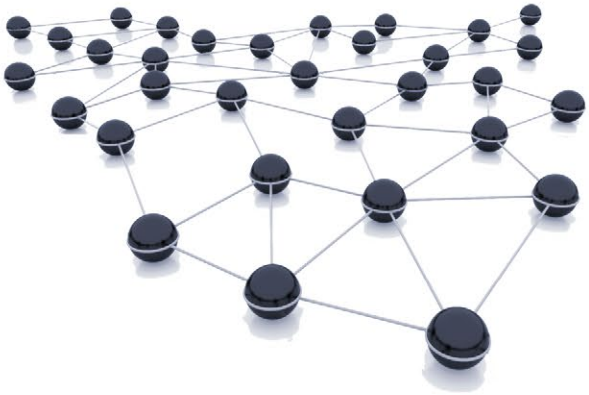
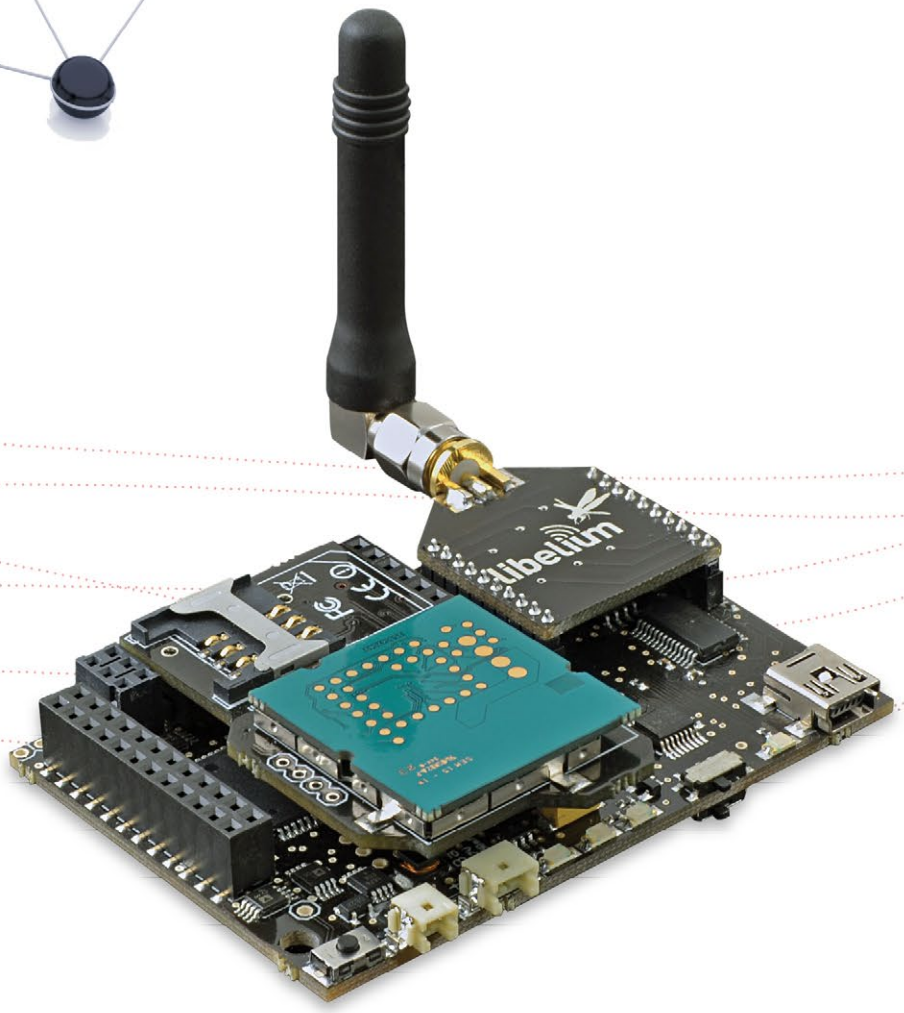
RS-485 Module

Communication Guide





libelium

waspmote

Document version: v4.4 - 01/2016

INDEX

1. Introduction 4
1.1. The standard 4
1.2. Functional features 5
1.3. Connectivity 5

2. Hardware 6
2.1. Electrical Characteristics 6
2.2. Connection Diagram 6
2.3. Consumption 7
2.4. Connector 7

3. Dual Radio with Expansion Board 9
3.1. Expansion Radio Board 9

4. Applications 11

5. Libelium's API 12

6. Library functions 13
6.1. Library constructor 13
6.2. Switching the module ON 13
6.3. Switching the module OFF 13
6.4. Configuring speed communication 14
6.5. Configuring the number of stop bits 14
6.6. Configuring the parity bit 15
6.7. Reset 16
6.8. Sending data 16
6.8.1. Sending a char 16
6.8.2. Sending an integer 17
6.8.3. Sending a string 17
6.8.4. Sending with base 17
6.8.5. Sending a long 18
6.9. Receiving data 18
6.10. Data available 18
6.11. Flush function 19
6.12. Tranmission control 19
6.13. Reception control 19

7. Code examples and extended information 20

8. API changelog 22

9. Documentation changelog 23

1. Introduction

1.1. The standard

RS-485 is the most versatile communication standard. The RS-485 standard defines the electrical characteristics of drivers and receivers for use in digital systems. The standard is published by the Telecommunications Industry Association/Electronic Industries Alliance (TIA/EIA). RS-485 signals are used in a wide range of computer and automation systems and are used in programmable logic controllers and on factory floors. Since it is differential, it resists electromagnetic interference from motors and welding equipment. It may be used to control video surveillance systems or to interconnect security control panels and devices such as access control card readers. It does not specify or recommend any communications protocol. In the next table the electrical characteristics of the standard are defined.

RS-485	
Standard	EIA RS-485
Physical Media	Twisted pair
Network Topology	Point-to-point, multi-dropped, multi-point
Maximum Devices	32 drivers or receivers
Mode of Operation	Differential signaling
Voltage Levels	-7 V to +12 V
Mark (1)	Positive voltages (B-A > +200 mV)
Space (0)	Negative voltages (B-A < -200 mV)
Available Signals	Tx+/Rx+, Tx-/Rx- (Half Duplex), Tx+,Tx-,Rx+,Rx- (Full Duplex)

Figure : RS-485 standard specifications

The number of maximum devices could increase improving some features. The Waspmote RS-485 module uses half duplex communication and the SP3485 integrated circuit wich provides a low consumption and good communication speed.

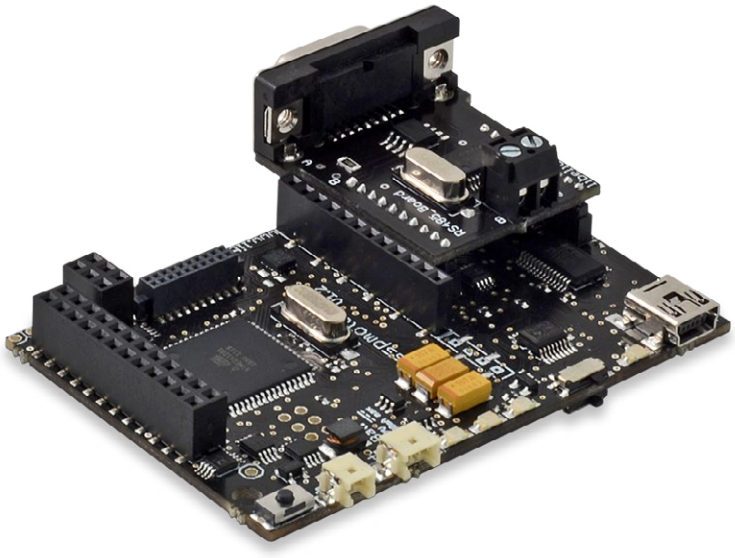


Figure : The RS-485 module on Waspmote

This list includes some of the most common uses of the standard:

- Industrial equipment
- Machine to Machine (M2M) communications
- Industrial Control Systems, including the most common versions of Modbus and Profibus
- Programmable logic controllers
- RS-485 is also used in building automation
- Interconnect security control panels and devices

1.2. Functional features

The differential transmission is the base of the functioning. The same information is sent trough the two wires, but with a phase difference of 180 degrees. Any interference introduced in the signal will affect equally both wires. By reversing the signals interferences are eliminated each other. Another noise immunity is the use of twisted pairs. Twisted pairs in RS-485 communication however adds immunity which is a much better way to fight noise. The resulting noise current is many factors lower than with an ordinary straight cable.

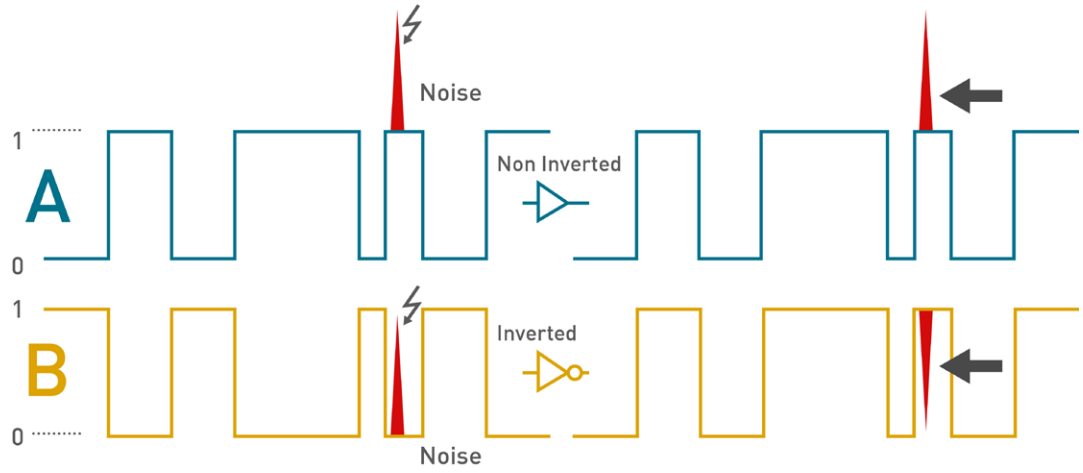


Figure : Differential signaling operation

1.3. Connectivity

Network topology is probably the reason why RS-485 is now the favorite interface in data acquisition and control applications. RS-485 is the only one of the interfaces capable of Internet working multiple transmitters and receivers in the same network. It is possible to connect 32 devices to the network. Currently available high-resistance RS-485 inputs allow this number to be expanded to 256. With the introduction of "automatic" repeaters and high-impedance drivers / receivers this "limitation" can be extended to hundreds (or even thousands) of nodes on a network. In RS-485, the communication is half duplex. It means bidirectional but not simultaneously communication. Only one device may be transmitting at any given time and all other are receiving.

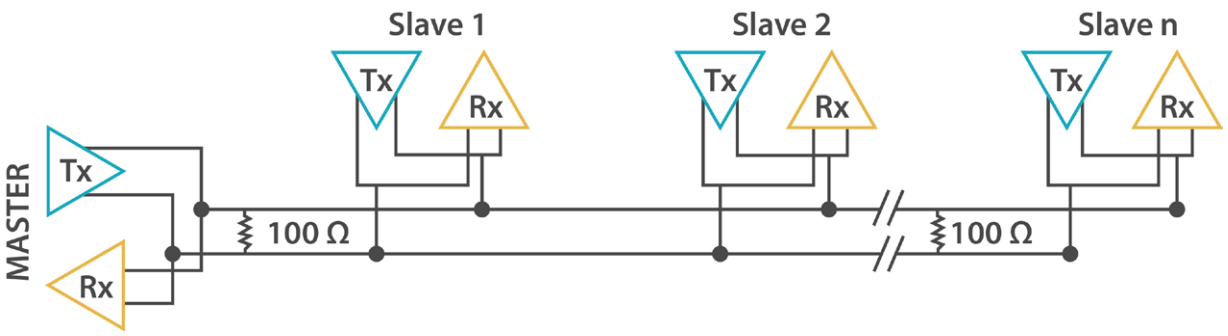


Figure : Multi-drop Network connection

2. Hardware

The RS-485 / Modbus module has been tested with various devices and is compatible with the majority of commercial modules, but this does not ensure the working with all of them. Be sure that the RS-485 module fits your technical requirements. The final user is the responsable to perform the task of communicatiing the RS-485 module with other commercial devices.

2.1. Electrical Characteristics

- **Board power voltages:** 3.3 V
- **Maximum admitted voltage**: -0.3 V to +4 V
- **Max data rate:** 2.5 Mbps
- **Temperature range:** [-40 ºC, 85 ºC]
- **Dimensions:** 33x31.5 mm

2.2. Connection Diagram

The RS-485 module uses the SPI pins for communication. The SPI port allows more speed communication and frees up the Waspmote's UART for other purposes. The Expansion Board allows to connect two communication modules at the same time in the Waspmote sensor platform. This means a lot of different combinations are now possible using any of the radios available for Waspmote (802.15.4, ZigBee, 868, 900, Bluetooth Pro/Low Energy, NFC/RFID, WiFi and GPRS/3G) and the RS-485 module.

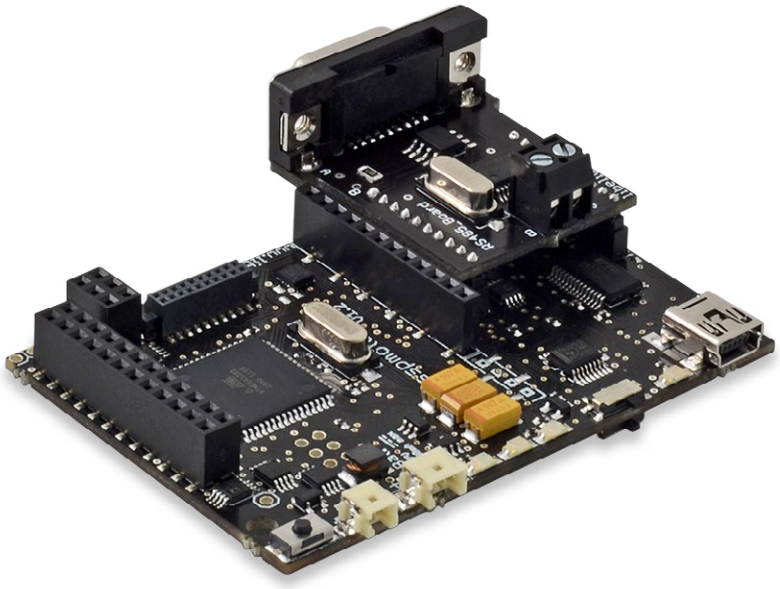


Figure : The RS-485 module in socket 0

> ***Note:*** *The RS-485 module can only be used in special Waspmote units which have been modified to drive the SPI pins to socket 0.*

The RS-485 Serial / Modbus module can be used with two different protocols:

- 1. RS-485 Serial standard (this is the scope of this guide).
- 2. Modbus protocol (which adds some features; see the Modbus Communication Guide for more details).

2.3. Consumption

The RS-485 module uses a low power transceiver. The board is guaranteed to run at data rates of 460800 bps. The typical consumption on the board is 7mA but this consumption can increase, due to current peaks while the module is transmitting data trough the bus. The typical peak current consumption is about 30mA.

2.4. Connector

The RS-485 standard does not specify the type of connector or pin-outs. RS-485 connectors can be DB9, DB25, Terminal Blocks, RJ-11, R-J45 or one of the round DIN connectors. The DB9 connector is one of the most common connectors and is used in many applications because it provides size and cost benefit. The RS-485 module uses pins 2 and 3 to transmit and receive data. Ground pin is not necessary because it is a differential bus.

Figure : DB9 connector

The RS-485 module comes with a standard male-female DB9 cable. This cable is useful for connecting the module to other RS-485 devices which have a DB9 male connector.

Figure : Male-female DB9 cable

Many devices don't have standard connectors. The RS-485 module includes a block connector for making wired connections.

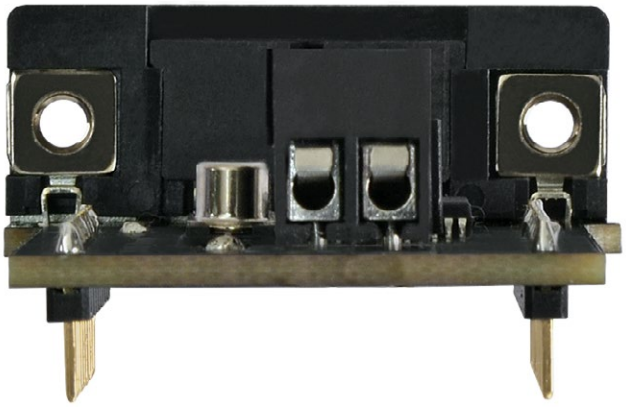


Figure : Terminal block connector

3. Dual Radio with Expansion Board

3.1. Expansion Radio Board

The Expansion Board allows to connect two communication modules at the same time in the Waspmote sensor platform. This means a lot of different combinations are possible using any of the wireless radios available for Waspmote: 802.15.4, ZigBee, DigiMesh, 868 MHz, 900 MHz, LoRaWAN, LoRa, Sigfox, Bluetooth Pro, Bluetooth Low Energy, RFID/NFC, WiFi, GPRS Pro, GPRS+GPS and 3G/GPRS. Besides, the following Industrial Protocols modules are available: RS-485/Modbus, RS-232 Serial/Modbus and CAN Bus.

Some of the possible combinations are:

- LoRaWAN - GPRS
- 802.15.4 - Sigfox
- 868 MHz - RS-485
- RS-232 - WiFi
- DigiMesh - 3G/GPRS
- RS-232 - RFID/NFC
- WiFi - 3G/GPRS
- CAN bus - Bluetooth
- etc.

Remark: GPRS Pro, GPRS+GPS and 3G/GPRS modules do not need the Expansion Board to be connected to Waspmote. They can be plugged directly in the socket1.

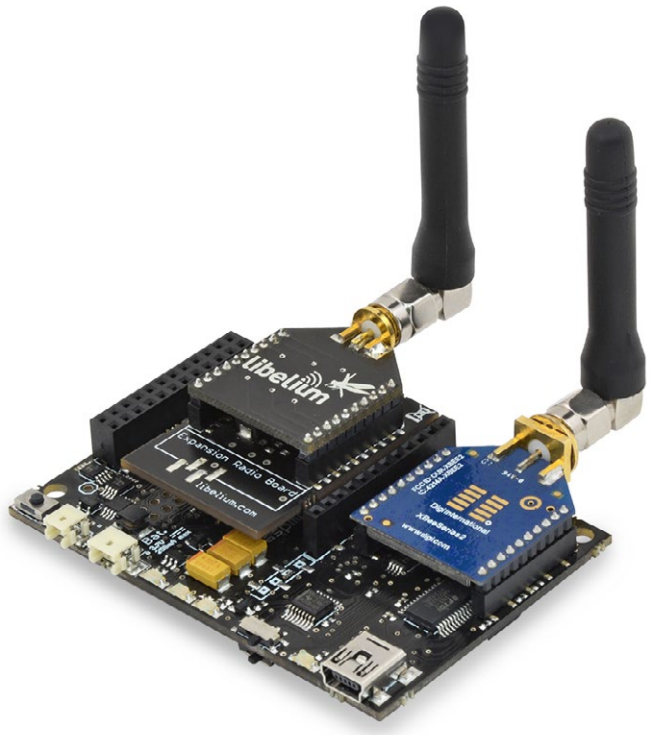


Figure : Waspmote with XBee radio on socket0 and WiFi module on socket 1

This API provides a function in order to initialize the RS-485 module module called `ON()`.

> ***Note:*** *The RS-485 module can be used only in the socket 0. If the user wants to use a wireless radio, he must use the socket 1*

The rest of functions are used the same way as they are used with older API versions. In order to understand them we recommend to read this guide.

WARNING:

- Avoid to use DIGITAL7 pin when working with Expansion Board. This pin is used for setting the XBee into sleep.
- Avoid to use DIGITAL6 pin when working with Expansion Board. This pin is used as power supply for the Expansion Board.

Incompatibility with Sensor Boards:

- Gases Board: Incompatible with SOCKET3B and NO_2 sensor.
- Agriculture Board: Incompatible with Sensirion and the atmospheric pressure sensor.
- Smart Metering Board: Incompatible with SOCKET2, SOCKET3, SOCKET4 and SOCKET5.
- Smart Cities Board: Incompatible with microphone and the CLK of the interruption shift register.
- Events Board: Incompatible with interruption shift register.

4. Applications

This module allows the user to interface the Waspmote ecosystem with RS-485 systems. Waspmote allows to perform three main applications:

1º- Connect any sensor to an existing RS-485 device/network

Waspmote can be configured to work as a node in the network, inserting sensor data into the RS-485 bus already present. Waspmote can obtain information from more than 70 sensors which are currently integrated in the platform by using specific sensor boards (e.g: CO, CO_2, temperature, humidity, acceleration, pH, IR, luminosity, etc). This way, the sensor information can be read from any RS-485 device connected to the bus.

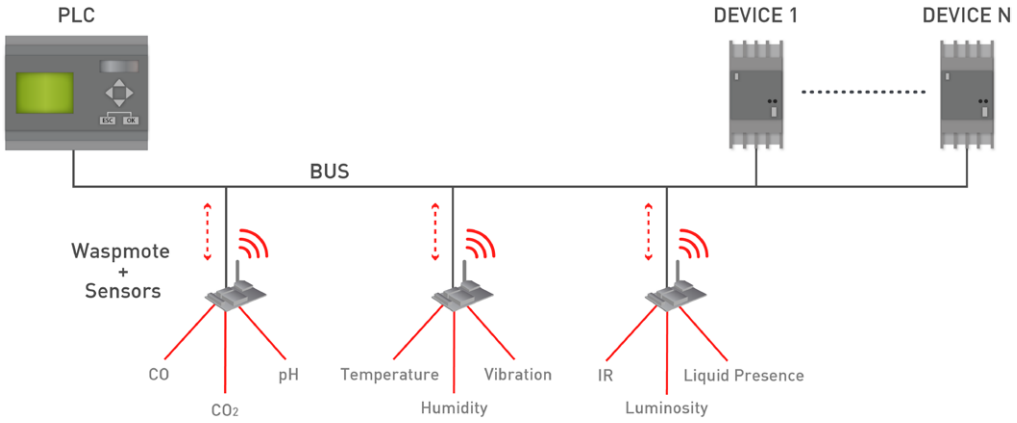


Figure : Waspmote integrated in an RS-485 network

2º- Add wireless connectivity to RS-485 devices

Waspmote can be configured to read the information coming from the RS-485 bus and send it wirelessly using any of the wireless modules available in the platform to a base station or even directly to a Cloud server. The available wireless technologies are: WiFi, 3G, GPRS, 802.15.4, ZigBee, LoRaWAN, LoRa, Sigfox, Bluetooth Pro, Bluetooth Low Energy, RF-868MHz, RF-900MHz.

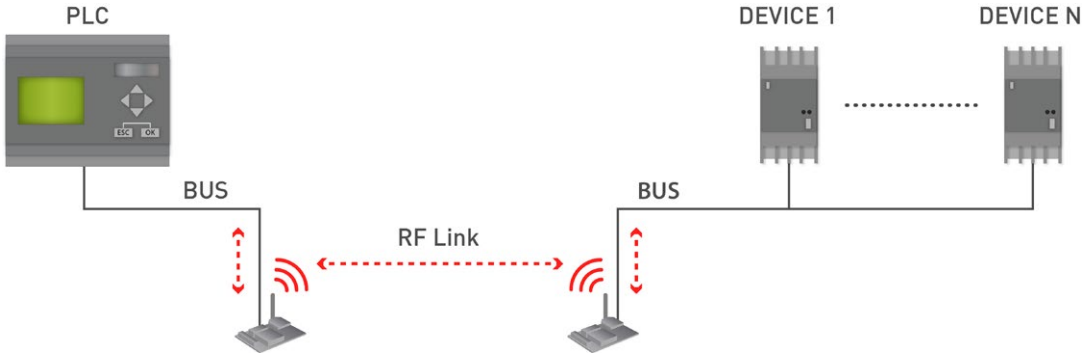


Figure : Waspmote replacing wired connections

3º- Connect to the Cloud RS-485 devices

Waspmote can be configured to read the information coming from the RS-485 bus and send it wirelessly directly to the Cloud using WiFi, 3G and GPRS radio interfaces.

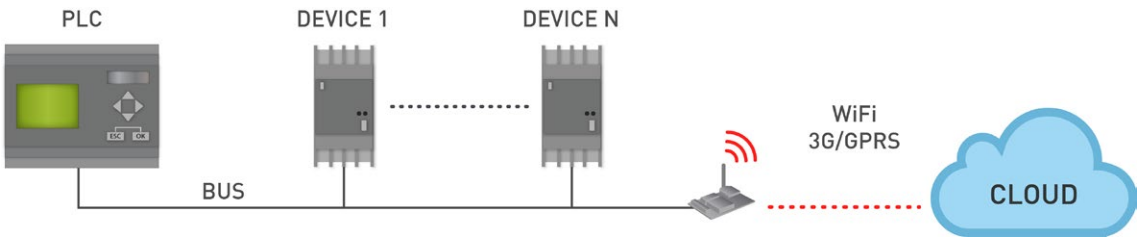


Figure : Cloud connection

5. Libelium's API

It is mandatory to include the RS-485 library when using this module. The following line must be introduced at the beginning of the code:

```
#include <Wasp485.h>
```

Waspmote's API RS-485 files:

- Wasp485.cpp
- Wasp485.h

APIs functions

- Private functions:

The following functions are executed inside the API functions. In normal conditions, the user must NOT manage or use them.

`void maxWrite(char address , char data);`	Writes data in the MAX3107 address.
`uint8_t maxRead(char address);`	Reads a data from the MAX3107. Returns the read data.
`void begin(void);`	Configures the MISO, MOSI, CS, SPCR.
`void setBitOrder(uint8_t bitOrder);`	Sets most significant bit first.
`void setClockDivider(uint8_t rate);`	Set the SPI clock divider relative to the system clock.
`void setDataMode(uint8_t mode);`	Sets the SPI data mode: that is, clock polarity and phase.
`uint8_t transfer(uint8_t _data);`	Transfers one byte over the SPI bus, both sending and receiving.
`void printIntegerInBase(unsigned long n , uint8_t base);`	Prints an integer in a given base.

Figure : Table of private functions

-Public functions:

`uint8_t ON(void)`	Powers the RS-485 module and opens the SPI
`OFF(void)`	Switches off the module and closes the SPI
`uint8_t baudRateConfig(unsigned long speed)`	It sets the speed of communication
`parityBit(bool state)`	Enable or disable the parity bit
`stopBitConfig(uint8_t numStopBits)`	Selects the number of stop bits to be inserted by the Transmitter
`uint8_t read(void)`	Receives data through the SPI.
`send(n)`	Sends data n through the SPI
`bool reset(void)`	All register bits are reset to their reset state and all FIFO buffers are cleared
`flush(void)`	Clear both the receive and transmit FIFOs of all data contents
`bool noiseReception(void)`	If noise is detected on the RX input during reception of a character
`uint8_t available(void)`	Returns true when the buffer is empty
`transmission(bool state)`	Disable/Enable transmission
`reception(bool state)`	Disable/Enable the receiver

Figure : Table of public functions

6. Library functions

6.1. Library constructor

To start using Waspmote RS-485 library, an object from class 'Wasp485' must be created. This object, called 'W485', is created inside Waspmote RS-485 library and it is public to all libraries. It is used through this guide to show how Waspmote RS-485 library works. When creating this constructor, all the variables are defined with an initial value by default.

6.2. Switching the module ON

This function powers the RS-485 module and configures the SPI bus. The default baud rate is 1200 bps and it can be modified by the function `baudRateConfig()`. The RS-485 module can be used only in the socket 0. This function is necessary to configure the module, so the RS-485 module must be plugged before, and returns zero, if the module has been configured correctly.

```
// Include always this library when you are using the Wasp485 functions
#include <Wasp485.h>

 void setup()
 {
       // Powers on the module and assigns the SPI in socket0
       if (W485.ON() == 0)
       {
             USB.println("RS-485 module started successfully");
       }
       else
       {
             USB.println("RS-485 did not initialize correctly");
       }
 }
```

You can see how to use this function in this example:

http://www.libelium.com/development/waspmote/examples/rs-485-01-send-data

6.3. Switching the module OFF

Switches off the RS-485 module and stops sending data frames. This function will disconnect the supply of the module so all data stored on the stack will be lost.

Example of use:

```
 {
       // Switches off the module and closes the SPI
       W485.OFF();
       delay(100);
 }
```

6.4. Configuring speed communication

The RS-485 module communication can be configured with the standard values of the RS-485 protocol. RS-485 module has been tested with 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200bps, 230400bps and 460800bps. All these speeds make RS-485 module very compatible with other devices. When a non-defined speed is introduced, the module is configured with the lowest communication speed (1200bps). The RS-485 module don't use a clock signal so the baud rate of all devices in a network must be configured before starting the communication.

```
// Include always this library when you are using the Wasp485 functions
#include <Wasp485.h>

void setup()
{
  // Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);
  // Powers on the module and assigns the SPI in socket0
  W485.ON();
  delay(100);
  // Configure the baud rate of the module
  W485.baudRateConfig(9600);
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs-485-01-send-data

6.5. Configuring the number of stop bits

The most common configuration used between computers is 8N1: eight bit characters, with one stop bit and no parity bit. With the RS-485 module you can configure the communication mode with one or two stop bits using the next function.

Example of use with one stop bit:

```
// Include always this library when you are using the Wasp485 functions
#include <Wasp485.h>

void setup()
{
  // Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);
  // Powers on the module and assigns the SPI in socket0
  W485.ON();
  delay(100);
  // Configure the baud rate of the module
  W485.baudRateConfig(9600);
  // Use one stop bit configuration
  W485.stopBitConfig(1);
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs-485-01-send-data

Example of use with two stop bits:

```
//Include always this library when you are using the Wasp485 functions
#include <Wasp485.h>

void setup()
{
  // Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);

  // Powers on the module and assigns the SPI in socket0
  W485.ON();
  delay(100);

  // Configure the baud rate of the module
  W485.baudRateConfig(9600);
  // Use one stop bit configuration
  W485.stopBitConfig(2);
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs485-01-send-data

6.6. Configuring the parity bit

Parity is used in many hardware applications to detect frame errors and is usually generated and checked by interface hardware. The RS-485 module uses an odd parity. Parity can be enabled or disabled depending on communication requirements.

Example of use with no parity:

```
//Include always this library when you are using the Wasp485 functions
#include <Wasp485.h>

void setup()
{
  // Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);

  // Powers on the module and assigns the SPI in socket0
  W485.ON();
  delay(100);
  // Configure the baud rate of the module
  W485.baudRateConfig(9600);
  // Configure the parity bit as disabled
  W485.parityBit(DISABLE);
  // Use one stop bit configuration
  W485.stopBitConfig(1);
  // Print hello message
  USB.println("Hello this is RS-485 communication send data example.");
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs-485-03-configuration-example/

Example of use with parity enabled:

```
//Include always this library when you are using the Wasp485 functions
#include <Wasp485.h>

void setup()
{
  // Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);
  // Powers on the module and assigns the SPI in socket0
  W485.ON();
  delay(100);
  // Configure the baud rate of the module
  W485.baudRateConfig(9600);
  // Configure the parity bit as disabled
  W485.parityBit(ENABLE);
  // Use one stop bit configuration
  W485.stopBitConfig(1);
  // Print hello message
  USB.println("Hello, this is RS-485 communication send data example");
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs-485-03-configuration-example/

6.7. Reset

After using this function all register bits are reset to their reset state and all FIFOs are cleared.

Example of use:

```
{
      // Resets the W485 module and clear the buffers
      W485.reset();
}
```

6.8. Sending data

Thanks to method overloading you can send any data through the SPI with the same function `send()`.

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs485-01-send-data

6.8.1. Sending a char

The most common way to send a character is to send it between single quotes (e.g. 'p').

Example of use:

```
{
  char data = 'p';
  // Send data through UART
  W485.send(data);
}
```

You can also send char variables by declaring the corresponding ASCII code. For example, if you initialize a char with “123”, it sends the corresponding ASCII “{“.

Example of use:

```
{
  char data = 123;
  // Send data through UART
  W485.send(data);
}
```

6.8.2. Sending an integer

You can send an unsigned or signed integer using the same function. If an unsigned int is declared you can assign a value from 0 to 65535 and if the variable is signed the value can be between -32768 and 32767.

Example of use:

```
{
  unsigned int data = 12345;
  // Send an unsigned int
  W485.send(data);
}


{
  int data = -12345;
  // Send a signed int
  W485.send(data);
}
```

6.8.3. Sending a string

The same function could be uses for sending string of characters.

Example of use:

```
{
  // Send a string trough the SPI
  W485.send(“Hello world”);
}
```

6.8.4. Sending with base

Send function allows to represent the variable in a specific base. You can select binary, octal, byte, decimal, and hexadecimal representation.

Example of use:

```
{
  int data = 12345;
  // Send 12345 in binary base. It prints 0110 000 0011 1001.
  W485.send(data, BIN);

  // Send 12345 in octal base. It prints 30071.
  W485.send(data, OCT);

  // Send 12345 in decimal base. It prints 12345.
  W485.send(data, DEC);

  // Send 12345 in hexadecimal base. It prints 3039.
  W485.send(data, HEX);
}
```

6.8.5. Sending a long

Long variables can be sent also with the function send. Long variables represent 32 bits or 4 bytes and the `send()` function allows signed and unsigned declaration.

Example of use:

```
{
  unsigned long data = 1234567;
  // Send and unsigned long
  W485.send(data);
}
```

6.9. Receiving data

The RS-485 module has a 128-byte buffer. This bytes can be read with the function `read()`. This function returns the read value from the buffer.

```
{
  // If data in response buffer
  if (W485.available())
  {
        while (W485.available())
        {
        // Read one byte from the buffer
        char data = W485.read();
        // Print data received in the serial monitor
        USB.print(data);
        }
  }
  delay(1);
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs485-02-receive-data/

6.10. Data available

This function returns the number of bytes available in the RS-485 buffer.

```
{
  // If data in response buffer
  if (W485.available())
  {
        while (W485.available())
        {
              // Read one byte from the buffer
              char data = W485.read();
              // Print data received in the serial monitor
              USB.print(data);
        }
  }
  delay(1);
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs485-02-receive-data/

6.11. Flush function

Flushes the buffer of incoming serial data. The `flush()` function waits for outgoing data to transmit before clearing the buffer content.

Example of use:

```
{
  // Flushes the buffer
  W485.flush();
  delay(10);
}
```

6.12. Tranmission control

Use this function to enable/disable transmission. If the transmission is disabled during transmission, the transmitter completes sending out the current character and then ceases transmission. Data still present in the transmit FIFO remains. The TX output is set to logic-high after transmission.

Example of use:

```
{
  // Disable the transmission
  W485.transmission(DISABLE);
}
```

You can see how to use this function in this example:
http://www.libelium.com/development/waspmote/examples/rs485-03-configuration-example/

The transmission could be enabled again by using the same function as we can see below.

Example of use:

```
{
  // Enable the transmission
  W485.transmission(ENABLE);
}
```

6.13. Reception control

Use this function to disable the receiver so that the receiver stops receiving data. All data present in the receive FIFO remains.

Example of use:

```
{
  // Disable the reception
  W485.reception(DISABLE);
}
```

The reception could be enabled again by using the same function as we can see below.

Example of use:

```
{
  // Disable the reception
  W485.reception(ENABLE);
}
```

7. Code examples and extended information

For more information about the Waspmote hardware platform go to:
http://www.libelium.com/waspmote
http://www.libelium.com/development/waspmote

In the Waspmote Development section you can find complete examples:
http://www.libelium.com/development/waspmote/examples

Example:

```
/*         ------ [RS-485_03] Configuration Example --------
 *
 *  This sketch shows the use of the RS-485 standard, and the use
 *  of the main functions of the library. This standard defines
 *  the electrical characteristics of drivers and receivers for
 *  use in digital systems. It does not specify or recommend any
 *  communications protocol. For a complete communication
 *  protocol, please see the Modbus examples.
 *
 *  Copyright (C) 2014 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
 *  Implementation:   Ahmad Saad
 */

//Include always this library when you are using the RS-485 functions
#include <Wasp485.h>

//Number of retires
uint8_t retries = 0;
uint8_t result = -1;

void setup()
{
// Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);

  // Powers on the module and assigns the SPI in socket0
  while ((result !=0) & (retries < 5))
  {
        retries ++;
        result = W485.ON();
        delay(1000);
  }
```

```
  if ( result == 0)
  {
        USB.println(“RS-485 module started successfully”);
  }
  else
  {
        USB.println(“RS-485 did not initialize correctly”);
  }

  delay(100);

  // Configure the baud rate of the module
  W485.baudRateConfig(9600);

  // Configure the parity bit as disabled
  W485.parityBit(DISABLE);

  // Use one stop bit configuration
  W485.stopBitConfig(1);

  W485.transmission(ENABLE);
  delay(250);
  W485.send(“Waspmote RS-485 module connected to the network”);

  delay(250);
  // Disables the transmission
  // Useful when sniffing the bus
  W485.transmission(DISABLE);
  delay(250);

  //Print hello message
  USB.println(“Hello, this is RS-485 communication configuration example”);
}


void loop()
{
  // Sniffing the bus. All data will be printed in the serial monitor.
  // If data in response buffer
  if (W485.available())
  {
        while (W485.available())
        {
              // Read one byte from the buffer
              char data = W485.read();
              // Print data received in the serial monitor
              USB.print(data ,BYTE);
        }

        USB.print(“\n”);
  }

  delay(1);
}
```

8. API changelog

Keep track of the software changes on this link:

www.libelium.com/development/waspmote/documentation/changelog/#RS-485

9. Documentation changelog

From v4.3 to v4.4

- References to the new LoRaWAN module

From v4.2 to v4.3

- References to the new Sigfox module

From v4.1 to v4.2

- References to the new LoRa module
- Created new chapter "API changelog"

From v4.0 to v4.1

- Expansion Radio Board section updated
- Reference to the standard male-female DB9 cable supplied