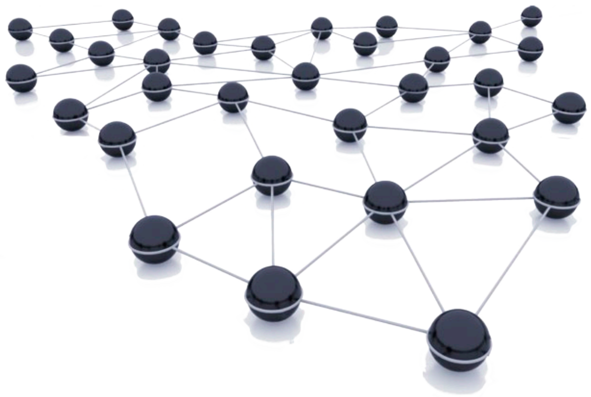
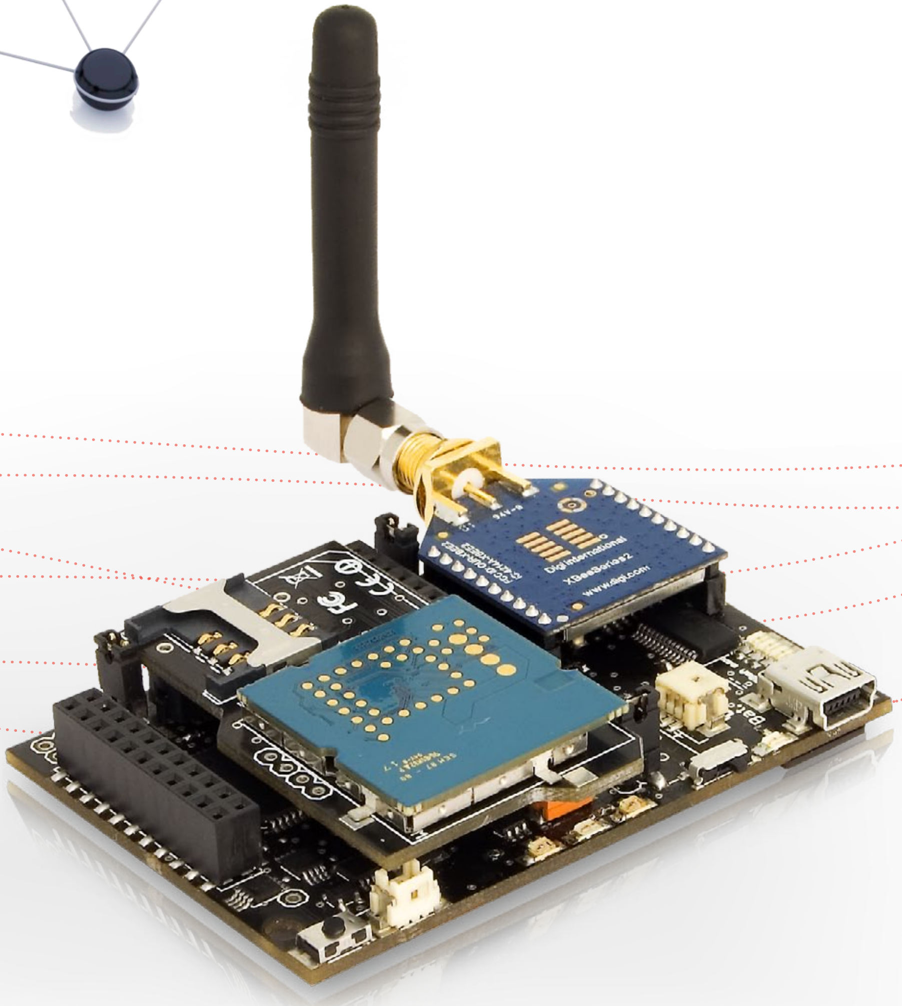
Waspmote RTC

Programming Guide





Document Version: v0.4 - 06/2012

INDEX

1. General Considerations

1.1. Waspmote Libraries

1.1.1. Waspmote RTC Files

WaspRTC.h ; WaspRTC.cpp

1.1.2. Constructor

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

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

1.1.3. Pre-Defined Constants

There are some constants defined in 'WaspRTC.h' to help with the comprehension of the code when reading the first times. These constants are related with RTC register addresses, some functions, inputs and alarm modes.

1.1.4. Variables

Some variables have been defined for storing time, date and alarm data. These variables have been named after the data they store (i.e. RTC.year stores the year, and RTC.month stores the month).

The Waspmote RTC allows setting the day of the week, which is a number between 1-7. This part of the date can be modified by the user providing they are sequential, it meaning that if Sunday is equal to 1, Monday must be equal to 2.

An array called 'registersRTC' has been created for storing the data to send to the RTC. This array is used in each writing or reading operation.

1.1.5. Flags

There are various flags used for handling interruptions while using Waspmote RTC.

1.1.5.1. Global Interruption Flag

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

1.1.5.2. Global Interruption Flag Array

It is used to store the number of times each interruption has been detected. It is used in this library and other libraries which generate interrupts too.

1.1.5.3. Global Interruption Counter

It is used to store the number of interruptions detected. It is used in this library and other libraries which generate interrupts too.

2. Initialization

Before getting information from the RTC, I2C bus has to be initialized.

2.1. Initializing the RTC

It powers the RTC up and initializes I2C bus for communicating with the RTC. It reads from the RTC time, date and alarms, setting the corresponding variables.

It returns nothing and modifies no flag.

Example of use

```
{
 RTC.ON(); // Executes the init process
}
```

2.2. Saving Battery

The RTC is powered by the main battery through a microcontroller pin and uses an auxiliary battery when the main power source is turned OFF in the Hibernate mode, guaranteing its non-stop running.

When the RTC is powered by the auxiliary battery, only the internal oscillator is enabled, using just 0,7uA. When RTC is powered by microcontroller, the power consumption is around 150-200uA. Because of that, when the RTC is only counting time or waiting to launch an alarm, it is recommended to power it down and use the auxiliary battery.

It is recommended to set the RTC off when it is not going to be used. This operation can be made using a function developed for that purpose.

Example of use

```
{
 RTC.setMode(RTC_ON); // Powers RTC UP
 RTC.setMode(RTC_OFF); // Powers RTC Down
}
```

2.3. Switching RTC Off

It turns it off and closes the I2C bus.

It returns nothing and modifies no flag.

Example of use

```
{
 RTC.OFF(); // Turns it off and closes the I2C bus
}
```

3. Setting Time and Date

First step to be able to use the RTC is setting time and date. Some functions have been created for managing these operations.

3.1. Setting Time and Date

It sets time and date on the RTC.

Time and Date are specified by the inputs, which corresponds to the different variables: year, month, date, day, hour, minute and second.

'day' is the day of the week, having a value between 1-7. By default, Sunday is equal to 1 and Saturday is equal to 7.

This function extracts each part of date and time, storing each part in the corresponding variable. After this, 'registersRTC' array is loaded with these variables and data is sent to the RTC.

It returns nothing and modifies no flag.

Example of use

```
{
  RTC.setTime("09:06:29:02:10:00:00"); // Set time and date on the RTC
}
```

Related Variables

year, month, hour, minute, second → stores the time and date set
day → stores the day of the week
date → stores the day of the month

3.2. Getting Time and Date

It gets time and date, storing them in the 'registersRTC' array and the corresponding variables.

It returns a string containing time and date in the following format: "YY:MM:DD:dow:hh:mm:ss".

Example of use

```
{
  char* time_date;
  time_date =RTC.getTime(); // Gets time and date from the RTC
}
```

Related Variables

year, month, hour, minute, second → stores the time and date set
day → stores the day of the week
date → stores the day of the month

3.3. Setting Time and Date from the GPS

It sets time and date on the RTC getting the data from the GPS.

Time and Date are specified by the values returned by the GPS when we call the function 'GPS.getPosition'.

'day' is the day of the week, having a value among 1-7. By default, Sunday is equal to 1 and Saturday is equal to 7. This function set this value to '1'.

This function extracts each part of date and time, storing each part in the corresponding variable. After this, 'registersRTC' array is loaded with these variables and data is sent to the RTC.

It returns nothing and modifies no flag.

Example of use

```
{
 RTC.setTimeFromGPS(); // Sets time and date on the RTC from the GPS
}
```

Related Variables

year, month, hour, minute, second → stores the time and date set
day → stores the day of the week
date → stores the day of the month

4. Setting Alarms

Waspmote RTC provides two alarms to enable interruptions and wake up external microcontroller.

Alarm1 and Alarm2 differ because of Alarm1 is set by day/date, hour, minutes and seconds and Alarm2 is set by day/date, hour and minutes.

When setting alarms there are three inputs: time, offset and mode.

- Time: represents the time/date for the alarm.
- Offset: represents the two modes for setting the alarm time: offset or absolute. When offset is selected, the input time is added to the actual time in the RTC and the result is set as the alarm time. When absolute is selected, the input time is set as the alarm time.
- Mode: represents the different alarm modes. Alarm1 has 6 modes and Alarm2 has 5 modes.

When the time set in Alarm1 or Alarm2 matches the time on RTC, a pin is enabled to indicate the match. This pin is connected to an interrupt pin on the microcontroller, so as the alarm can wake up the microcontroller from a sleep power mode.

4.1. Setting the Alarm1 (using a string as input)

It sets the Alarm1 to the specified time, offset and mode.

The input 'time' has the following format: "dd:hh:mm:ss".

The input 'offset' has some possible values:

```
RTC_OFFSET: ‘time’ is added to the actual time read from RTC
RTC_ABSOLUTE: ‘time’ is set as the time for Alarm1
```

The input 'mode' specifies the mode for Alarm1. Possible values are:

```
RTC_ALM1_MODE1: Day, hours, minutes and seconds match
RTC_ALM1_MODE2: Date, hours, minutes and seconds match
RTC_ALM1_MODE3: Hours, minutes and seconds match
RTC_ALM1_MODE4: Minutes and seconds match
RTC_ALM1_MODE5: Seconds match
RTC_ALM1_MODE6: Once per second
```

When this function is called, the Alarm1 is set and no more functions need to be executed.

It returns nothing, but when the Alarm1 matches the time, interruption flags will be modified to indicate it.

Example of use

```
{
 RTC.setAlarm1(“29:11:00:00”,RTC_ABSOLUTE,RTC_ALM1_MODE2); // Sets Alarm1
 RTC.setAlarm1(“00:00:05:00”,RTC_OFFSET,RTC_ALM1_MODE2); // Sets Alarm1
}
```

Related Variables

```
day_alarm1→ stores the day or date of the Alarm1
hour_alarm1, minute_alarm1, second_alarm1 → store the time of the Alarm1
```

4.2. Setting the Alarm1

It sets the Alarm1 to the specified time, offset and mode.

The inputs 'day_date', '_hour', '_minute' and '_second' specify the time for the Alarm1.

The input 'offset' has some possible values:

```
RTC_OFFSET: 'time' is added to the actual time read from the RTC
RTC_ABSOLUTE: 'time' is set as the time for the Alarm1
```

The input 'mode' specifies the mode for the Alarm1. Possible values are:

```
RTC_ALM1_MODE1: Day, hours, minutes and seconds match
RTC_ALM1_MODE2: Date, hours, minutes and seconds match
RTC_ALM1_MODE3: Hours, minutes and seconds match
RTC_ALM1_MODE4: Minutes and seconds match
RTC_ALM1_MODE5: Seconds match
RTC_ALM1_MODE6: Once per second
```

When this function is called, the Alarm1 is set and no more functions need to be executed.

It returns nothing, but when the Alarm1 matches the time, interruption flags will be modified to indicate it.

Example of use

```
{
  RTC.setAlarm1(29,11,0,0,RTC_ABSOLUTE,RTC_ALM1_MODE3); // Sets Alarm1
  RTC.setAlarm1(0,0,5,0,RTC_OFFSET,RTC_ALM1_MODE4); // Sets Alarm1
}
```

Related Variables

```
day_alarm1 → stores the day or date of Alarm1
hour_alarm1, minute_alarm1, second_alarm1 → store the time of Alarm1
```

4.3. Getting the Alarm1

It gets the Alarm1 time from RTC.

It returns a string containing this time and date for the Alarm1.

Example of use

```
{
  char*  time_alarm1;
  RTC.getAlarm1(); // Gets time for Alarm1
}
```

Related Variables

```
day_alarm1 → stores the day or date of the Alarm1
hour_alarm1, minute_alarm1, second_alarm1 → store the time of the Alarm1
```

4.4. Setting the Alarm2 (using a string as input)

It sets the Alarm2 to the specified time, offset and mode.

The input 'time' has the following format : "dd:hh:mm".

The input 'offset' has some possible values:

```
RTC_OFFSET: 'time' is added to the actual time read from the RTC
RTC_ABSOLUTE: 'time' is set as the time for the Alarm2
```

The input 'mode' specifies the mode for the Alarm2. Possible values are:

```
RTC_ALM2_MODE1: Day,hours and minutes match
RTC_ALM2_MODE2: Date,hours and minutes match
RTC_ALM2_MODE3: Hours and minutes match
RTC_ALM2_MODE4: Minutes match
RTC_ALM2_MODE5: Once per minute
```

When this function is called, the Alarm2 is set and no more functions need to be executed.

It returns nothing, but when the Alarm2 matches the time, interruption flags will be modified to indicate it.

Example of use

```
{
  RTC.setAlarm2("29:11:00",RTC_ABSOLUTE,RTC_ALM2_MODE2); // Sets Alarm2
  RTC.setAlarm2("00:00:05",RTC_OFFSET,RTC_ALM2_MODE2); // Sets Alarm2
}
```

Related Variables

```
day_alarm2 → stores the day or date of Alarm2
hour_alarm2, minute_alarm2 → store the time of Alarm2
```

4.5. Setting the Alarm2

It sets the Alarm2 to the specified time, offset and mode.

The inputs 'day_date', '_hour' and '_minute' specify the time for the Alarm2.

The input 'offset' has some possible values:

```
RTC_OFFSET: 'time' is added to the actual time read from RTC
RTC_ABSOLUTE: 'time' is set as the time for Alarm2
```

The input 'mode' specifies the mode for the Alarm2. Possible values are:

```
RTC_ALM2_MODE1: Day,hours and minutes match
RTC_ALM2_MODE2: Date,hours and minutes match
RTC_ALM2_MODE3: Hours and minutes match
RTC_ALM2_MODE4: Minutes match
RTC_ALM2_MODE5: Once per minute
```

When this function is called, Alarm2 is set and no more functions need to be executed.

It returns nothing, but when Alarm2 matches the time, interruption flags will be modified to indicate it.

Example of use

```
    {
     RTC.setAlarm2(29,11,0,RTC_ABSOLUTE,RTC_ALM2_MODE3); // Sets Alarm2
     RTC.setAlarm2(0,0,5,RTC_OFFSET,RTC_ALM2_MODE4); // Sets Alarm2
    }
```

Related Variables

```
     day_alarm2  → stores the day or date of the Alarm2
     hour_alarm2, minute_alarm2 → store the time of the Alarm2
```

4.6. Getting Alarm2

It gets the Alarm2 time from RTC.

It returns a string containing this time and date for the Alarm2.

Example of use

```
    {
     char*  time_alarm2;
     RTC.getAlarm2(); // Gets time for Alarm2
    }
```

Related Variables

```
     day_alarm2 → stores the day or date of the Alarm2
     hour_alarm2, minute_alarm2 → store the time of the Alarm2
```

4.7. Clear Alarm Flags

It clears alarm flags (A1F and A2F) in the RTC.

If these flags are not cleared after an interrupt is captured, no more interrupts could be captured.

Example of use

```
    {
     RTC.clearAlarmFlag(); // Clear Alarm Flags on the RTC
    }
```

5. Getting Temperature

The Waspmote RTC is provided with an internal temperature sensor which can be used to know the temperature in the same board to calibrate its internal oscillator.

5.1. Getting Temperature

It gets temperature from the RTC. It reads associated registers to temperature and stores the temperature in a variable called 'temp'. If temperature is negative, a variable called 'tempNegative' is set to TRUE.

It returns the temperature value.

Example of use

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

Related Variables

```
temp →stores the temperature read from the RTC
tempNegative →TRUE if temperature is negative and FALSE if not
```

6. Using RTC with hibernate

If the hibernate mode is used in a script, RTC alarms must only be used to set the wake up from the hibernate mode. When the hibernate jumper is not connected, any RTC alarm arriving while the code is running could cause internal collisions. The RTC alarm is supposed to happen when Wasp is hibernating.

There are several way to set alternative alarms:

- use the Watchdog
- compare current time and date with previous conditions
- use the function millis()

7. 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