Chapter 2 Waspmote

2.1. What's Waspmote?

Waspmote is the brand name of a modular platform created by Libelium and used in Wireless Sensor Networks (WSN). A wireless sensor network (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants and to cooperatively compute their data through the network. The WSN is built of nodes, in this case each node is a Waspmote module.

The idea of a modular architecture is to integrate only the modules needed in each device. Modules can be changed and expanded depending on the needs, this is the strong point of the platform.

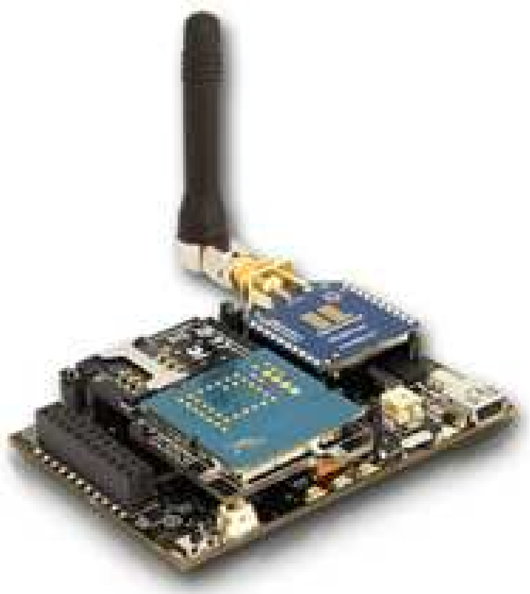


Figure 2.1. *Waspmote*

2.2. Specifications.

- Microcontroller: ATmega1281
- SRAM: 8KB
- EEPROM: 4KB
- FLASH: 128KB
- SD Card: 2GB
- Weight: 20gr
- Dimensions: 73.5 x 51 x 13 mm
- Temperature Range: [-20ºC, +65ºC]

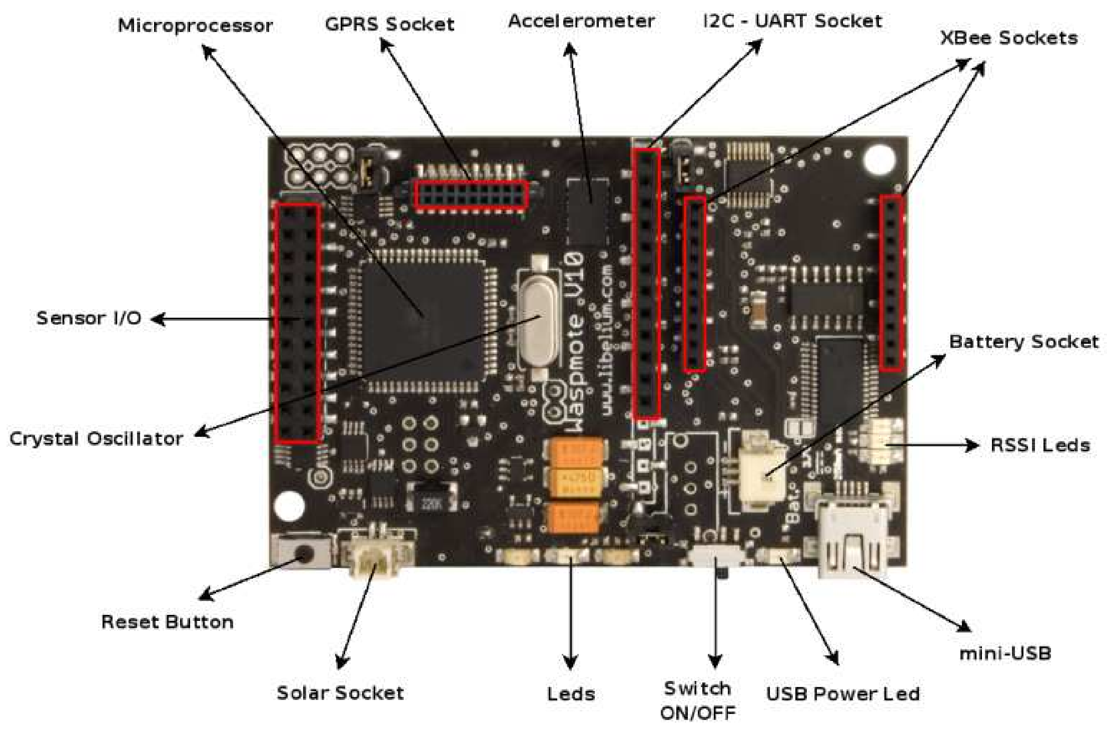


Figure 2.2. *Main Waspmote components - Top side*

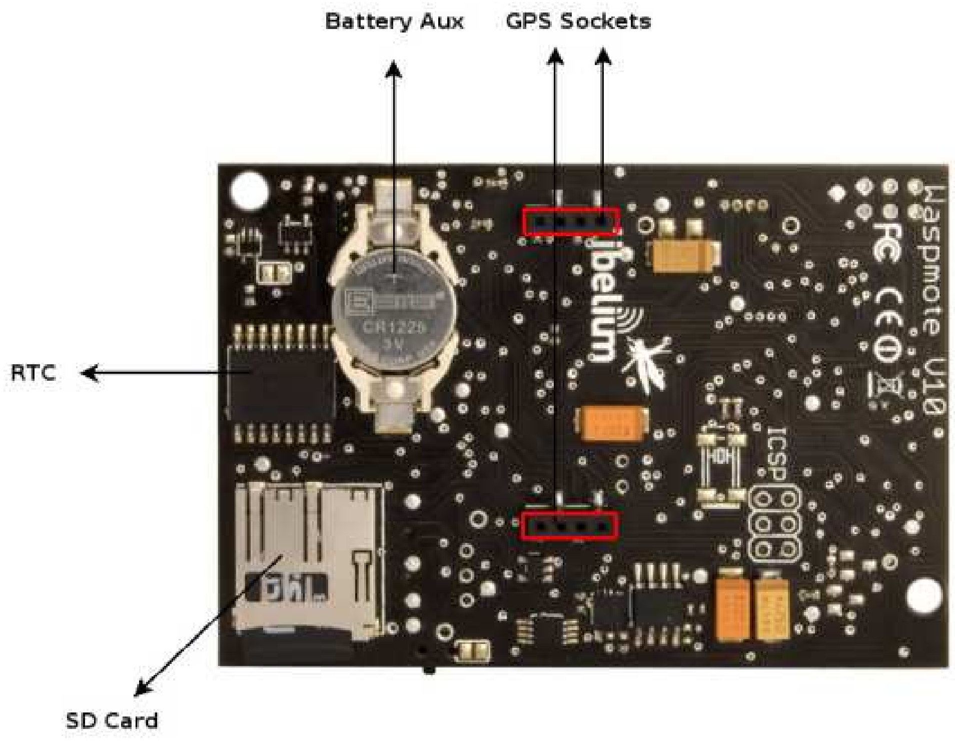


Figure 2.3. *Main Waspmote components - Bottom side*

2.3. Built-in sensors on the board.

- Temperature: -40ºC , +85ºC. Accuracy: 0.25ºC
- Accelerometer: +/- 2g (1024 LSb/g) / +/- 6g (340LSb/g) 40Hz/160Hz/640Hz/2560Hz

2.4. Electrical Data.

- Minimum operational battery voltage: 3.3V
- Maximun operational battery voltage: 4.2V
- Voltage in any pin: [-0.5V, +3.8V]
- Maximum current fron any digital I/O pin: 40mA
- Charged battery voltage: 4.2V

2.5. Block Diagram.

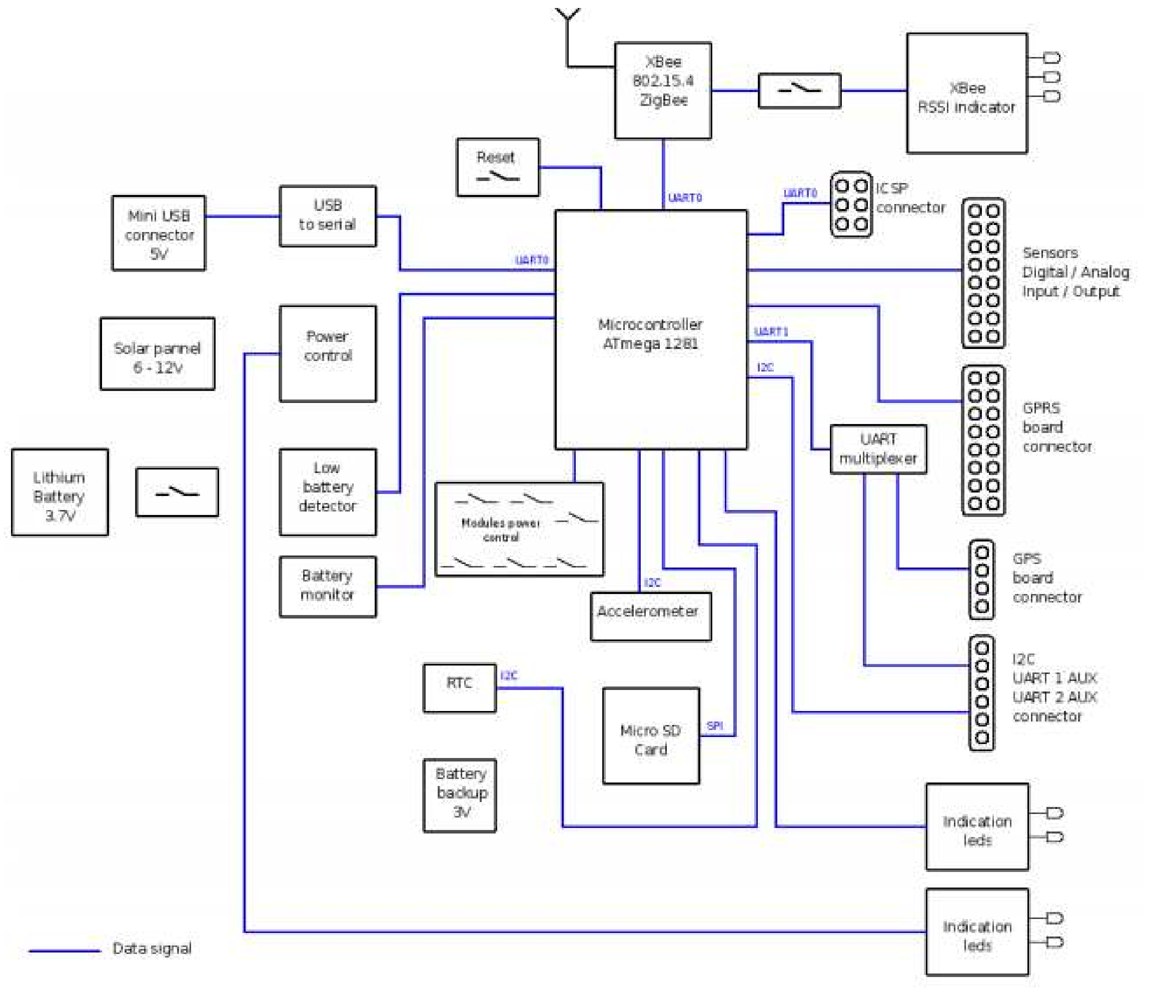


Figure 2.4. *Waspmote block diagram - Data signal*

The XBee module, the ICSP connector and USB connector share the same UART, so these modules can't work simultaneously. This feature creates most of the problems communicating with the board. However, we can obtain the expansion radio board and connect the XBee module in UART1.

The other microcontroller UART is connected to a four channel multiplexer, and it is possible to select in the same program which of the four new channels is required to connect to the UART on the microcontroller. These channels are connected as follows. One is connected to the GPRS/3G board, the other to the GPS and the other two are accessible to the user in the auxiliary I2C – UART connector.

The I2C communication bus is also used in Waspmote where three devices are connected in parallel: the accelerometer, the RTC and the digital potentiometer (digipot) which configures the low battery alarm threshold level. In all cases, the microcontroller acts as master while the other devices connected to the bus are slaves. The SPI port on the microcontroller is used for communication with the micro SD card.

2.6. Energy System.

The following schematic shows the power supply circuit:

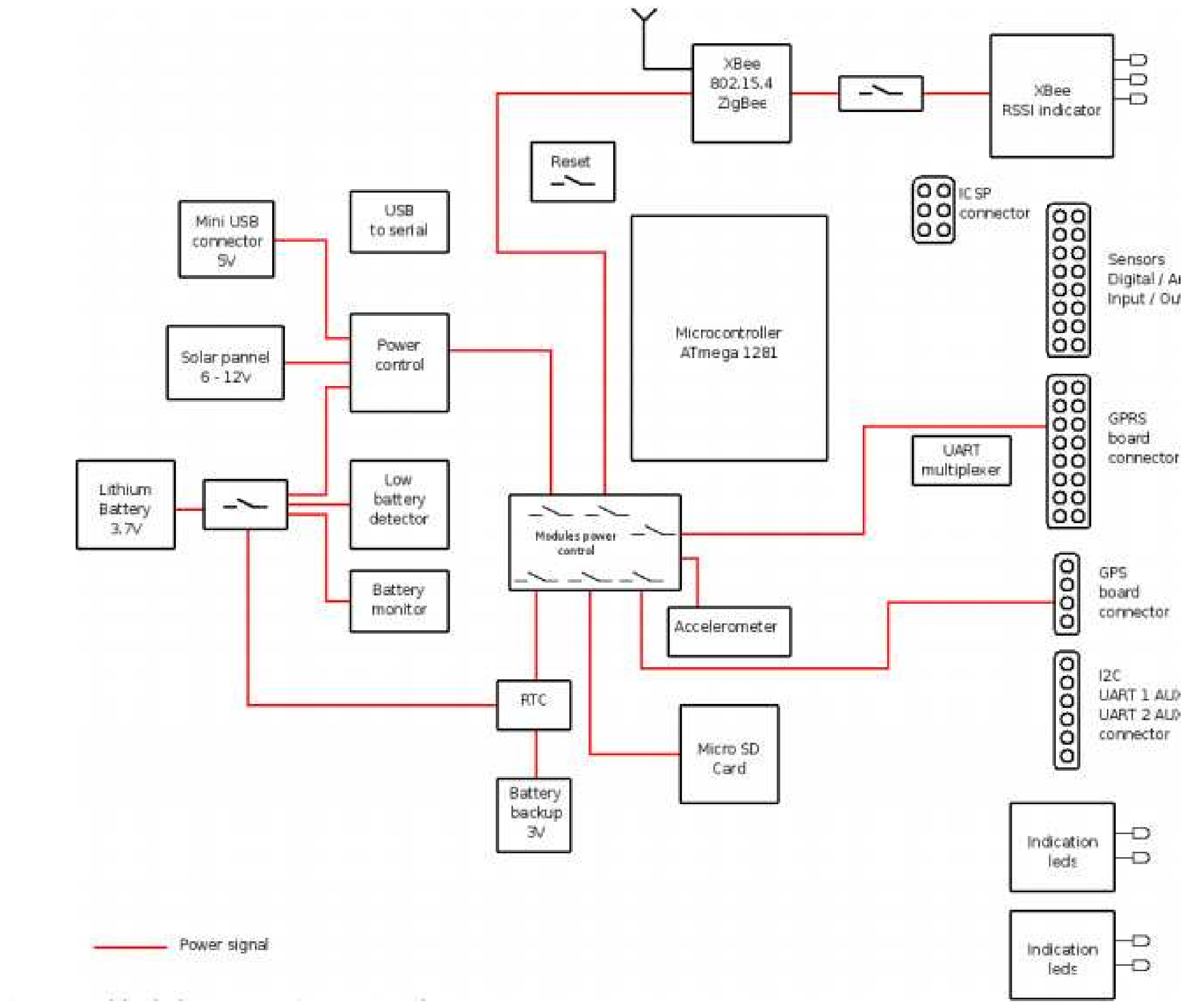


Figure 2.5. *Waspmote block diagram - Power signal*

2.7. LEDs.

Waspmote board has the following LEDs:

- **LED 0 – programmable LED.**

A green indicator LED is connected to the microcontroller. It is totally programmable by the user from the program code. In addition, the LED 0 indicates when Waspmote resets, blinking each time a reset on the board is carried out.

- **LED 1 – programmable LED.**

A red indicator LED is connected to the microcontroller. It is totally programmable by the user.

The functions for handling these LEDs are “Utils.setLED() Utils.getLED() and Utils.blinkLEDs(). These functions are capable of changing the state of the LEDs. There are one function to change their state, other one to get their state and another to blink LEDs. An example of use is:

```
{
 Utils.setLED(LED0,LED_ON); // Sets the LED0 ON
 delay(1000); // Delay of 1000ms
 Utils.setLED(LED0,LED_OFF); // Sets the LED0 OFF
 state=Utils.getLED(LED1);   // Gets the state of LED1
 Utils.blinkLEDs(1000);   // Blink LEDs using a delay of1000ms
}
```

- **Charging battery LED indicator.**

A red LED indicating that there is a battery connected in Waspmote which is being charged, the charging can be done through a mini USB cable or through a solar panel connected to Waspmote. Once the battery is completely charged, the LED switches off automatically.

- **USB Power LED indicator.**

A green LED which indicates when Waspmote is connected to a compatible USB port either for battery charging or programming. When the LED is on it indicates that the USB cable is connected correctly, when the USB cable is removed the LED will switch off automatically.

- **RSSI LEDs**

3 LEDs have been included to show the RSSI (Received Strength Signal Indicator) . The RSSI parameter indicates the signal quality of the last packet received of the ZigBee/802.15.4 frames. The XBee modules provide this information in all protocol and frequency variants.

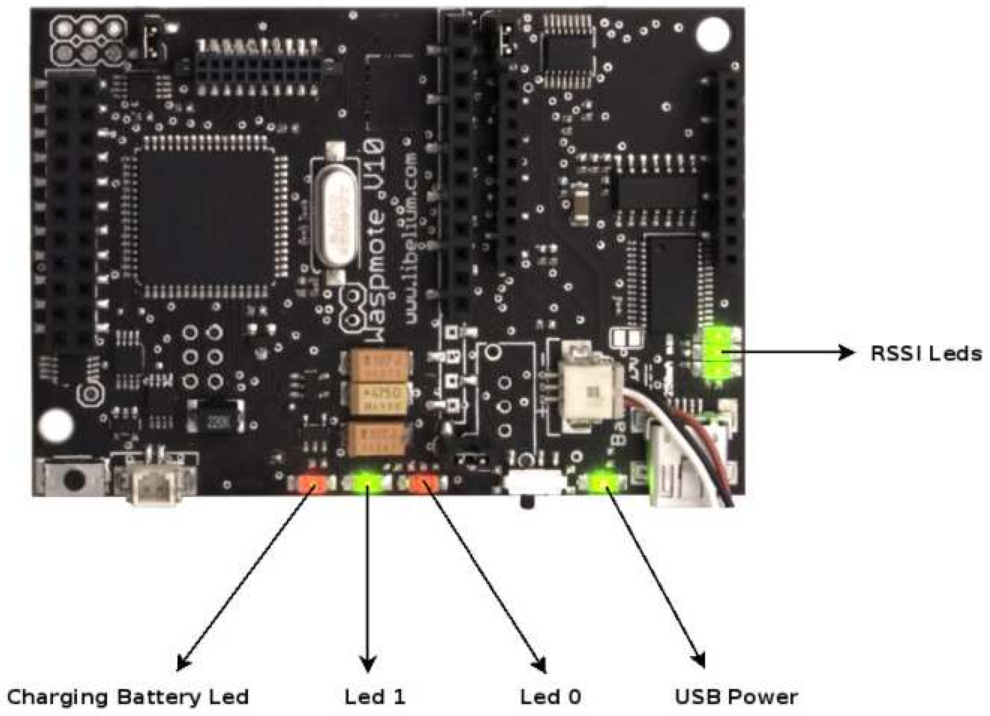


Figure 2.6. *Visual indicator LED*

In Waspmote this value is obtained simply by executing the function:

```
{
  xbee.getRSSI();
}
```

Three LEDs have been included to show the RSSI . These LEDs indicate the signal quality of the last packet received. By default this indicator system is disconnected to avoid unnecessary consumption. For this reason, a jumper must initially be connected (see section 1.8) so that this part of the circuit is connected to the XBee module.

LEDs switched ON	RSSI received
LED 1	RSSI > (Sensitivity + 11.1 dB)
LEDs 1+2	RSSI > (Sensitivity + 21.23 dB)
LEDs 1+2+3	RSSI > (Sensitivity + 31.82 dB)

Table 2.1. *Relationship of lit LEDs with RSSI received*

The XBee.setRSSIOutput() function must be run to establish how long these LEDs are turned on. This time can vary from 0 (deactivated) to the time a new packet is received (FF).

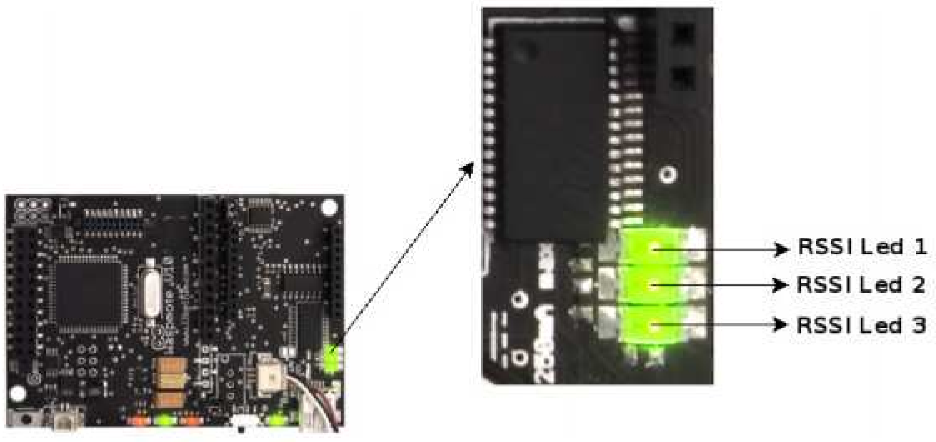


Figure 2.7. *RSSI level LED indicator*

```
{
  xbee.setRSSItime(0xFF);
  xbee.getRSSItime();
}
```

2.8. Jumpers.

In Waspmote, there are three jumpers to activate and deactivate certain functions, the description of the jumpers is:

- **Programming enabling jumper.**

Board programming is only possible if this jumper is set.

- **RSSI indicater enabling jumper.**

If the jumper is set, the indicating function for the RSSI LEDs is enabled. If it is not being in used, the jumper should be taken out to avoid extra consumption.

- **Hibernate mode enabling jumper.**

If the jumper is not set, Waspmoete can't be programmed. Hibernate operation mode needs the jumper to be removed during Waspmote set up.

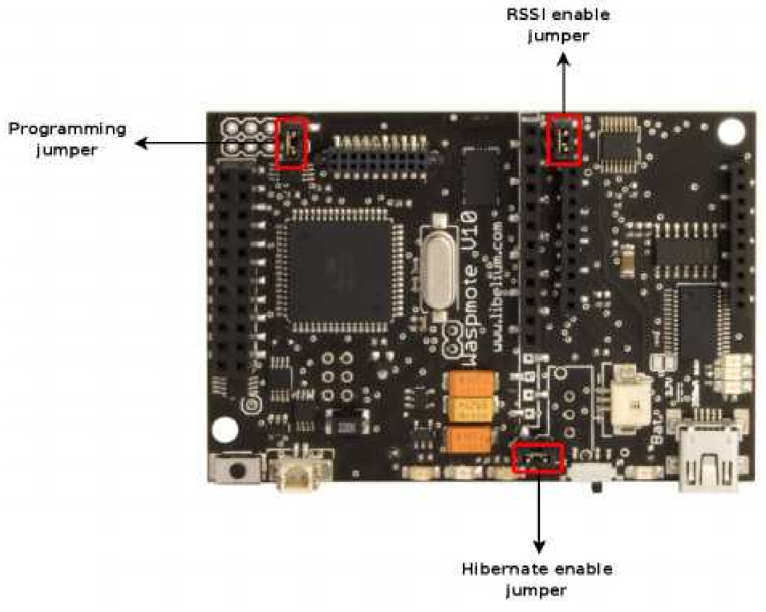


Figure 2.8. *Jumpers in Waspmote*

2.9. Inputs/Outputs.

Waspmote can communicate with other external devices through the following inputs/outputs:

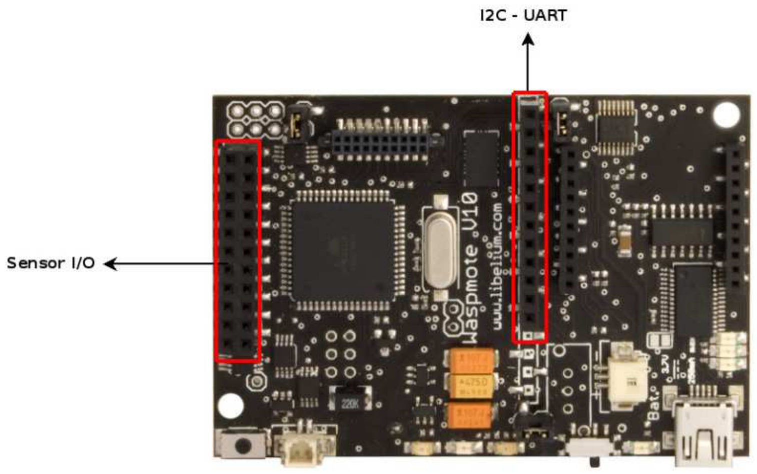


Figure 2.9. *Inputs and Outputs in Waspmote*

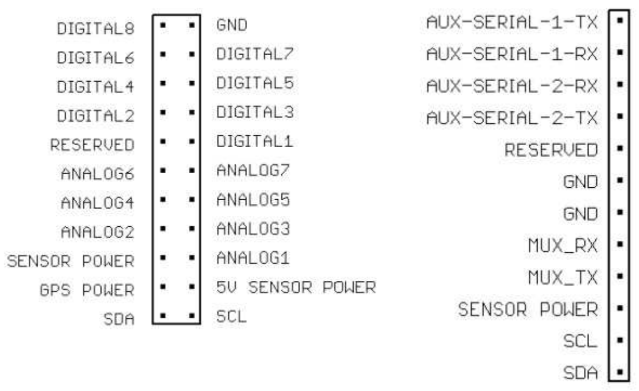


Figure 2.10. *Description of sensor connector pins (left) and description of auxiliary I2C-UART connector pins (right)*

2.9.1. Analog.

Waspmote has 7 accessible analog inputs in the sensor connector. Each input is directly connected to the microcontroller. The microcontroller uses a 10 bit successive approximation analog digital converter (ADC).

The reference voltage value for the inputs is 0V (GND). The maximum value of input voltage is 3.3V which corresponds to the microcontroller's general power voltage.

To obtain input values, the function "analogRead(analog input) " is used, the function's input parameter will be the name of the input to be read "ANALOG1, ANALOG2…" (see sensor connector figure). The value obtained from this function will be an integer number between 0 and 1023, 0 corresponds to 0 V and 1023 to 3.3 V.

```
{
   int val = 0;
   val = analogRead(ANALOG1); // Read the input ANALOG1
}
```

The analog input pins can also be used as digital input/output pins. If these pins are going to be used as digital ones, the following table for pin names must be taken into account:

2.9.2. Digital.

Waspmote has digital pins which can be configured as input or output depending on the needs of the application. The voltage values corresponding to logic 0 is 0V and voltage corresponding to logic 1 is 3.3 V.

Analog Pin	Digital Pin
ANALOG1	14
ANALOG2	15
ANALOG3	16
ANALOG4	17
ANALOG5	18
ANALOG6	19
ANALOG7	20

Table 2.2. *Mapping analog-digital pin*

```
{// Setting pin mode
  pinMode(DIGITAL1,INPUT); // Sets DIGITAL1 as an input
 pinMode(DIGITAL4,OUTPUT); // Sets DIGITAL4 as an output
 // Reading digital inputs
 val=digitalRead(DIGITAL1); // Reads the value (0-1)
 // Writing digital outputs
 digitalWrite(DIGITAL4,HIGH); // Writes High=3.3V  to Digital4
 digitalWrite(DIGITAL5,LOW); // Writes Low=0V  to Digital5
}
```

2.9.3. PWM.

DIGITAL1 pin can also be used as output PWM (Pulse Width Modulation) with which an analog signal can be “simulated”. It is actually a square wave between 0V and 3.3V for which the proportion of time when the signal is high can be changed (its working cycle) from 0% to 100%, simulating a voltage of 0V (0%) to 3.3V (100%).The resolution is 8 bit, so up to 255 values between 0-100% can be configured. The instruction to control the PWM output is analogWrite(DIGITAL1, value) ; where value is the analog value (0-255). As example:

```
{
  analogWrite(DIGITAL1,128);
}
```

2.9.4. USB.

USB is used in Waspmote for communication with a computer or compatible USB device. This communication allows the microcontroller’s program to be loaded. For USB communication, one of the microcontroller's UARTs is used. The FT232RL carries out the conversion to USB standard.

```
void setup()
{
 USB.close();    // Closes the UART
 USB.begin(); // Opens the UART at 38400bps by default
}
void loop()
{// Reading data
 if(USB.available()) // If data is available 1 is returned
        {
                  data_read=USB.read(); // Reads data from UART0
        }
 USB.flush();    // Frees the UART buffer. Data unread are lost.
// Writing data
 USB.println(); // Writes an EOL
 char * string=Hello;
 USB.print(string); // Writes a string to the UART
 USB.println(string); // Writes a string to the UART adding EOL
 int     integer=54345;
 USB.print(integer);    // Writes the number 54345 to the UART
 USB.println(integer); // Writes the number 54345 adding EOL
}
```

2.10. Architecture and System.

The Waspmote's architecture is based on the Atmel ATMEGA 1281 microcontroller. This processing unit starts executing the bootloader binary, which is responsible for loading into the memory the compiled programs and libraries previously stored in the FLASH memory, so that the main program that has been created can finally begin running.

When Waspmote is connected and starts the bootloader, there is a waiting time (62.5ms) before beginning the first instruction, this time is used to start loading new compiled programs updates. If a new program is received from the USB during this time, it will be loaded into the FLASH memory (128KB) substituting already existing programs. Otherwise, if a new program is not received, the last program stored in the memory will start running.

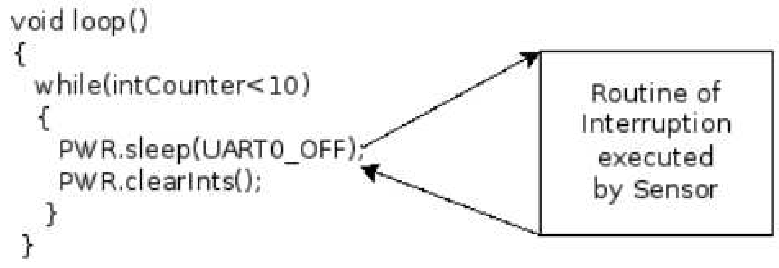
The structure of the codes is divided into 2 basic parts: a part named setup and one called loop. Both parts of the code have sequential behaviour, executing instructions in the set order. For example, the following code shows the structure of a program.

```
void setup()
{
 xbeeDM.init(DIGIMESH,FREQ2_4G,NORMAL); // XBee DigiMesh library
 xbeeDM.ON(); // Powers XBee
}
void loop()
{
  xbeeDM.setChannel(0x0A); // Chosing a channel : channel 0x0A
  if( !xbeeDM.error_AT ) XBee.println(”Channel␣set␣OK”);
  else XBee.println(”Error␣while␣changing␣channel”);

 xbeeDM.setPAN(PANID); // Chosing a PANID : PANID=0x1234
 if( !xbeeDM.error_AT ) XBee.println(”PANID␣set␣OK”);
 else XBee.println(”Error␣while␣changing␣PANID”);

 xbeeDM.encryptionMode(1); //KEY=”WaspmoteKey”
  if( !xbeeDM.error_AT ) XBee.println(”Security␣enabled”);
  else XBee.println(”Error␣while␣enabling␣security”);
 xbeeDM.setLinkKey(KEY);
  if( !xbeeDM.error_AT ) XBee.println(”Key␣set␣OK”);
  else XBee.println(”Error␣while␣setting␣Key”);

  xbeeDM.writeValues(); // Keep values
  if( !xbeeDM.error_AT ) XBee.println(”Changes␣stored␣OK”);
  else XBee.println(”Error␣while␣storing␣values”);
}
```

The setup is the first part of the code, which is only run once when the code is initialized. In this part it is recommendable to include initialization of the modules which are going to be used, as well as the part of the code which is only important when Waspmote is started. The part named loop runs continuously, forming an infinite loop. Because of the behaviour of this part of the code, the use of interruptions is recommended to perform actions with Waspmote.

A common programming technique to save energy would be based on blocking the program (either keeping the micro awake or asleep in particular cases) until some of interruptions available in Waspmote show that an event has occurred. This way, when an interruption is detected the associated function, which was previously stored in an interruption vector, is executed. To be able to detect the capture of interruptions during the execution of the code, a series of flags have been created and will be activated to indicate the event which has generated the interruption.

When Waspmote is reset or switched on, the code starts again from the setup function and then the loop function. By default, variable values declared in the code and modified in exe-



```
void loop()
{
  while(intCounter<10)
  {
    PWR.sleep(UART0_OFF);
    PWR.clearInts();
  }
}
```

Figure 2.11. *Interruption*

cution will be lost when a reset occurs or there is no battery. To store values permanently, it is necessary to use the microcontroller's EEPROM (4KB) non-volatile memory. Another option is the use of the high capacity 2GB SD card.

Waspmote uses a quartz oscillator which works at a frequency of 8MHz as a system clock. In this way, every 125ns the microcontroller runs a low level (machine language) instruction. It must be taken into account that each line of C++ code of a program compiled by Waspmote includes several instructions in machine language.

Waspmote is a device prepared for operation in adverse conditions with regards to noise and electromagnetic contamination, for this reason, to ensure stable communication at all times with the different modules connected through a serial line to the UARTs (XBee, GPRS/3G, USB) a maximum transmission speed of 38400bps has been set for XBee, GRPS and USB, and 4800 for the GPS, so that the success rate in received bits.

2.11. Interruptions.

Interruptions are signals received by the microcontroller which indicate it must stop the task is doing to attend to an event that has just happened. Interruption control frees the microcontroller from having to control sensors all the time and makes the sensors inform warn Waspmote when a determined value (threshold) is reached. Waspmote is designed to work with 2 types of interruptions: Synchronous and asynchronous.

•**Synchronous Interruptions** They are programmed by timers. They allow to program when we want them to be triggered. There are two types of timer alarms: periodic and relative.

- Periodic Alarms are those to which we specify a particular moment in the future, for example: " Alarm programmed for every fourth day of the month at 00:01 and 11 seconds" , they are controlled by the RTC.

- Relative alarms are programmed taking into account the current moment, eg: “Alarm programmed for 5 minutes and 10 seconds” , they are controlled through the RTC and the microcontroller’s internal Watchdog.

• **Asynchronous Interruptions**

These are not programmed so it is not known when they will be triggered. Types:

- Sensors: the sensor boards can be programmed so that when a sensor reaches a certain threshold it triggers an alarm.

- Low battery: Waspmote has a circuit which controls the level of battery left at anytime. When a certain threshold is reached an alarm is generated (”hey! I’m dying!”) which warns the control centre that one of the motes is running out of battery. This critical battery level will vary depending on the type of modules that Waspmote has, as well as the final application. For this reason, this threshold can be defined dynamically through the programming of a digital potentiometer by the microcontroller.

- Accelerometer: The accelerometer that is built into the Waspmote can be programmed so that certain events such as a fall or change of direction generate an interrupt.

- GPRS/3G Module: receiving a call, an SMS or data trough a TCP or UDP socket generates an interruption.

All interruptions, both synchronous and asynchronous can wake Waspmote up from the Sleep and the Deep Sleep mode . However, only the synchronous interruption by the RTC is able to wake it up from the Hibernate mode.The Hibernate mode totally disconnects the Waspmote power, leaving only the auxiliary battery powering the RTC to wake Waspmote up when the time alarm is reached. Because of this disconnection, when the RTC generates the corresponding alarm, the power in Waspmote is reconnected and the code starts again from the setup.

The way of detecting whether a reboot from the Hibernate mode has happened is to check whether the corresponding flag has been activated. Activation of this flag happens when the ‘ifHibernate()’ function is called, which must be done in the setup part of the code. This way, when Waspmote starts, it tests if it is a normal start or if it is an start from the Hibernate mode.

2.11.1. Flags.

• **intConf**
This flags vector is used to set which modules are activated to generate interruptions. Only interruptions that are previously activated in this vector will be received.

RES	SENS2	SENS	UART1	UART0	PIN	TIM2	TIM1	TIM0	WTD	RTC	BAT	ACC	LAI	HAI	RES
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 2.12. *intConf flag*

- **intFlag**

This flag is used to find out which module generated the captured interrupt. This flag marks the position corresponding to the module which generated the interrupt.

HIB	SENS2	SENS	UART1	UART0	PIN	TIM2	TIM1	TIM0	WTD	RTC	BAT	ACC	LAI	HAI	RES
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 2.13. *intFlag flag*

- **intCounter**

This flag is used to count the total number of interruptions which have been produced. Each time an interruption is generated this counter is increased.

- **intArray**

This flag is used to count the total number of interruptions that have been produced for each module. Each time an interruption is produced the position corresponding to the module which has generated it is increased.

HIB	SENS2	SENS	UART1	UART0	PIN	TIM2	TIM1	TIM0	WTD	RTC	BAT	ACC	LAI	HAI	RES
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 2.14. *intArray flag*

2.11.2. Attaching interruptions.

It attaches the interruption to the corresponding microcontroller pin and subroutine associated to it. In the following example, it attaches interrupt on pin ACC_INT_ACT using subroutine onHAIwakeUP on HIGH level.

```
{
  attachInterrupt(ACC_INT_ACT, onHAIwakeUP, HIGH);
}
```

2.11.3. Dettaching interruptions.

It detaches the interruption from the corresponding microcontroller pin.

```
{
  detachInterrupt(ACC_INT_ACT); // Detaches the interruption
}
```

2.11.4. Enabling interruptions.

When this function is called, 'intConf' flag is updated with the new active interruption. After that, it is attached to th corresponding microcontroller pin and associated subroutine.

```
{
  enableInterrupts(ACC_INT); // Enables the interruption
}
```

2.11.5. Disabling interruptions.

When this function is called, 'intConf' flag is updated with the interruption that has been detached. After that, it is detached from the corresponding microcontroller pin and associated subroutine

```
{
  disableInterrupts(ACC_INT);
}
```

2.12. Operational modes.

Waspmote has 4 operational modes.

- **ON**: Normal operation mode. Consumption in this state is 9mA.

- **Sleep**: The main program is paused, the microcontroller passes to a latent state, from which it can be woken up by all asynchronous interruptions and by the synchronous interruption generated by the Watchdog. The duration interval of this state is from 32ms to 8s . Consumption in this state is 62 μA.

• **Deep Sleep**: The main program pauses, the microcontroller passes to a latent state from which it can be woken up by all asynchronous interruptions and by the synchronous interruption triggered by the RTC. The interval of this cycle can be from 8 seconds to minutes, hours, days . Consumption in this state is 62 μA.

• **Hibernate**: The main program stops, the microcontroller and all the Waspmote modules are completely disconnected. The only way to reactivate the device is through the previously programmed alarm in the RTC (synchronous interrupt). The interval of this cycle can be from 8 seconds to minutes, hours, days. When the device is totally disconnected from the main battery the RTC is powered through an auxiliary battery from which it consumes 0.7 μA.

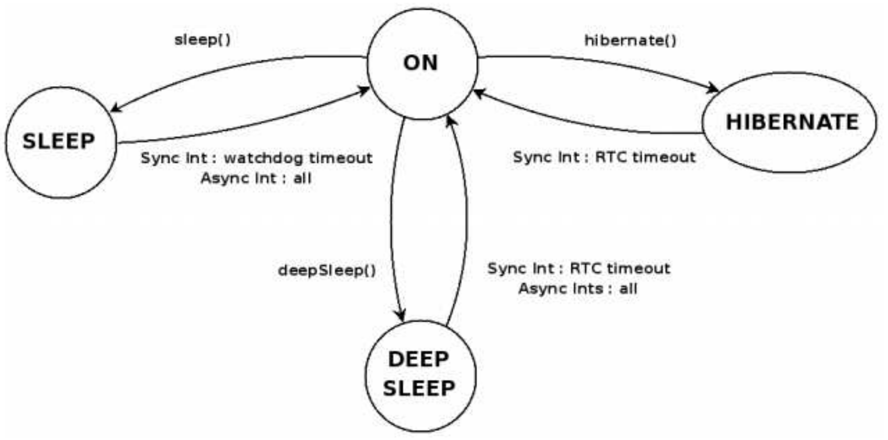


Figure 2.15. *Diagram of operational modes in Waspmote*

Mode	Consumption	Micro	Cycle	Accepted Interruptions
ON	9 mA	ON	-	Synchronous and Asynchronous
Sleep	62 μA	ON	32ms-8s	Watchdog and Asynchronous
Deep Sleep	62 μA	ON	8s-min/hours/days	Synchronous (RTC) and Asynchronous
Hibernate	0.7 μA	OFF	8s-min/hours/days	Synchronous (RTC)

Table 2.3. *Features of operational modes*

2.12.1. Sleep.

Sensor interruption Before setting this state, some interruption by sensor should be programmed to be able to wake up the microcontroller from this state.

```
{
 PWR.sleep(UART0_OFF | BAT_OFF);
// Sleep switching off UART0 and battery monitor
}
```

Watchdog interruption It enables Watchdog interruption to be able to wake the microcontroller up after the time specified. Possible interval values goes from 32ms to 8s.

```
{
 PWR.sleep(WTD_32MS, ALL_OFF);
// Sleep switching off all the modules, waking up after 32 ms
}
```

2.12.2. Deep Sleep.

It enables RTC interruption to be able to wake the microcontroller up when the RTC alarm is launched.

```
{
 PWR.deepSleep(00:00:00:10,RTC_OFFSET,RTC_ALM1_MODE2,ALL_OFF);
  // Sleep switching all off, waking up after 10 seconds
 PWR.deepSleep(15:17:00:00,RTC_ABSOLUTE,RTC_ALM1_MODE2,ALL_OFF);
  // Sleep switching all off, waking up at 17:00 on day 15th.
}
```

2.12.3. Considerations.

When we set an operational mode in Waspmote, we have to take into account that **Waspmote could be in an operational mode but other devices as XBee could be in another one**.

2.13. SD Memory Card

Waspmote has external storage support such as SD (Secure Digital) cards. These micro-SD cards are used specifically to reduce board space to a minimum.

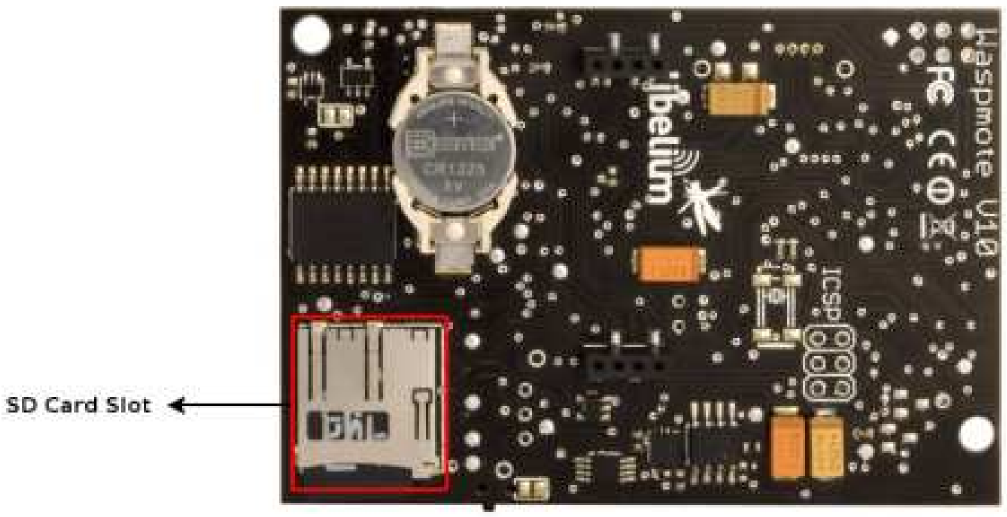


Figure 2.16. *SD card slot*

Waspmote uses the FAT16 file system and can support cards up to 2GB . The information that Waspmote stores in files on the SD can be accessed from different operating systems such as Linux, Windows or Mac-OS. There are many SD card models; any of them has defective blocks, which are ignored when using the Waspmote's SD library. However, when using OTA (Programming Over The Air), those SD blocks cannot be avoided, so that the execution could crash (see appendix C). Libelium implements a special process to ensure the SD cards we provide will work fine with OTA.

The only SD cards that Libelium can assure that work correctly with Waspmote are the SD cards we distribute officially. Although, the SD cards of the brand Trascend works properly.

To communicate with the SD module we use the SPI bus. This bus is a communications standard used to transfer information between electronic devices which accept clock regulated bit flow. The SPI includes lines for the clock, incoming data and outgoing data, and a selection pin. The SD card is powered through a digital pin from the microcontroller. It is not therefore necessary to use a switch to cut the power, putting a low pin value is enough to set the SD consumption to 0 μA.

The limit in files and directories creation per level is 256 files per directory and up to 256 sub-directories in each directory. There is no limit in the number of nested levels.

2.13.1. Starting.

Before start using the SD card, it needs to be initialized. This process checks if an SD card is present on the slot, initializes SPI bus, opens partition and opens root directory. The SD card is powered by a microcontroller pin, so this pin needs to be declared as an output and give a high value. After the SD card has been powered up, it needs to be initialized. A function has been developed for this task, that initializes the use of SD cards, looks into the root partition, opens the file system and initializes the public pointer that can be used to access the file system.

```
{
 SD.begin(); // Declare microcontroller pins
 SD.setMode(SD_ON); // Power SD Card up
 SD.init(); // Executes the init process
}
```

The function SD.init() updates the flag called SD.flag which save a code error. This flag shows the state of the SD card during initialization and operation. Possible values are:
• 0 : nothing failed, all the processes has been executed properly.
• 1 : no SD card in the slot.
• 2 : initialization failed.

With the function SD.isSD() we can know if there is a SD in card slot. It returns 1 if SD card is present and 0 if not.

When we want to closes the directory, file system and partition pointers. We use the following code:

```
{
SD.close();
}
```

This function closes all the pointers in use in the library, so that they can be reused again after a call to init function. It becomes very useful if e.g. the card is removed and inserted again.

In another case, if we want to power down the SD card, then:

```
{
SD.close();
SD.setMode(SD_OFF); // Powers SD down
}
```

2.13.2. SD card information.

When SD has been initialized properly, pointers can be used to access partition, file system and root directory. There are some functions that return information about the SD card.

```
{
 // Gets disk info, returning it and storing this info.
 char* diskInfo;
 diskInfo=SD.print_disk_info();
 // Get total size of a SD card
 offset_t diskSiZe;
 diskSize=SD.getDiskSize();
 // Get available space of SD card
 offset_t diskFree;
 diskFree=SD.getDiskFree();
}
```

2.13.3. Directory operations.

The principal idea of working with directories is that we are working with pointers. You can notice that the notation is very similar to Linux.

```
{
  // Creates a directory, in the current directory,
  // called ''francisco".
  uint8_t dirCreation;
  // It returns 1 on creation and 0 on error,
  //activating the flag too.
  dirCreation=SD.mkdir("francisco");
 // Deletes the directory, in the current directory,
 // called "francisco".
 uint8_t delState;
   // It returns 1 if the directory is erased properly,
  //and 0 if error.
 delState=SD.del("francisco");
}
```

NOTE: SD.del() deletes a directory, including all the files inside the directory, but if there are some sub-directories an error will be returned.

To organize an SD card, it is possible to create and manage directories. There are some functions related with directories.

Changing directory.

```
{
 // cdState=1-> correct , cdState=0-> error .
 uint8_t cdState;
 // Change to directory called "francisco".
 cdState=SD.cd("francisco");
}
```

NOTE: In root directory it has no sense changing directory to '..', so function will return error when doing that.

Finding directory.

If it exists and it is a directory '1' will be returned, '0' will be returned if it exists but it is not a directory and '-1' will be returned if it does not exist.

```
{
 int8_t isdir;
 // Checks the existence of francisco in the current directory
 isdir=SD.isDir("francisco");
}
```

2.13.4. Files operations.

To store data, files can be created and managed. There are some functions related to files operations.

Creating and deleting files.

These operations return 1, if the operation takes place satisfactory and 0 if not.

```
{
 uint8_t fileCreation;
 fileCreation=SD.create("medidas");
 uint8_t fileDelete;
 fileDelete=SD.del("medidas_antiguas");
}
```

Opening and closing files.

```
{
 struct fat_file_struct*fp;
 fp=SD.openFile("file"); // Opens file and assigns 'fp' to it.
 SD.closeFile(fp);
}
```

NOTE: If a file is opened with previous function and it is not closed before using another file function, it will not work properly. Only one file pointer can be managed at the same time.

Reading and writing data.

```
char* dataRead;
//It stores in "buffer"17 characters after jumping 3
dataRead=SD.cat("medidas_antiguas",3,17);
//It stores in 'buffer' 100 characters from the beginning
dataRead=SD.cat("medidas_antiguas",0,100);
//It stores in "buffer"3 lines after jumping over the 2 first
dataRead=SD.catln("medidas_antiguas",2,3);
uint8_t writeState;
//It writes "hello" on file at position 0
writeState=SD.writeSD("francisco","hello",0);
//It writes "hello" on file at position 20
writeState=SD.writeSD("francisco","hello",20);
//It writes "hello" on file at position 20
writeState=SD.writeSD("francisco","hello",20,3);
//It writes "hello" at the end of file
writeState=SD.append("francisco","hello");
//It writes "hello" at end of file with EOL
writeState=SD.appendln("francisco","hello");
}
```

2.14. Sensors.

2.14.1. Temperature.

The Waspmote RTC (DS3231SN from Maxim) has a built in internal temperature sensor which it uses to recalibrate itself. Waspmote can access the value of this sensor through the I2C bus.

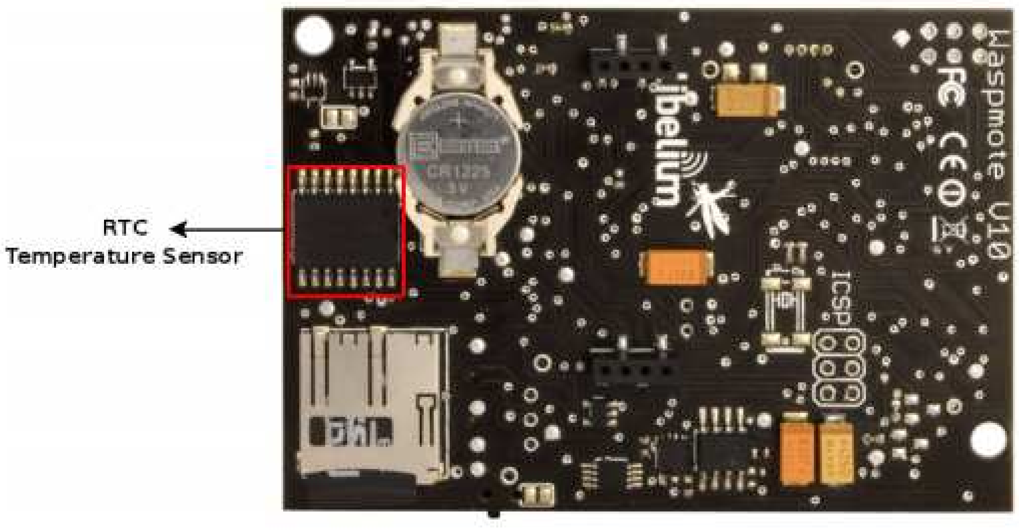


Figure 2.17. *Temperature sensor*

The sensor is shown in a 10-bit two complement format. It has a resolution of 0.25 grades centigrades. The measurable temperature range is between -40 grades centigrades C and +85 grades centigrades. The following example gets temperature from the RTC. It reads associated registers to temperature and stores the temperature in a variable called temperature. If temperature is negative, a variable called tempNegative is set to TRUE.

```
{
  uint8_t temperature=0;
  temperature=RTC.getTemperature(); // Gets temperature
}
```

2.14.2. Accelerometer.

Waspmote has a built in acceleration sensor LIS3LV02DL from STMicroelectronics which informs the mote of acceleration variations experienced on each one of the 3 axes (X,Y,Z). The integration of this sensor allows the measurement of acceleration on the 3 axes (X,Y,Z), establishing 2 kind of events: Free Fall and Direction Detection Change.

The LIS3lV02DL is defined as a 3-axis linear accelerometer which allows measurements to be taken from the sensor through the I2C interface. The accelerometer has a 12 bit resolution (4096 possible values). Another of the parameters that can be adjusted is the accelerometer's refresh rate, i.e. how often internal log updates are required. This allows different bandwidths to be used. The higher the refresh rate, the higher the consumption, but there is also greater accuracy and more real data are obtained.

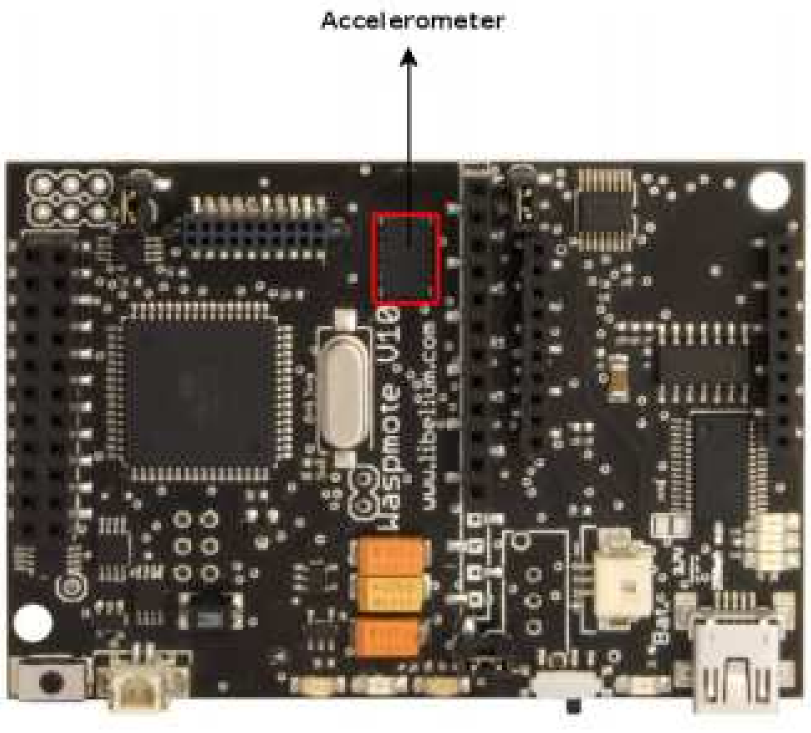
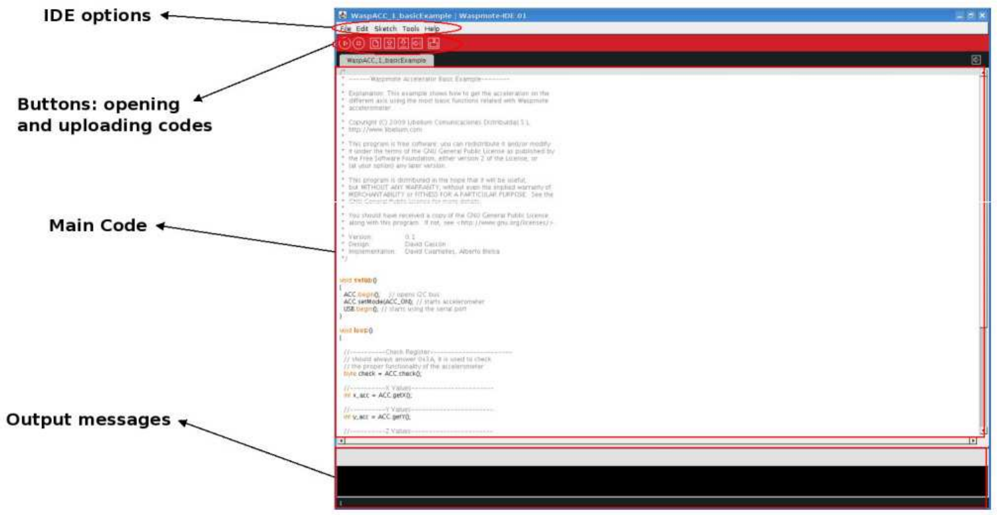


Figure 2.18. *Accelerometer*

```
void setup()
{
 ACC.close(); // Closes I2C bus and accelerometer hibernate mode
 ACC.ON(); // Initializes the accelerometer
 ACC.reboot(); // Executes the reboot process
}
void loop()
{
 acc.getX(); // Gets acceleration on X axis.
 acc.getY(); // Gets acceleration on Y axis.
 acc.getZ(); // Gets acceleration on Z axis.
}
```

2.15. Waspmote IDE

Waspmote IDE is a development environment based on Arduino [1] IDE.


IDE options
Buttons: opening and uploading codes
Main Code
Output messages
WaspACC_1_basicExample | Waspmote-IDE 01
File Edit Sketch Tools Help
WaspACC_1_basicExample

```
/*
 *  ------Waspmote Accelerator 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) 2009 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 2 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 Cuartielles
 *  Implementation:    David Cuartielles, Alberto Bielsa
 */

void setup()
{
  ACC.begin();    // opens I2C bus
  ACC.setMode(ACC_ON); // starts accelerometer
  USB.begin(); // starts using the serial port
}

void loop()
{

  //-------------Check Register---------------------------
  // should always answer 0x3A, it is used to check
  // the proper functionality of the accelerometer
  byte check = ACC.check();

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

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

  //-------------Z Values---------------------------
```

Figure 2.19. *Waspmote IDE overview*

• **IDE options:** this part is the configuration menu. It allows the Waspmote IDE configuration as well as the board or port we are using.

• **Buttons:** this part is a button menu for compiling, opening, storing or uploading a code to the board.The following figure shows the different buttons and their function.

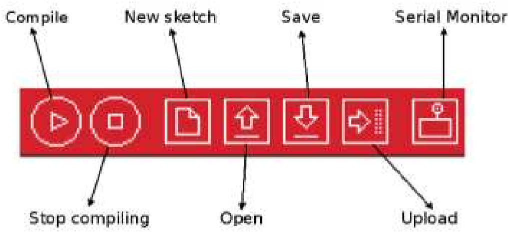


Figure 2.20. *Buttons menu*

• **Main Code:** this part contains the main code that will be uploaded to Waspmote board, following a structure divided in two parts: 'setup' and 'loop'. 'setup' is only executed once, when the program starts, and 'loop' is executed indefinitely.

[1] Arduino is an open-source physical computing platform based on a simple i/o board and a development environment that implements the Processing/Wiring language.

• **Output messages:** this part shows some output messages as error messages (compilation or upload) or success messages (compiling or uploading properly done).

2.16. Programming a Waspmote.

The steps to program a Waspmote are as follows:

Step 1: Remove the Xbee card from the Waspmote: the USB used to connect the Waspmote to the PC and the Xbee card use the same bus. You can use one at a time (USB or Xbee).

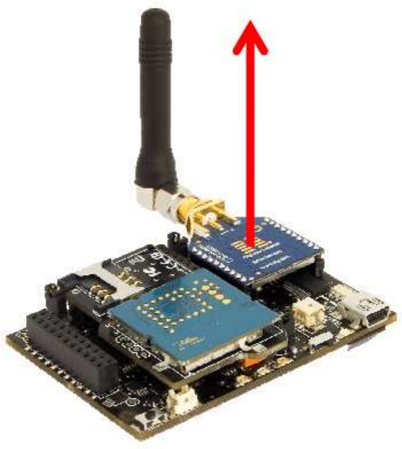


Figure 2.21. *Step 1*

Step 2: Connect the Waspmote to the PC using the provided USB cable.

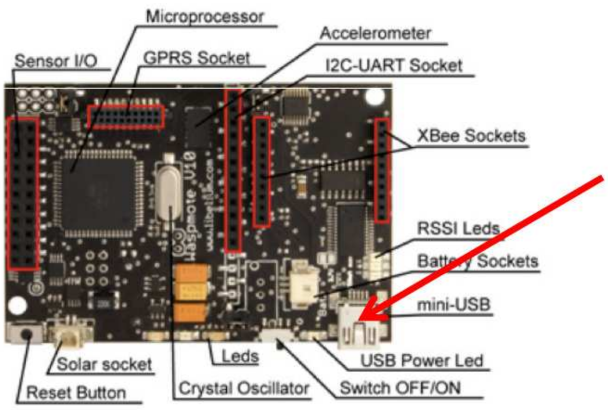


Figure 2.22. *Step 2*

Step 3: Switch ON the Waspmote.

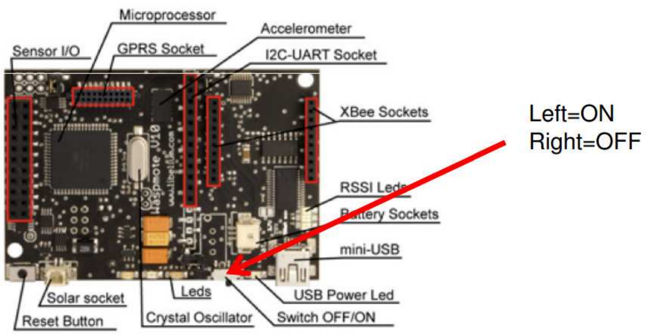
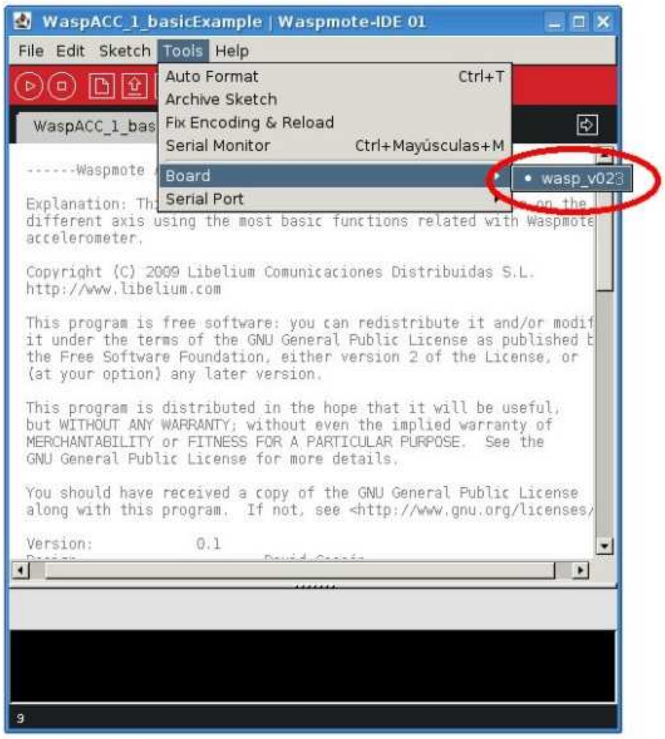


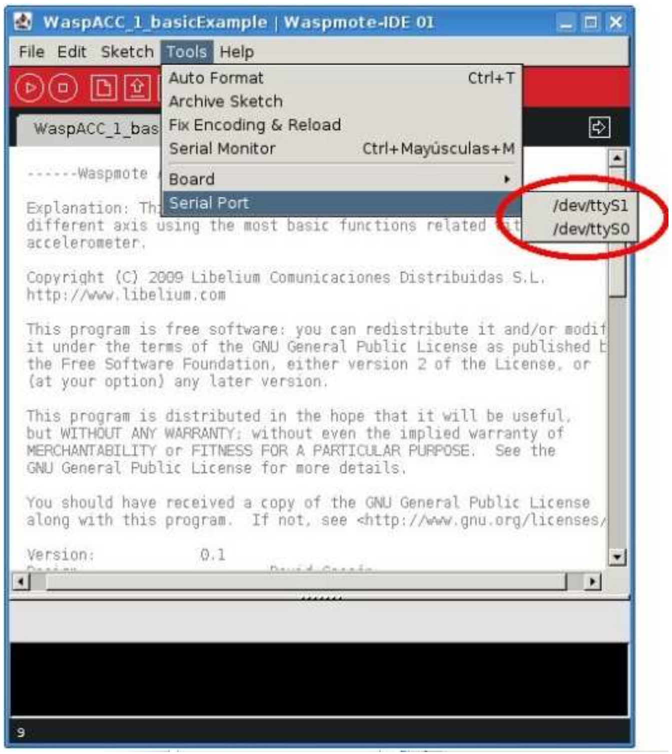
Figure 2.23. *Step 3*

Step 4: Open the Waspmote IDE. Select the Tools menu and choose the Wasp board (WASP v01) and the correct serial port (tty/USBx on Linux, COMx on Windows).


WaspACC_1_basicExample | Waspmote-IDE 01
File Edit Sketch Tools Help
WaspACC_1_bas
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Mayúsculas+M
Board
Serial Port
• wasp_v023

```
------Waspmote

Explanation: Th                                   on the
different axis using the most basic functions related with Waspmote
accelerometer.

Copyright (C) 2009 Libelium Comunicaciones Distribuidas S.L.
http://www.libelium.com

This program is free software: you can redistribute it and/or modif
it under the terms of the GNU General Public License as published b
the Free Software Foundation, either version 2 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
```

Figure 2.24. *Step 4.1*



```
------Waspmote

Explanation: Th
different axis using the most basic functions related
accelerometer.

Copyright (C) 2009 Libelium Comunicaciones Distribuidas S.L.
http://www.libelium.com

This program is free software: you can redistribute it and/or modif
it under the terms of the GNU General Public License as published b
the Free Software Foundation, either version 2 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
```

Figure 2.25. *Step 4.2*

Step 5: Write the source code. It will appear in the IDE window.

Step 6: Click on the upload button and wait until the code has been compiled and uploaded.

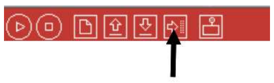
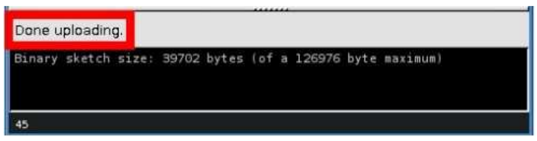


Figure 2.26. *Step 6: Upload button*

At the end, you will see:



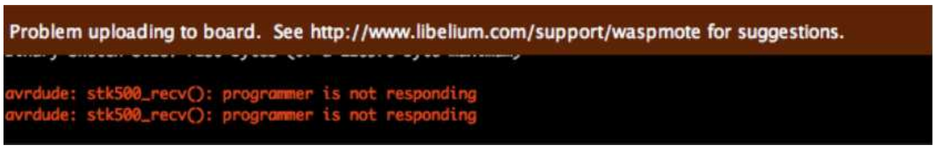
```
Done uploading.
Binary sketch size: 39702 bytes (of a 126976 byte maximum)
45
```

Figure 2.27. *Step 6: Done uploading*

Step 7: Reset the board.

2.17. Errors while you are uploading a program.

If you get an error like the following, then switch the Waspmote on!



```
Problem uploading to board.  See http://www.libelium.com/support/waspmote for suggestions.

avrdude: stk500_recv(): programmer is not responding
avrdude: stk500_recv(): programmer is not responding
```

Figure 2.28. *IDE Error*

Another type of error are as follows:
avrdude:stk500-getsync():not in sync:resp=0xXX
avrdude:stk500-disable():protocol error ,expect=0xXX, resp=0xXX

Where XX is a different number in each case. Answer the following questions to solve the problem.
1) Have you switched Waspmote on?
2) Have you removed the XBee module? XBee module and USB share the same UART, so the code will not be uploaded if the XBee is on Waspmote.
3) Are all the jumpers in their positions? There are three jumpers that must be connected when programming the board.
4) If the previous questions don't help you, please try to uninstall the FTDI drivers and install them again.

2.18. Application Programming Interface.

An API (Application Programming Interface) has been developed to facilitate applications programming using Waspmote. This API includes all the modules integrated in Waspmote, as well as the handling of other functionalities such as interruptions or the different energy modes.The API has been developed in C/C++ , structured in the following way:
•**General configuration**: WaspClasses.h, WaspVariables.h,WaspConstants.h, Wconstants.h, pins_arduino.h, pins_arduino.c, utils.h, utils.cpp, WProgram.h,Wrandom
•**Shared**: binary.h, byteordering.h, byteordering.cpp, HardwareSerial.h, HardwareSerial.cpp, WaspRegisters.h,WaspRegisters.c, wiring_analog.c, wiring.h, wiring.c, wiring_digital.c, wiring_private.h, wiring_pulse.c, wiring_serial.c, wiring_shift.c
•**SD Storage**: fat_config.h, fat.h, fat.cpp, partition_config.h, partition.h, partition.cpp, sd_raw_config.h, sd_raw.h, sd_raw.cpp, sd_reader_config.h, WaspSD.h, WaspSD.cpp
•**I2C Communication**: twi.h, twi.c, wire.h, wire.cpp
•**Accelzrometer**: WaspACC.h, WaspACC.cpp
•**GSM - GPRS/3G**: WaspGPRS.h, WaspGPRS.cpp, WaspGPRSConstants.h

•**GPS**: WaspGPS.h, WaspGPS.cpp
•**Energy Control**: WaspPWR.h, WaspPWR.cpp
•**RTC**: WaspRTC.h, WaspRTC.cpp
•**Sensors**: WaspSensorEvent.h, WaspSensorEvent.cpp, WaspSensorGas.h, WaspSensorGas.cpp, WaspSensorPrototyping.h, WaspSensorPrototyping.cpp
•**USB**: WaspUSB.h, WaspUSB.cpp
•**XBee**: WaspXBee.h, WaspXBee.cpp, WaspXBeeConstants.h, WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBee802.h, WaspXBee802.cpp, WaspXBeeZB.h, WaspXBeeZB.cpp, WaspXBeeDM.h, WaspXBeeDM.cpp, WaspXBee868.h,WaspXBee868. cpp, WaspXBeeXSC.h, WaspXBeeXSC.cpp
•**Interruptions**: Winterruptions.c

2.19. How to include new API versions to Waspmote IDE?

1. Copy your own API version or the last updated release to the following directory: .../waspmote-ide-v.02-windows/hardware/cores

2. Open the following file: .../waspmote-ide-v.02-windows/hardware/boards. The following example explains how to update from API v.022 to v.023:

```
##########################################################

wasp01.name=waspmote-api-v.022

wasp01.upload.protocol=stk500
wasp01.upload.maximum_size=122880
wasp01.upload.speed=38400

wasp01.bootloader.low_fuses=0xdf
wasp01.bootloader.high_fuses=0xd0
wasp01.bootloader.extended_fuses=0xf5
wasp01.bootloader.path=wasp_v8.1
wasp01.bootloader.file=ATmegaBOOT_v0003.hex
wasp01.bootloader.unlock_bits=0x3F
wasp01.bootloader.lock_bits=0x0F

wasp01.build.mcu=atmega1281
wasp01.build.f_cpu=8000000L
wasp01.build.core=waspmote-api-v.022
```

You have to replace the lines in the code where it appears v.022 and you should set v.023:

```
##############################################################

wasp01.name=waspmote-api-v.023

wasp01.upload.protocol=stk500
wasp01.upload.maximum_size=122880
wasp01.upload.speed=38400

wasp01.bootloader.low_fuses=0xdf
wasp01.bootloader.high_fuses=0xd0
wasp01.bootloader.extended_fuses=0xf5
wasp01.bootloader.path=wasp_v8.1
wasp01.bootloader.file=ATmegaBOOT_v0003.hex
wasp01.bootloader.unlock_bits=0x3F
wasp01.bootloader.lock_bits=0x0F

wasp01.build.mcu=atmega1281
wasp01.build.f_cpu=8000000L
wasp01.build.core=waspmote-api-v.023
```

3. When boards.txt is saved with the new configuration, you must open a new Waspmote IDE window in order to see these changes.