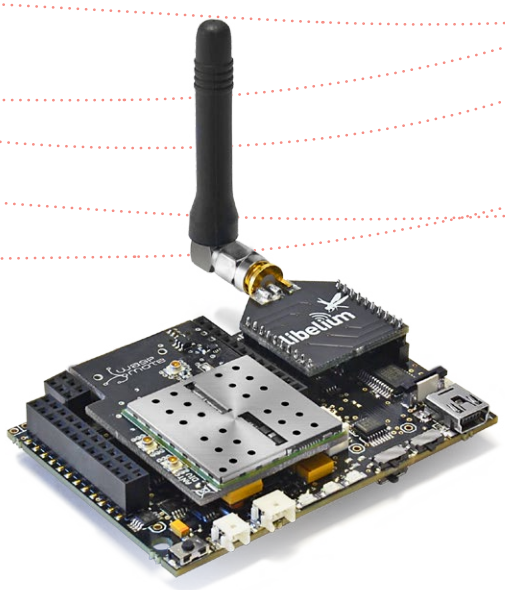
Waspmote Accelerometer

Programming Guide







libelium

wasp mote

Document version: v7.0 - 02/2017

INDEX

1. Introduction

This guide explains the accelerometer features and functions. There are no great variations in this library for our new product lines Waspmote v15 and Plug & Sense! v15, released on October 2016.

Anyway, if you are using previous versions of our products, please use the corresponding guides, available on our Development website.

You can get more information about the generation change on the document "New generation of Libelium product lines".

1.1. Waspmote libraries

1.1.1. Waspmote ACC files

WaspACC.h; WaspACC.cpp

1.1.2. Constructor

To start using Waspmote ACC library, an object from class 'WaspACC' must be created. This object, called ACC, is created inside the Waspmote ACC library and it is public to all libraries. It is used through the guide to show how the Waspmote ACC library works.

When creating this constructor, no variables are initialized by default.

1.1.3. Configuration registers and constants

There are many registers and constants defined in Waspmote ACC library. These registers and constants are used to define some configuration parameters used in Free-Fall or Inertial Wake-Up events and other tasks in the library. Through this guide, all these values will be explained.

The thresholds used to generate interruptions are values selected according to the tests performed. However, each developer should analyze its scenario and decide if the thresholds have to be modified.

1.1.4. Flags

There are various flags used to handle interruptions and errors while using the Waspmote accelerometer.

1.1.4.1. Status flag

The status flag is used to set reading or writing errors or hardware interruptions. Possible values are:

- `ACC_ERROR_READING`: error reading register.
- `ACC_ERROR_WRITING`: error writing register.

1.1.4.2. Global Interruption Flag

It is used to check the port in which the interruption has got activated. It is used in this library, and it will be used in other libraries which generate interrupts too.

2. Initialization

Before getting information from the accelerometer, it has to be initialized, setting its registers with the appropriate values.

2.1. Initializing the accelerometer

The `ON()` function initializes the communication with the accelerometer via I2C bus and sets up the registers.

It makes no tests and checks nothing from the communication, therefore returns nothing and modifies no flags.

Full-scale value can be selected when initializing the accelerometer, options are:

`FS_2G` → ±2G (default)

`FS_4G` → ±4G

`FS_8G` → ±8G

Example of use:

```
{
    // Initializes the accelerometer with a full-scale value of ±2G
    ACC.ON();

    // Initializes the accelerometer with a full-scale value of ±2G
    ACC.ON( FS_2G );

    // Initializes the accelerometer with a full-scale value of ±4G
    ACC.ON( FS_4G );

    // Initializes the accelerometer with a full-scale value of ±8G
    ACC.ON( FS_8G );
}
```

2.2. Rebooting the accelerometer

The `boot()` function reboots the accelerometer, taking for granted the accelerometer is on and forcing the sensor to reboot by writing to the control register 2.

It makes no tests and checks nothing from the communication, therefore returns nothing and modifies no flags.

Example of use:

```
{
    ACC.boot();
}
```

2.3. Closing the accelerometer

The `OFF()` function sets the accelerometer module to the Power Down mode taking for granted the accelerometer is on and powering down the sensor.

It closes I2C bus as well.

It makes no tests and checks nothing from the communication, therefore returns nothing and modifies no flags.

Example of use:

```
{
  ACC.OFF();
}
```

2.4. Getting status

The `getStatus()` function gets the accelerometer status.

It returns a byte containing the status of the accelerometer reading from the proper register. In the case there is a communication error the `ACC_ERROR_READING` flag will be set.

Example of use:

```
{
  uint8_t status;
  status = ACC.getStatus();
}
```

2.5. Getting correct working

The `check()` function gets if the accelerometer is present and is working properly. It reads a dummy register that should always answer 0x32, otherwise an error occurred.

This function can be used for determining if the accelerometer is on the board and checking if it is still working properly.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error communicating to the register.

Example of use:

```
{
  uint8_t status;
  status = ACC.check();
}
```

Available information:

On correct working, 'status' should be equal to 0x32.

3. Getting acceleration on axis

3.1. Registers

There are several registers used to get acceleration on three axis. These registers are defined in the Waspmote ACC library as constants, that can be modified by the user if required. However, these definitions should not be modified unless registers addresses change in future accelerometer revisions.

Note: The *accelerometer can use three scales: ±2g, ±4g and ±8g. See section "Initializing the accelerometer" for more information. The following examples have been tested using the ±2g scale.*

3.2. Getting acceleration on axis X

The `getX()` function checks accelerometer's acceleration on X axis. The value returned is defined in mG units.

It returns the combined contents of data registers `outXhigh` and `outXlow` as an integer.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error communicating to the register.

Example of use:

```
{
   int accX;
   accX = ACC.getX();
}
```

3.3. Getting acceleration on axis Y

The `getY()` function checks accelerometer's acceleration on Y axis. The value returned is defined in mG units.

It returns the combined contents of data registers `outYhigh` and `outYlow` as an integer.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   int accY;
   accY = ACC.getY();
}
```

3.4. Getting acceleration on axis Z

The `getZ()` function checks accelerometer's acceleration on Z axis. The value returned is defined in mG units.

It returns the combined contents of data registers `outZhigh` and `outZlow` as an integer.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   int accZ;
   accZ = ACC.getZ();
}
```

4. Accelerometer interrupts

The Waspmote's accelerometer can provide an interrupt signal and offers several possibilities for personalizing this signal. The registers involved in the interrupt generation behavior are:

- `CTRL_REG3`
- `INT1_CFG:`
- `INT1_SRC`
- `INT1_THS`
- `INT1_DURATION`

4.1. Free Fall interrupt

Free-fall detection refers to a specific configuration of the interrupt registers that allows the recognition of device free-fall. In real cases, a "free fall zone" is defined around the zero-g level, where all accelerations are small enough to generate the interrupt.

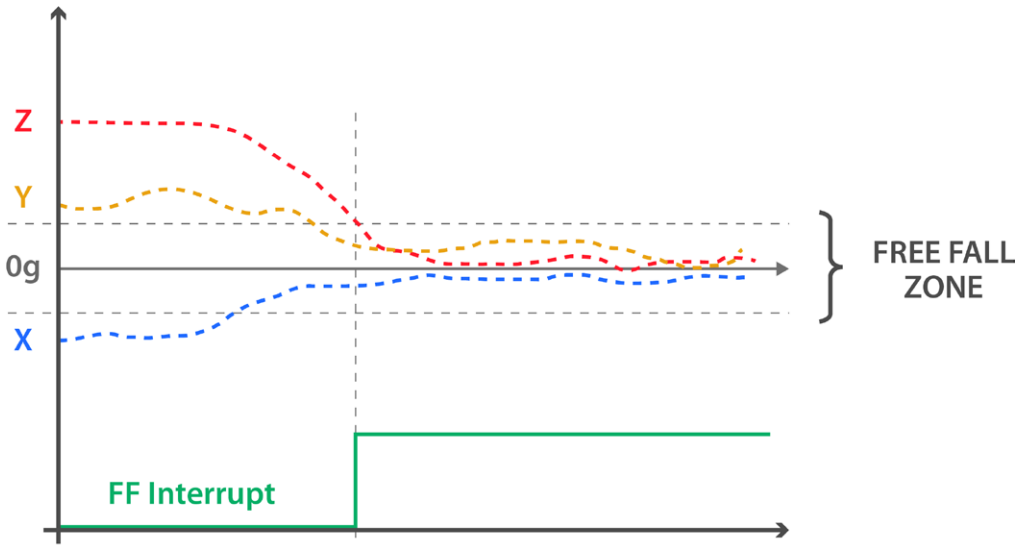


Figure 1: Free-Fall interruption

4.1.1. Setting the Free Fall interrupt

The `setFF()` function sets the Free Fall interrupt.

First of all, it clears previous interruptions and writes into the below explained registers the selected values. After writing the registers, the hardware interruption is attached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.setFF();
}
```

The default free-fall zone margin is set to 448 mG. It can be changed when calling the `setFF()` function. For this purpose, the new margin has to be defined as input in mG units.

Example of use:

```
{
   ACC.setFF( 300 );
}
```

4.1.2. Unsetting the Free Fall interrupt

The `unsetFF()` function clears and unsets the Free Fall interrupt.

First of all, it clears `CTRL_REG3` and `INT1_SRC` registers and the hardware interruption is detached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.unsetFF();
}
```

4.2. Inertial Wake-Up interrupt

The wake-up interrupt will generate an event when the acceleration on axis X, Y or Z exceeds a threshold.

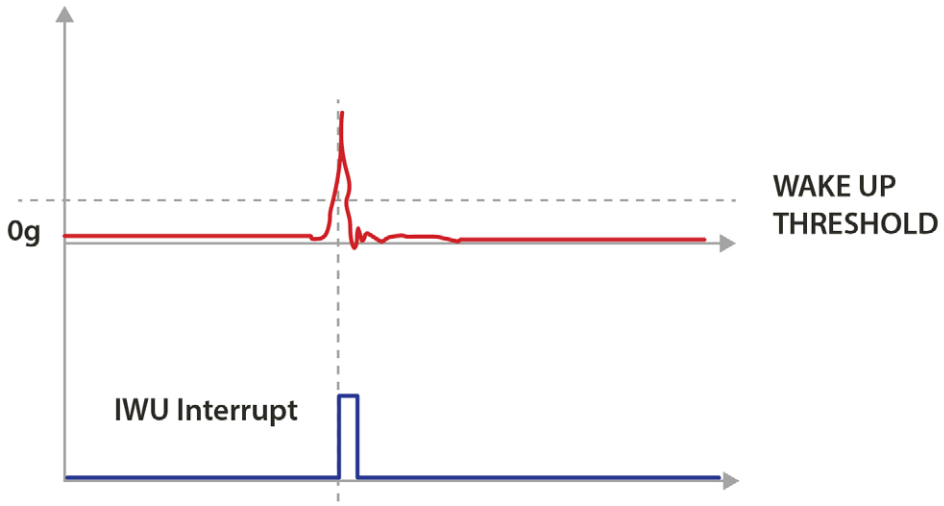


Figure 2: Inertial wake-up interruption

4.2.1. Setting the Inertial Wake-Up interrupt

The `setIWU()` function sets the Inertial Wake-Up interruption.

First of all, it clears previous interruptions and writes into the below explained registers the selected values. After writing the registers, the hardware interruption is attached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.setIWU();
}
```

The default inertial wake-up threshold is set to 448 mG. It can be changed when calling the setFF() function. For this purpose, the new margin has to be defined as input in mG units.

Example of use:

```
{
   ACC.setIWU( 300 );
}
```

4.2.2. Unsetting the Inertial Wake-Up interrupt

The `unsetIWU()` function clears and unsets the Inertial Wake-Up interruption.

First of all, it clears `CTRL_REG3` and `INT1_SRC` registers and the hardware interruption is detached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.unsetIWU();
}
```

4.3. 6D movement interrupt

The 6D movement interrupt is generated when the device moves from one direction to a different direction.

4.3.1. Setting the 6D movement interrupt

The `set6DMovement()` function sets the Inertial 6D movement interruption.

First of all, it clears previous interruptions and writes into the below explained registers the selected values. After writing the registers, the hardware interruption is attached to the pin.

It returns 'flag' to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.set6DMovement();
}
```

4.3.2. Unsetting the 6D movement interrupt

The `unset6DMovement()` function clears and unsets the Inertial Wake-Up interruption.

First of all, it clears `CTRL_REG3` and `INT1_SRC` registers and the hardware interruption is detached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.unset6DMovement();
}
```

4.4. 6D position interrupt

The 6D position interrupt is generated when the device is stable in a direction. The interrupt is active as long as the position is maintained.

4.4.1. Setting the 6D position interrupt

The `set6DPosition()` function sets the 6D position interrupt.

First of all, it clears previous interruptions and writes into the below explained registers the selected values. After writing the registers, the hardware interruption is attached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.set6DPosition();
}
```

4.4.2. Unsetting 6D position interrupt

The `unset6DPosition()` function clears and unsets the 6D position interrupt.

First of all, it clears `CTRL_REG3` and `INT1_SRC` register and the hardware interruption is detached to the pin.

It returns flag to check if there was any error while writing to registers or attaching the interruption.

Example of use:

```
{
   ACC.unset6DPosition ();
}
```

5. Accelerometer modes

There are seven work modes for the accelerometer:

- On
- Power Down
- Low power 1
- Low power 2
- Low power 3
- Low power 4
- Low power 5

To change between these modes, there are two functions that have been developed.

5.1. Setting accelerometer mode

It sets the accelerometer's work mode. It configures the accelerometer to a new work mode, where the possibilities are:

`ACC_ON`	Normal mode: Output data rate 50 Hz
`ACC_POWER_DOWN`	Turn power off
`ACC_LOW_POWER_1`	Low power mode: Output data rate 0.5 Hz
`ACC_LOW_POWER_2`	Low power mode: Output data rate 1 Hz
`ACC_LOW_POWER_3`	Low power mode: Output data rate 2 Hz
`ACC_LOW_POWER_4`	Low power mode: Output data rate 5 Hz
`ACC_LOW_POWER_5`	Low power mode: Output data rate 10 Hz

Since it calls `writeRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error communicating to the register.

Example of use:

```
{
   ACC.setMode(ACC_LOW_POWER_2);
}
```

Related variable:

`accMode` → stores the work mode

5.2. Getting accelerometer mode

It checks accelerometer's work mode.

It returns the value for the accelerometer's work mode, the possibilities are:

`ACC_ON`	Normal mode: Output data rate 50 Hz
`ACC_POWER_DOWN`	Turn power off
`ACC_LOW_POWER_1`	Low power mode: Output data rate 0.5 Hz
`ACC_LOW_POWER_2`	Low power mode: Output data rate 1 Hz
`ACC_LOW_POWER_3`	Low power mode: Output data rate 2 Hz
`ACC_LOW_POWER_4`	Low power mode: Output data rate 5 Hz
`ACC_LOW_POWER_5`	Low power mode: Output data rate 10 Hz

It does not call any other functions and therefore it will not activate any flags.

Example of use:

```
{
   uint8_t workMode;
   workMode = ACC.getMode();
}
```

6. Advanced functions

The Accelerometer module working mode is to set and read registers, thus when it is wanted to change a data a register is set and when data wants to be read a register is read. There are some functions for writing and reading registers. As well, there are some registers that are used to configure the accelerometer module. These registers are called `CTRL_REG1`, `CTRL_REG2`, `CTRL_REG3`, `CTRL_REG4` and `CTRL_REG5` and are defined in the Waspmote ACC library as constants.

Some functions have been created to set and read these registers. However, these functions are not necessary for a basic accelerometer use.

6.1. Writing a register

It writes a byte into a register in the accelerometer.

It returns 1 if there is no error or 0 if there is error.

Example of use:

```
{
   uint8_t reg1 = B01000111;
   ACC.writeRegister(CTRL_REG1,reg1);
}
```

6.2. Reading a register

It reads a register from the accelerometer.

It returns 1 if there is no error or 0 if there is error.

Example of use:

```
{
   uint16_t statusRegister = 0;
   statusRegister = ACC.readRegister(statusReg);
}
```

6.3. Setting CTRL_REG1

It sets accelerometer's control register 1.

It returns '0' if error.

Since it calls `writeRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg1 = B00100111;
   ACC.setCTRL1(reg1);
}
```

6.4. Getting CTRL_REG1

It checks accelerometer's control register 1.

It returns the contents of control register 1.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg1;
   reg1 = ACC.getCTRL1();
}
```

6.5. Setting CTRL_REG2

It sets accelerometer's control register 2.

It returns '0' if error.

Since it calls `writeRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg2 = B00001000;
   ACC.setCTRL2(reg2);
}
```

6.6. Getting CTRL_REG2

It checks accelerometer's control register 2.

It returns the contents of control register 2.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg2;
   reg2 = ACC.getCTRL2();
}
```

6.7. Setting CTRL_REG3

It sets accelerometer's control register 3.

It returns '0' if error.

Since it calls `writeRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg3 = B00001000;
   ACC.setCTRL3(reg3);
}
```

6.8. Getting CTRL_REG3

It checks accelerometer's control register 3.

It returns the contents of control register 3.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg3;
   reg3 = ACC.getCTRL3();
}
```

6.9. Setting CTRL_REG4

It sets accelerometer's control register 4.

It returns '0' if error.

Since it calls `writeRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg4 = B00001000;
   ACC.setCTRL4(reg4);
}
```

6.10. Getting CTRL_REG4

It checks accelerometer's control register 4.

It returns the contents of control register 4.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg4;
   reg4 = ACC.getCTRL4();
 }
```

6.11. Setting CTRL_REG5

It sets accelerometer's control register 5.

It returns '0' if error.

Since it calls `writeRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg5 = B00001000;
   ACC.setCTRL5(reg5);
}
```

6.12. Getting CTRL_REG5

It checks accelerometer's control register 5.

It returns the contents of control register 5.

Since it calls `readRegister()`, it will not activate any flags by itself, but the other functions will activate `ACC_ERROR_READING` in case there was an error when communicating to the register.

Example of use:

```
{
   uint8_t reg5;
   reg5 = ACC.getCTRL5();
}
```

6.13. Sleep to wake mode

The sleep to wake function, in conjunction with Low Power mode, allows further reduction of system power consumption and the development of new smart applications. The accelerometer can be set in a low-power operating mode, characterized by lower date rate refreshments. In this way the device, even if "sleeping", continues sensing acceleration and generating interrupt requests.

When the sleep to wake function is activated, the accelerometer is able to automatically wake up Waspmote as soon as the interrupt event has been detected, increasing the output data rate and bandwidth. With this feature the Waspmote can be efficiently switched from Low Power mode to full performance, depending on user-selectable positioning and acceleration events, thus ensuring power saving and flexibility.

6.13.1. Setting sleep to wake mode

It sets the sleep to wake mode of the accelerometer.

Example of use:

```
{
   ACC.setSleepToWake();
}
```

6.13.2. Unsetting the sleep to wake mode

It unsets the sleep to wake mode of the accelerometer.

Example of use:

```
{
   ACC.unSetSleepToWake();
}
```

7. Code examples and extended information

In the Waspmote Development section you can find complete examples:

http://www.libelium.com/development/waspmote/examples

Next lines show a complete example of use of the accelerometer functions:

```
/*
 *  ------ [ACC_1] Waspmote Accelerometer Basic Example --------
 *
 *  Explanation: This example shows how to get the acceleration on the
 *  different axis using the most basic functions related with Waspmote
 *  accelerometer
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *  Version:           0.1
 *  Design:            David Gascón
 *  Implementation:    Marcos Yarza
 */

void setup()
{
  ACC.ON();
  USB.ON(); // starts using the serial port
  USB.println(F("ACC_01 example"));
}

void loop()
{

  //----------Check Register----------------------
  // should always answer 0x32, it is used to check
  // the proper functionality of the accelerometer
  byte check = ACC.check();

  //----------X Value-----------------------
  int x_acc = ACC.getX();

  //----------Y Value-----------------------
  int y_acc = ACC.getY();

  //----------Z Value-----------------------
  int z_acc = ACC.getZ();

  //-------------------------------

  USB.print(F("\n-------------------------------\nCheck: 0x"));
  USB.println(check, HEX);
  USB.println(F("\n \t0X\t0Y\t0Z"));
```

```
 USB.print(F(“ ACC\t”));
 USB.print(x_acc, DEC);
 USB.print(F(“\t”));
 USB.print(y_acc, DEC);
 USB.print(F(“\t”));
 USB.println(z_acc, DEC);

 delay(5000);
}
```

8. API Changelog

Keep track of the software changes on this link:

http://www.libelium.com/development/waspmote/documentation/changelog/#Accelerometer