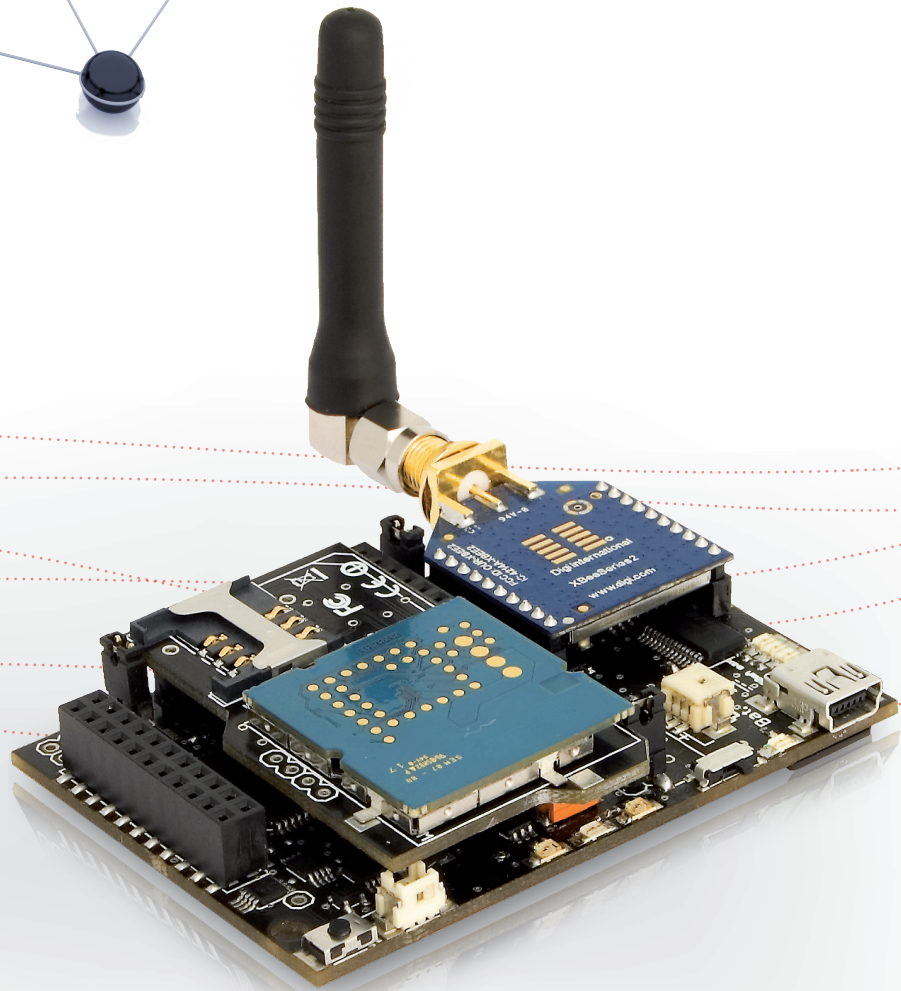
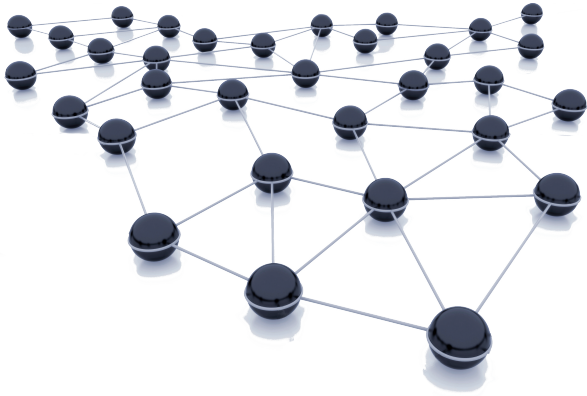


# Wasp mote Interruptions Programming Guide



Document Version: v0.1 -10/2009  
© Libelium Comunicaciones Distribuidas S.L.

# INDEX

<b>1. General Considerations.....</b>	<b>3</b>
1.1. Wasmote Libraries .....	3
1.1.1. Wasmote Interruptions Files .....	3
1.1.2. Structure .....	3
<b>2. Functions.....</b>	<b>3</b>
2.1. Attaching interruptions.....	3
2.2. Detaching interruptions.....	3
2.3. Enabling interruptions.....	3
2.4. Disabling interruptions.....	4
2.5. High Activate Subroutines .....	4
2.6. Low Activate Subroutines.....	4
2.7. Clearing 'intFlag' .....	4
<b>3. Code examples and extended information .....</b>	<b>5</b>

# 1. General Considerations

## 1.1. Waspote Libraries

### 1.1.1. Waspote Interruptions Files

Winterrupts.c

### 1.1.2. Structure

The functions used to manage interruptions are stored in 'Winterrupts.c'. This C file has no header file associated, but the functions are declared in 'wiring.h'. Since it is a C file, no constructor is needed and no functions declared in other C++ Waspote libraries can be used, only the libraries developed in C.

## 2. Functions

### 2.1. Attaching interruptions

It attaches the interruption to the corresponding microcontroller pin and subroutine associated to it.

This function returns nothing.

This function is used internally by 'enableInterrupts'.

Example of use

```
{
  attachInterrupt(ACC_INT_ACT, onHAIwakeUP, HIGH); // Attach Interrupt on pin
  'ACC_INT_ACT' using subroutine 'onHAIwakeUP' on 'HIGH' level
}
```

### 2.2. Detaching interruptions

It detaches the interruption from the corresponding microcontroller pin.

This function returns nothing.

This function is used internally by 'disableInterrupts'.

Example of use

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

### 2.3. Enabling interruptions

It enables the specified interruption by 'conf' input.

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

Example of use

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

## 2.4. Disabling interruptions

It disables the specified interruption by 'conf' input.

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.

Example of use

```
{
  disableInterrupts(ACC_INT); // Disables the specified interruption
}
```

## 2.5. High Activate Subroutines

It setups the default interrupt for the High Activate Interrupts. There are some modules that activates interruptions on HIGH. These modules are Accelerometer, UART1 (GPRS), RTC and other additional modules which can be interrupted like sensors.

This subroutine is executed when the interruption is detected on the corresponding microcontroller pin.

We have to check 'intConf' flag and the monitoring pin, to know what module has activated the interruption, due to more than one module is multiplexed on the same microcontroller pin. When the two conditions match, 'intFlag' is activated on the correct position to show the module that has activated the interruption.

A counter called 'intCounter' is incremented each time an interruption is detected.

A counter array called 'intArray' is incremented in the correct position to know how many times a module has activated the interruption.

## 2.6. Low Activate Subroutines

It setups the default interrupt for the Low Activate Interrupts. There are some modules that activate interruptions on LOW. These modules are Low Battery, internal Watchdog and other additional modules which can be interrupted like sensors.

This subroutine is executed when the interruption is detected on the corresponding microcontroller pin.

We have to check 'intConf' flag and the monitoring pin, to know what module has activated the interruption, due to more than one module is multiplexed on the same microcontroller pin. When the two conditions match, 'intFlag' is activated on the correct position to show the module that has activated the interruption.

A counter called 'intCounter' is incremented each time an interruption is detected.

A counter array called 'intArray' is incremented in the correct position to know how many times a module has activated the interruption.

## 2.7. Clearing 'intFlag'

It clears the global flag 'intFlag'.

Example of use

```
{
  clearIntFlag(); // Clears 'intFlag'
}
```

### 3. Code examples and extended information

For more information about the WaspMote hardware platform go to the Support section:

**<http://www.libelium.com/support/waspmote>**

Extended information about the API libraries and complete code examples can be found at:

**<http://www.libelium.com/development/waspmote>**