Modbus Library for Arduino

This library allows your Arduino to communicate via Modbus protocol. The Modbus is a master-slave protocol used in industrial automation and can be used in other areas, such as home automation.

The Modbus generally uses serial RS-232 or RS-485 as physical layer (then called Modbus Serial) and TCP/IP via Ethernet or WiFi (Modbus IP).

In the current version the library allows the Arduino operate as a slave, supporting Modbus Serial and Modbus over IP. For more information about Modbus see:

http://pt.wikipedia.org/wiki/Modbus http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

Author's note (motivation and thanks):

It all started when I found the Modbus RTU Arduino library of Juan Pablo Zometa. I had extend the library to support other Modbus functions.

After researching several other Modbus libraries I realized strengths and weaknesses in all of them.
I also thought it would be cool have a base library for Modbus and derive it for each type of physical layer used.

I appreciate the work of all the authors of the other libraries, of which I used several ideas to compose the modbus-arduino.
At the end of this document is a list of libraries and their authors.

Features

- Operates as a slave (master mode in development)
- Supports Modbus Serial (RS-232 or RS485) and Modbus IP (TCP)
- Reply exception messages for all supported functions
- Modbus functions supported:
    - 0x01 - Read Coils
    - 0x02 - Read Input Status (Read Discrete Inputs)
    - 0x03 - Read Holding Registers
    - 0x04 - Read Input Registers
    - 0x05 - Write Single Coil
    - 0x06 - Write Single Register
    - 0x0F - Write Multiple Coils
    - 0x10 - Write Multiple Registers

Notes:

1) When using Modbus IP the transport protocol is TCP (port 502) and, by default, the connection is terminated to each transmitted message, that is, is not a keep-alive type connection. If you need a TCP keep-alive connection you have to remove comments of this line in ModbusIP.h header (or ModbusIP_* headers):

```
#define TCP_KEEP_ALIVE
```

2) The offsets for registers are 0-based. So be careful when setting your supervisory system or your testing software. For example, in ScadaBR (http://www.scadabr.com.br)
offsets are 0-based, then, a register configured as 100 in the library is set to 100 in ScadaBR. On the other hand, in the CAS Modbus Scanner
(http://www.chipkin.com/products/software/modbus-software/cas-modbus-scanner/) offsets are 1-based, so a register configured as 100 in library should be 101 in this software.

3) Early in the library Modbus.h file there is an option to limit the operation
to the functions of Holding Registers, saving space in the program memory.
Just comment out the following line:

```
#define USE_HOLDING_REGISTERS_ONLY
```

Thus, only the following functions are supported:

- 0x03 - Read Holding Registers
- 0x06 - Write Single Register
- 0x10 - Write Multiple Registers

4) When using Modbus Serial is possible to choose between Hardware Serial(default) or Software Serial. In this case you must edit the ModbusSerial.h file and comment out the following line:

```
#define USE_SOFTWARE_SERIAL
```

Now, You can build your main program putting all necessary includes:

```
#include <Modbus.h>
#include <ModbusSerial.h>
#include <SoftwareSerial.h>
```

And in the setup() function:

```
SoftwareSerial myserial(2,3);
mb.config(&myserial, 38400);   // mb.config(mb.config(&myserial, 38400, 4) for RS-485
```

How to

There are five classes corresponding to five headers that may be used:

- Modbus - Base Library
- ModbusSerial - Modbus Serial Library (RS-232 and RS-485)
- ModbusIP - Modbus IP Library (standard Ethernet Shield)
- ModbusIP_ENC28J60 - Modbus IP Library (for ENC28J60 chip)
- ModbusIP_ESP8266AT - Modbus IP Library (for ESP8266 chip with AT firmware)

If you want to use Modbus in ESP8266 without the Arduino, I have news: http://www.github.com/andresarmento/modbus-esp8266

By opting for Modbus Serial or Modbus IP you must include in your sketch the corresponding header and the base library header, eg:

```
#include <Modbus.h>
#include <ModbusSerial.h>
```

Modbus Jargon

In this library was decided to use the terms used in Modbus to the methods names, then is important clarify the names of
register types:

Register type	Use as	Access	Library methods
Coil	Digital Output	Read/Write	addCoil(), Coil()
Holding Register	Analog Output	Read/Write	addHreg(), Hreg()
Input Status	Digital Input	Read Only	addIsts(), Ists()
Input Register	Analog Input	Read Only	addIreg(), Ireg()

Notes:

1. *Input Status* is sometimes called *Discrete Input*.
2. *Holding Register* or just *Register* is also used to store values in the slave.
3. Examples of use: A *Coil* can be used to drive a lamp or LED. A *Holding Register* to store a counter or drive a Servo Motor. A *Input Status* can be used with a reed switch in a door sensor and a *Input Register* with a temperature sensor.

Modbus Serial

There are four examples that can be accessed from the Arduino interface, once you have installed the library.
Let's look at the example Lamp.ino (only the parts concerning Modbus will be commented):

```
#include <Modbus.h>
#include <ModbusSerial.h>
```

Inclusion of the necessary libraries.

```
const int LAMP1_COIL = 100;
```

Sets the Modbus register to represent a lamp or LED. This value is the offset (0-based) to be placed in its supervisory or testing software.
Note that if your software uses offsets 1-based the set value there should be 101, for this example.

```
ModbusSerial mb;
```

Create the mb instance (ModbusSerial) to be used.

```
mb.config (&Serial, 38400, SERIAL_8N1);
mb.setSlaveId (10);
```

Sets the serial port and the slave Id. Note that the serial port is passed as reference, which permits the use of other serial ports in other Arduino models.
The bitrate and byte format (8N1) is being set. If you are using RS-485 the configuration of another pin to control transmission/reception is required.
This is done as follows:

```
mb.config (& Serial, 38400, SERIAL_8N1, 2);
```

In this case, the pin 2 will be used to control TX/RX.

```
mb.addCoil (LAMP1_COIL);
```

Adds the register type Coil (digital output) that will be responsible for activating the LED or lamp and verify their status.
The library allows you to set an initial value for the register:

```
mb.addCoil (LAMP1_COIL, true);
```

In this case the register is added and set to true. If you use the first form the default value is false.

```
mb.task ();
```

This method makes all magic, answering requests and changing the registers if necessary, it should be called only once, early in the loop.

```
digitalWrite (ledPin, mb.Coil (LAMP1_COIL));
```

Finally the value of LAMP1_COIL register is used to drive the lamp or LED.

In much the same way, the other examples show the use of other methods available in the library:

```
void addCoil (offset word, bool value)
void addHreg (offset word, word value)
void addIsts (offset word, bool value)
void addIreg (offset word, word value)
```

Adds registers and configures initial value if specified.

```
bool Coil (offset word, bool value)
bool Hreg (offset word, word value)
bool Ists (offset word, bool value)
bool IREG (offset word, word value)
```

Sets a value to the register.

```
bool Coil (offset word)
Hreg word (word offset)
bool Ists (offset word)
IREG word (word offset)
```

Returns the value of a register.

Modbus IP

There are four examples that can be accessed from the Arduino interface, once you have installed the library.
Let's look at the example Switch.ino (only the parts concerning Modbus will be commented):

```
#include <SPI.h>
#include <Ethernet.h>
#include <Modbus.h>
#include <ModbusIP.h>
```

Inclusion of the necessary libraries.

```
const int SWITCH_ISTS = 100;
```

Sets the Modbus register to represent the switch. This value is the offset (0-based) to be placed in its supervisory or testing software.
Note that if your software uses offsets 1-based the set value there should be 101, for this example.

```
ModbusIP mb;
```

Create the mb instance (ModbusIP) to be used.

```
mac byte [] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
ip byte [] = {192, 168, 1, 120};
mb.config (mac, ip);
```

Sets the Ethernet shield. The values of the MAC Address and the IP are passed by the config() method.
The syntax is equal to Arduino Ethernet class, and supports the following formats:

```
void config (uint8_t * mac)
void config (uint8_t * mac, IPAddress ip)
void config (uint8_t * mac, IPAddress ip, IPAddress dns)
void config (uint8_t * mac, IPAddress ip, IPAddress dns, gateway IPAddress)
void config (uint8_t * mac, IPAddress ip, IPAddress dns, IPAddress gateway, subnet IPAddress)
```

Then we have:

```
mb.addIsts (SWITCH_ISTS);
```

Adds the register type Input Status (digital input) that is responsible for detecting if a switch is in state on or off.
The library allows you to set an initial value for the register:

```
mb.addIsts (SWITCH_ISTS, true);
```

In this case the register is added and set to true. If you use the first form the default value is false.

```
mb.task ();
```

This method makes all magic, answering requests and changing the registers if necessary, it should be called only once, early in the loop.

```
mb.Ists (SWITCH_ISTS, digitalRead (switchPin));
```

Finally the value of SWITCH_ISTS register changes as the state of the selected digital input.

Modbus IP(ENC28J60)

The Arduino standard Ethernet shield is based on chip WIZnet W5100, therefore the IDE comes with this library installed. If you have a shield based on ENC28J60 from Microchip you must install another Ethernet library. Among several available we chose EtherCard.

Download the EtherCard in https://github.com/jcw/ethercard and install it in your IDE.
Use the following includes in your sketches:

```
#include <EtherCard.h>
#include <ModbusIP_ENC28J60.h>
#include <Modbus.h>
```

Done! The use of Modbus functions is identical to the ModbusIP library described above.

Notes:

1. EtherCard is configured to use the pins 10, 11, 12 and 13.

2. The voltage for shields based on ENC28J60 is generally 3.3V.
3. Another alternative is to use the ENC28J60 UIPEthernet library, available from https://github.com/ntruchsess/arduino_uip. This library was made so that mimics the same standard Ethernet library functions, whose work is done by Wiznet W5100 chip. As the ENC28J60 not have all the features of the other chip, the library UIPEthernet uses a lot of memory, as it has to do in software what in the shield Wiznet is made in hardware. If for some reason you need to use this library, just change the file ModbusIP.h and your sketches, changing the lines:

```
#include <Ethernet.h>
```

by

```
#include <UIPEthernet.h>
```

Then, you can use the ModbusIP library (not the ModbusIP_ENC28J60).
In fact it allows any library or skecth, made for
Wiznet shield be used in shield ENC28J60. The big problem with this approach
(and why we chose EtherCard) is that UIPEthernet library + ModbusIP uses about 60%
arduino program memory, whereas with Ethercard + ModbusIP_ENC28J60
this value drops to 30%!

Nano Ethernet Shield ENC28J60

Dit ethernet shield kan je gemakkelijk verbinden met een Arduino Nano.
De namen van de pinnen staan vermeldt op de onderkant printplaat.

Pinout van module:

- SS(CS) pin: D10
- MOSI(SI) pin: D11
- MISO(SO) pin: D12
- SCK pin: D13

Modbus IP (ESP8266 AT)

Modules based on ESP8266 are quite successful and cheap. With firmware that
responds to AT commands (standard on many modules) you can use them as a
simple wireless network interface to the Arduino.

The firmware used in the module (at_v0.20_on_SDKv0.9.3) is available at:
http://www.electrodragon.com/w/ESP8266_AT-command_firmware

(Other AT firmwares compatible with ITEAD WeeESP8266 Library should work)

Warning: Firmware such as NodeMCU and MicroPython does not work because libraries
used here depend on a firmware that responds to AT commands via serial interface.
The firmware mentioned are used when you want to use ESP8266 modules without the Arduino.

You will need the WeeESP8266 library (ITEAD) for the Arduino. Download from:

https://github.com/itead/ITEADLIB_Arduino_WeeESP8266 and install in your IDE.

Notes:

1. The ESP8266 library can be used with a serial interface by hardware (HardwareSerial) or by software (SoftwareSerial). By default it will use HardwareSerial, to change edit the file ESP8266.h removing the comments from line:

```
#define ESP8266_USE_SOFTWARE_SERIAL
```

2. Remember that the power of ESP8266 module is 3.3V.

For Modbus IP (ESP8266 AT) there is four examples that can be accessed from the Arduino interface.
Let's look at the example Lamp.ino (only the parts concerning Modbus will be commented):

```
#include <ESP8266.h>
#include <SoftwareSerial.h>   //Apenas se utilizar Softwareserial para se comunicar com o módulo
#include <Modbus.h>
#include <ModbusIP_ESP8266AT.h>
```

Inclusion of the necessary libraries.

```
SoftwareSerial wifiSerial(2 , 3);
```

Creates the serial interface via software using pins 2 (RX) and 3 (TX). So it can use
hardware for the serial communication with the PC (e.g. for debugging purposes) in Arduino models that have only one serial (Ex .: Arduino UNO).

```
ESP8266 wifi(wifiSerial, 9600);
```

Create the wifi object (ESP8266) specifying the rate in bps.
Warning: If you use SoftwareSerial do not specify a baud rate of 115200bps or more for the serial because it will not function. Some firmware / modules comes with 115200bps by default. You will have to change the module via AT command:

```
AT+CIOBAUD=9600
```

Continuing with our example:

```
const int LAMP1_COIL = 100;
```

Sets the Modbus register to represent a lamp or LED. This value is the offset (0-based) to be placed in its supervisory or testing software.
Note that if your software uses offsets 1-based the set value there should be 101, for this example.

```
ModbusIP mb;
```

Create the mb instance (ModbusSerial) to be used.

```
mb.config(wifi, "your_ssid", "your_password");
```

Configure ESP8266 module. The values quoted correspond to the network name (SSID) and security key.
By default IP configuration is received via DHCP. See the end of the section how to have an Static IP
(important so you do not need to change the master / supervisor if the IP changes).

Folowing, we have:

```
mb.addCoil (LAMP1_COIL);
```

Adds the register type Coil (digital output) that will be responsible for activating the LED or lamp and verify their status.
The library allows you to set an initial value for the register:

```
mb.addCoil (LAMP1_COIL, true);
```

In this case the register is added and set to true. If you use the first form the default value is false.

```
mb.task();
```

This method makes all magic, answering requests and changing the registers if necessary, it should be called only once, early in the loop.

```
digitalWrite(ledPin, mb.Coil(LAMP1_COIL));
```

Finally the value of LAMP1_COIL register is used to drive the lamp or LED.

Quite similarly to other examples show the use of other methods available in the library.

Using a static IP on the ESP8266 module

We are aware today of two options:

1) In your router configure the MAC address of the module so that the IP address provided by DHCP is always the same (Most routers have this feature).

2) In your code, include two lines to change the IP address after the module configuration:

```
mb.config(wifi, "your_ssid", "your_password");
delay(1000);
wifiSerial.println("AT+CIPSTA=\"192.168.1.44\"");
```

Note .: For the module to receive IP via DHCP again you will need to remove the lines and run (at least once) the command: AT + CWDHCP = 1.1 via direct connection to the module, either:

```
wifiSerial.println("AT+CWDHCP=1,1");
```

Other Modbus libraries

Arduino Modbus RTU

Author: Juan Pablo Zometa, Samuel and Marco Andras Tucsni

Year: 2010

Website: https://sites.google.com/site/jpmzometa/

Simple Modbus

Author: Bester.J

Year: 2013 Website: https://code.google.com/p/simple-modbus/

Arduino-Modbus slave

Jason Vreeland [CodeRage]

Year: 2010

Website: http://code.google.com/p/arduino-modbus-slave/

Mudbus (Modbus TCP)

Author: Dee Wykoff

Year: 2011

Website: http://code.google.com/p/mudbus/

ModbusMaster Library for Arduino

Author: Doc Walker

Year: 2012

Website: https://github.com/4-20ma/ModbusMaster

Website: http://playground.arduino.cc/Code/ModbusMaster

Contributions

http://github.com/andresarmento/modbus-arduino

prof (at) andresarmento (dot) com

License