Modbus communication between Raspberry PI2 and Arduino

Steve Despres

Mail: stevedp@live.fr

26/06/2017

IstiA
École d'Ingénieurs de l'Université d'Angers

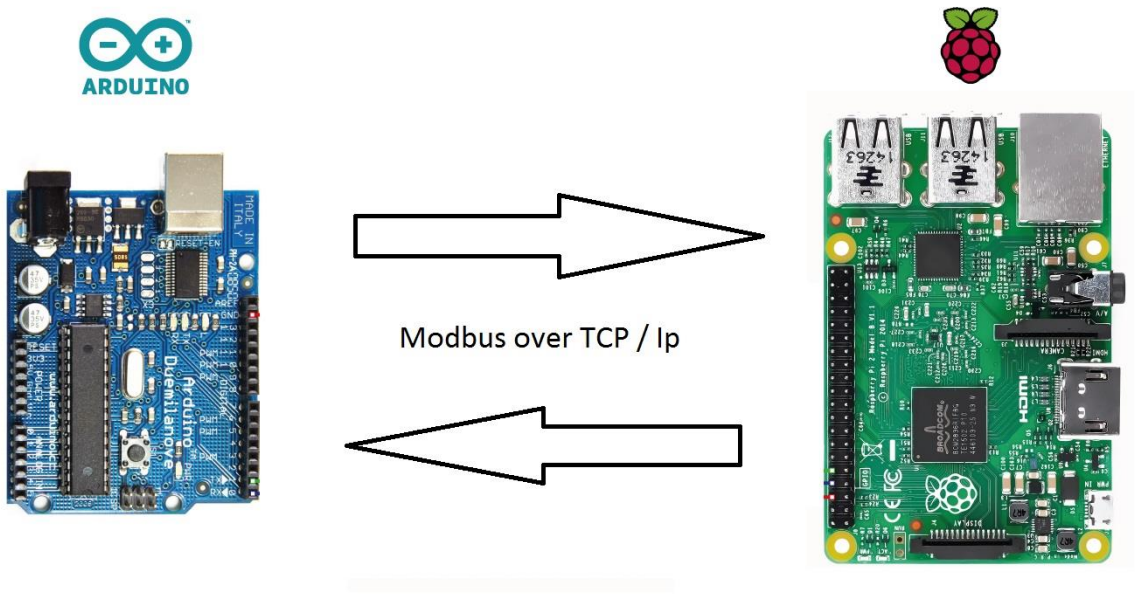
Ingeniaritza Goi Eskola Teknikoa
Escuela Técnica Superior de Ingeniería
Bilbao

Summary

Introduction

I'm a French student, from the engineering school ISTIA, in university of Angers and I'm in internship at the engineering school of Bilbao in Spain to work in this project.

The objective of this project is to realize a basic communication between 2 systems, a Raspberry Pi2 and an Arduino Uno in the first part (with Ethernet), then a WeMos D1 in the second part (with Wi-Fi). This communication must follow the Modbus protocol using TCP mode; it's called Modbus over TCP/IP (Ethernet).



It's working like a Server and a Client, with specific method of data encapsulation and can use 4 different object types to stock and share data. To know more about the Modbus protocol, you can look the official documentation.

Object type	Access	Size
Coil	Read-write	1 bit
Discrete input	Read-only	1 bit
Input register	Read-only	16 bits
Holding register	Read-write	16 bits

In this following document, I describe the different steps to realize this Modbus communication using a switch and Ethernet cables in first time then using a Wi-Fi. I describe how I configure the softwares and I program the 2 systems.

Materials

Equipment

In this part, I specify my work equipment and my tools that I use to do this project.

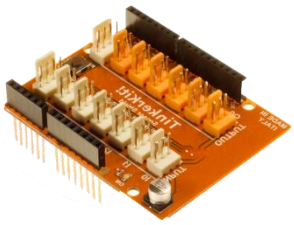
** : only for Ethernet communication*

+: only for Wi-Fi communication

- Computer : Windows 10
- Raspberry Pi 2 : Raspbian Jessie
- *Arduino Uno : Usb cable (only for Ethernet communication)
- *Ethernet Shield for Arduino
- Tinker Kit : with a LED and potentiometer
- +Dongle Wi-Fi for Raspberry
- *Switch Cisco
- WeMos D1
- Many Ethernet cables
- Many alimentation cable
- HDMI cable : to display the RPi2 screen






Software

Here, I specify the different softwares that I use.

** : only for Ethernet communication*

+: only for Wi-Fi communication

- Virtual Machine : not necessary
- Arduino IDE : to develop Arduino and WeMos
- Eclipse : to develop the RPi2
- Tera Term : to connect yourself to the RPi2 (ssh)
- Libmodbus library : library (C/C++) in Eclipse to RPi2
- *Modbus Arduino library : to Arduino IDE (Github of Andre Saramento)
- +Modbus EPS8266 library : to WeMos in Arduino IDE (Github of Andre Saramento) :
    - https://github.com/andresarmento/modbus-arduino

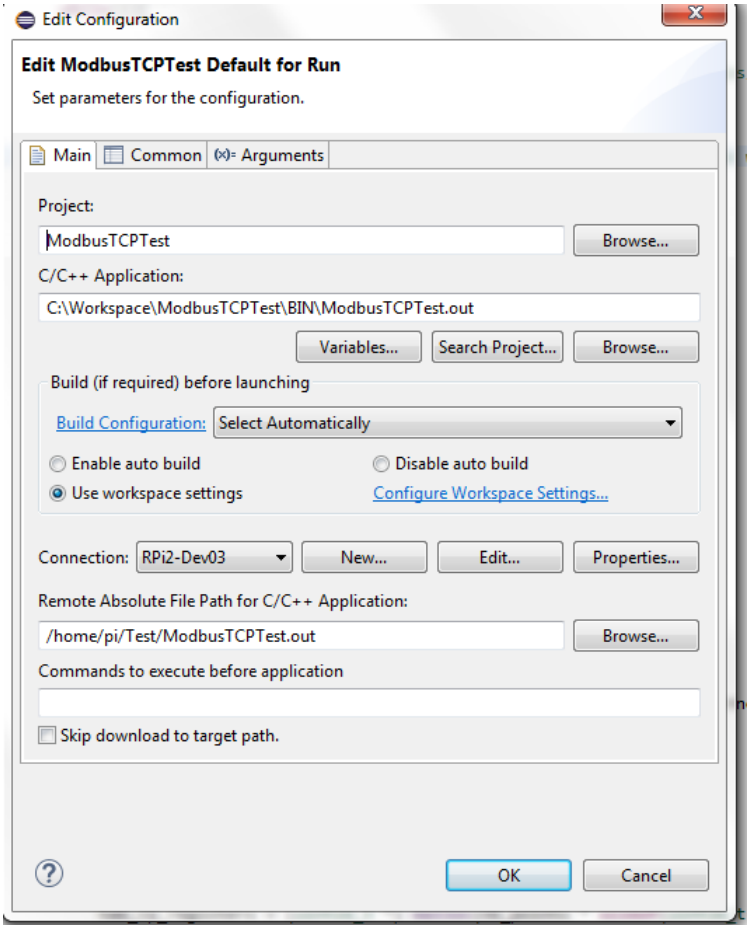
Configuration

Firstly, I need to put a static Ethernet interface on the Raspberry, in the file /etc/network/interfaces in Raspbian : I use 192.168.0.147

```
auto eth0

iface eth0 inet static

    address 192.168.0.147

    netmask 255.255.255.0

    gateway 192.168.0.1
```

After that, I connect my RPi2 to my computer with Ethernet cable, and configure my local Ethernet ip to be in the same network of the RPI2: I use 192.168.0.1 as IP and 255.255.255.0 as mask (not necessary to write a DNS and Gateway). Now I can ping the RPi2.

In Eclipse, to use a cross compiler, I use a Makefile and this configuration:

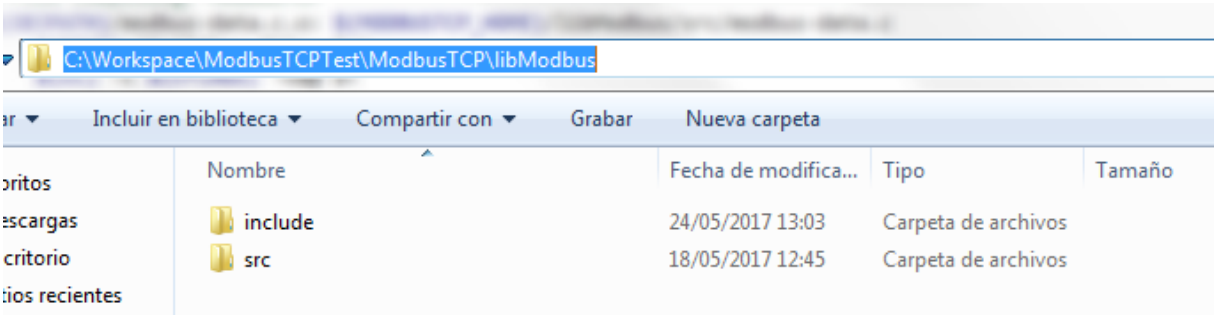


The Makefile is essential to compile directly on the RPi2. It describes the different configurations to do actions, to load the libraries and to compile. I use a Makefile already written; just I have added the path of the libmodbus library with these lines.

```
#=============================================================================
# Building options
#=============================================================================

# Include paths
CINC_PATH += -I $(MAIN_HOME)/include -I $(MODBUSTCP_HOME)/libModbus/include

# Library paths
CLIB_PATH += -L$(LIBPATH)

# Compiling and Linking options
CFLAGS   := $(CINC_PATH) $(CFLAGS)
CPPFLAGS := $(CINC_PATH) $(CPPFLAGS)
LDFLAGS  := $(CLIB_PATH) $(LDFLAGS)
```

Thanks in the Make file, my Eclipse workspace contains a BIN file to place .out file, an OBJ file to place the objects of the project and a LIB file to place the libraries compiled.

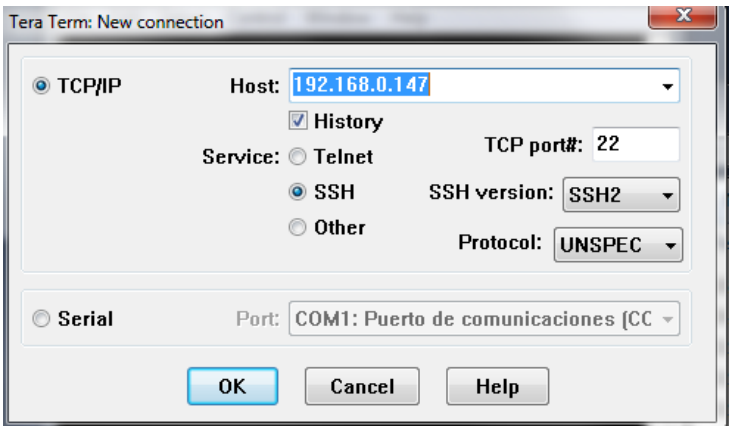
To add a libmodbus library, downloaded on the official website, I have added a directory named ModbusTCP in my Workspace. Inside there is another directory “libmodbus” which contains 2 directories: “include” and “src”. The “include” directory must have the header file (.h) of the libmodbus library, and the “src” directory must have the source file (.c).



Normally, the library is added in Eclipse and the cross-compiler works.

To configure the Arduino IDE, it’s easier. You just have to install it, specify the port, the communication speed (9600) and the target (Arduino Uno). Then to add the library, you have to select the archive .zip of the Modbus Arduino Librairy (github Andre Saramento) in the Arduino IDE options.

To use Tera Term, the Raspberry must be connected by Ethernet cable with the computer. It's the configuration:



After that, you can access to the Raspbian terminal directly your PC instead of with HDMI cable.

Communication with Ethernet

To start a Modbus communication, you must have a server and a client. Here, the server is the Arduino because it will receive data from potentiometer, will stock them in a Modbus register and then, the client (RPi2) will recover these data to stock them. After, the client will calculate the average of the 10 last values, it will write it on another Modbus register and the server will recover this average to light the LED.

Arduino

Once the Arduino IDE and the Modbus library installed, it's ready to develop the Modbus server. The Arduino, the Ethernet shield and a TinkerKit must be connected together and connected to the computer with Ethernet cable.

HREG_1 and HREG_2 are the Modbus registers addresses.

In the Setup part, I define the input/output connected (LED and potentiometer of the TinkerKit), I define the Modbus registers and I configure the IP and mac addresses. It must write the mac address of the Arduino shield, and then specify an IP address to be in the same network of the RPi2, I chose 192.168.0.146. The configuration is done by the Modbus library.

In the Loop part, mb.task() manages the Modbus protocol (request, query, connection, …). The value in the HREG_2 (the average calculated by the client) is reading and send to light the LED, and the value of the potentiometer is writing in HREG_1 each 500ms.

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


// Modbus Registers Offsets (0-9999)
const int HREG_1 = 200;
const int HREG_2 = 400;

//Used Pins
const int sensorPin = A0;

//ModbusIP object
ModbusIP mb;

long ts;


void setup() {

    Serial.begin(9600);      // open the serial port at 9600 bps:

    // The media access control (ethernet hardware) address for the shield
```

```
    byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0x5C, 0x08 };
    // The IP address for the shield
    byte ip[] = { 192, 168, 0, 146 };
    //Config Modbus IP
    mb.config(mac, ip);

    // Add 2 Holding register
    //register 1
    mb.addHreg(HREG_1);
    //register 2
    mb.addHreg(HREG_2);

    digitalWrite(A0, LOW);
    digitalWrite(A0, INPUT_PULLUP);  // set pullup on analog pin 0
    //Output 9 = LED
    pinMode(9, OUTPUT);


    ts = millis();
}

void loop() {


   //Call once inside loop() - all magic here
   mb.task();

  //Read the average of the last values of potentiometer [0 - 1024]
provided by the Raspberry in the register 2
  //And use this value for the LED
   analogWrite(9, (mb.Hreg(HREG_2)/4) );

   //Read each 500ms
   if (millis() > ts +500 ) {
       ts = millis();

       //Add the values of potentiometer in the register 1
       mb.Hreg(HREG_1, analogRead(sensorPin));

       Serial.print("\val = ");
       Serial.print(analogRead(sensorPin));
       Serial.print("\naverage = ");
       Serial.print(mb.Hreg(HREG_2));
       Serial.print("\n");

   }
}
```

Raspberry Pi 2

Now, I have to develop the RPi2 to be a Modbus client. To do that, the libmodbus library is essential; it provides many functions to manage the Modbus protocol. I used the test code of libmodbus to write that.

Firstly, the RPi2 (192.168.0.147) connects itself to the Arduino (192.168.0.146) with TCP mode, and the variables are initialized

In the While loop, it's here where the RPi2 read the register 1 (HREG_1) to recover the value of potentiometer, stock the 10 last values in a tab and then calculate the average. After that, the average is writing in the register 2 (HREG_2). This is done each 750ms.

With these codes, the Modbus communication works.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <modbus.h>
#include "unit-test.h"

enum {
    TCP,
    TCP_PI,
    RTU
};

/*============================================================================*/
/* Macros y definiciones
*/
/*============================================================================*/

#define RPI2TEST01_NO_ERROR        0
#define RPI2TEST01_ERROR           -1

#define LOOP             1
#define ADDRESS_START    0
#define ADDRESS_END     99

/*============================================================================*/
/* Funciones de usuario
*/
/*============================================================================*/

//Function to test the compilation on the RPi2
int CheckRPi2Test01(void)
{
 int out = RPI2TEST01_NO_ERROR;
 return out;

}
```

```
//Function to calculate the average of the n values in tab
uint16_t CalculAverage( uint16_t* tab, int n )
{
    uint16_t sum =0;
    int i;
    for(i=0;i<n;i++){ sum += tab[i];}

    return sum/n;
}

/*=======================================================================*/
/* Programa Principal
*/
/*=======================================================================
===*/
/*=======================================================================*/
int main(int argc, char *argv[])
{
    int out = RPI2TEST01_ERROR;

        // Many variables usefull
        uint16_t *tab_rp_registers;
        uint16_t *TAB_RP_REG;
        modbus_t *ctx;

        uint8_t value;
        int nb_points;
        int rc, rg, n,i;
        float real;
        uint32_t ireal;

        struct timeval response_timeout;
        int use_backend;
        int nb_loop;
        int addr;

        bool tabFull = false;
        uint16_t average=0;
        n=0;

        //Config to TCP
        if (argc > 1) {

            if (strcmp(argv[1], "tcp") == 0) {
                use_backend = TCP;
            } else {
                fprintf(stderr, "NO TCP\n");
            }

        } else {
            /* By default */
            use_backend = TCP;
        }

        //Config backend
        if (use_backend == TCP) {
            ctx = modbus_new_tcp("192.168.0.146", 502);
        } else {
            fprintf(stderr, "NO TCP\n");
        }
```

```
        if (ctx == NULL) {
            fprintf(stderr, "Unable to allocate libmodbus context\n");
            return -1;
        }

        modbus_set_debug(ctx, TRUE);
        modbus_set_error_recovery(ctx,MODBUS_ERROR_RECOVERY_LINK );
        modbus_set_error_recovery(ctx,MODBUS_ERROR_RECOVERY_PROTOCOL);

        //Allocate and initialize the memory to store the values of
registers
        TAB_RP_REG = (uint16_t *) malloc(10 * sizeof(uint16_t));
        memset(TAB_RP_REG, 0, 10 * sizeof(uint16_t));

        printf("**TESTING : Holding REGISTERS**\n");

        /** Holding REGISTERS **/
        while(1){

            //Connection => if error
           if (modbus_connect(ctx) == -1) {
                fprintf(stderr, "Connection failed:
%s\n",modbus_strerror(errno));
                modbus_free(ctx);
                return -1;
            }

            /* Allocate and initialize the memory to store the registers */
            nb_points = (UT_REGISTERS_NB > UT_INPUT_REGISTERS_NB) ?
                UT_REGISTERS_NB : UT_INPUT_REGISTERS_NB;

            //Tab register
            tab_rp_registers = (uint16_t *) malloc(nb_points *
sizeof(uint16_t));
            memset(tab_rp_registers, 0, nb_points * sizeof(uint16_t));

            //Read input registers (HREG_1 addr=200) : value of
potentiometer
            rc = modbus_read_registers(ctx, 200,UT_INPUT_REGISTERS_NB,
                                       tab_rp_registers);

            printf("Modbus_read_input_registers: ");

            if (rc != UT_INPUT_REGISTERS_NB) {
                printf("FAILED (nb points %d)\n", rc);
                goto close;
            }


            //Values traitment
            for (i=0; i < UT_INPUT_REGISTERS_NB; i++) {

                //If tab of values is full
                if(n>9){
                    tabFull =true;
                    n=0;
                }

                //Keep the 10 last values of registers in a tab
                TAB_RP_REG[n]= tab_rp_registers[i];
                n++;
```

```
                //if tab is full => Calcul average of the 10 last values
                if( tabFull ){
                    average = CalculAverage(TAB_RP_REG, 10);
                //if tab is not full => Calcul average with the values
already obtained
                } else{
                    average = CalculAverage(TAB_RP_REG, (n));
                }
                //Print
                //The value of potntiometer
                printf("value = %d\n",tab_rp_registers[i]);
                //The average calculated
                printf("average = %d\n",average);

            }
            printf("OK\n");

            /* Close the connection */
            modbus_close(ctx);

            //New connection
             ctx = modbus_new_tcp("192.168.0.146", 502);

           if (modbus_connect(ctx) == -1) {
                fprintf(stderr, "Connection failed:
%s\n",modbus_strerror(errno));
                modbus_free(ctx);
                return -1;
            }
               //Write the average in the register REG_2 (addr=400)
            rc = modbus_write_register(ctx, 400,average);

            sleep(0.75);

            /* Close the connection */
            modbus_close(ctx);
        }

    close:
        /* Free the memory */

        free(tab_rp_registers);
        free(TAB_RP_REG);

        /* Close the connection */
        modbus_close(ctx);
        modbus_free(ctx);

 return out;

}
```

Communication with Wi-Fi

To do the same modbus communication but using Wi-Fi, I have to develop a Wi-Fi access point on the Raspberry. Then, in this case, I use a WeMos D1, it's like an Arduino with a Wi-Fi dongle, to connect it with Wi-Fi and to realize the Modbus communication.

Raspberry Pi 2

To do a Wi-Fi access point on the Raspberry, I use a RPi2, a Wi-Fi dongle and a screen (or ssh connection) to have an access to the terminal. I followed this tutorial to configure the Raspberry : https://pimylifeup.com/raspberry-pi-wireless-access-point/

The main steps are :

- Update and upgrade the RPi2
- Install these packages : hostapd ad dnsmask
- Create a Wi-Fi interface in the file /etc/network/interfaces
- Specify a static IP, I choose 192.168.1.147
- Configure the Wi-Fi interface : SSID, password, mode, etc… in /etc/hostapd/hostapd.conf
- Configure the dnsmask service in /etc/dnsmask.conf
- Launch the Wi-Fi

After that, the Wi-Fi is enabled and you can connect any Wi-Fi system with the SSID and the password.

About the Modbus communication, it's the same code as with Ethernet cables, just the ip changes. I have added some lines to create a .CSV file, with the date & hour in the name, to write all the values of the potentiometer and the average inside. Moreover, I have added this line in /etc/rc.local to launch the Wi-Fi when the Raspberry is powered on :

Sudo hostapd –dd /etc/hostapd/hostapd.conf

```
/*========================================================================*/
/* Ficheros de cabeceras
*/
/*========================================================================*/

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <modbus.h>
#include <fstream>
#include <iostream>
```

```
#include <time.h>

using namespace std;

#include "unit-test.h"

enum {
    TCP,
    TCP_PI,
    RTU
};


/*============================================================================*/
/* Macros y definiciones
*/
/*============================================================================*/

#define RPI2TEST01_NO_ERROR        0
#define RPI2TEST01_ERROR           -1

#define LOOP             1
#define ADDRESS_START    0
#define ADDRESS_END     99


/*============================================================================*/
/* Funciones de usuario
*/
/*============================================================================*/

int CheckRPi2Test01(void)
{
 int out = RPI2TEST01_NO_ERROR;
 return out;

}

uint16_t CalculAverage( uint16_t* tab, int n )
{
    uint16_t sum =0;
    int i;
    for(i=0;i<n;i++){ sum += tab[i];}

    return sum/n;
}

/*============================================================================*/
/* Programa Principal
*/
/*============================================================================*/


int main(int argc, char *argv[])
{

    int out = RPI2TEST01_ERROR;

        uint16_t *tab_rp_registers;
        uint16_t *TAB_RP_REG;
        modbus_t *ctx;
```

```
        int i;
        uint8_t value;
        int nb_points;
        int rc, rg, n;
        float real;
        uint32_t ireal;

        struct timeval response_timeout;
        int use_backend;
        int nb_loop;
        int addr;

        bool tabFull = false;
        uint16_t average=0;
        n=0;

        //Usefull to create file name with the date
        time_t t = time(NULL);
        struct tm tm = *localtime(&t);
        char *fileName = (char*) malloc(sizeof(char ) * 16 );
        char *date = (char*) malloc(sizeof(char ) * 32 );
        char *path = (char*) malloc(sizeof(char ) * 64 );

        //Config to TCP
        if (argc > 1) {

            if (strcmp(argv[1], "tcp") == 0) {
                use_backend = TCP;
            } else {
                fprintf(stderr, "NO TCP\n");
            }

        } else {
            /* By default */
            use_backend = TCP;
        }


        //Config backend
        if (use_backend == TCP) {
            ctx = modbus_new_tcp("192.168.1.119", 502);
        } else {
            fprintf(stderr, "NO TCP\n");

        }


        if (ctx == NULL) {
            fprintf(stderr, "Unable to allocate libmodbus context\n");
            return -1;
        }

        modbus_set_debug(ctx, TRUE);
        modbus_set_error_recovery(ctx,MODBUS_ERROR_RECOVERY_LINK );
        modbus_set_error_recovery(ctx,MODBUS_ERROR_RECOVERY_PROTOCOL);

        //Allocate and initialize the memory to store the values of
registers
        TAB_RP_REG = (uint16_t *) malloc(10 * sizeof(uint16_t));
        memset(TAB_RP_REG, 0, 10 * sizeof(uint16_t));
```

```
        printf("** UNIT TESTING : Holding REGISTERS**\n");
        //Creation of file name with the date and hour
        //Specify the directory
        strcpy(fileName, "dataModbus_");
        sprintf(date,"%02d_%02d_%04d_%02dh%02d.csv", tm.tm_mday,
tm.tm_mon+1, tm.tm_year+1900, tm.tm_hour, tm.tm_min );
        strcat(fileName, date);
        strcpy(path, "ModbusDataFiles/");
        //Path to create the file in directory
        strcat(path,fileName);


        bool isEmpty = true;

        /** Holding REGISTERS **/
        while(1){

            //Creating or opening file "fileName" in directory
ModbusDataFiles/
            FILE *file = fopen(path, "ab+");

            if(file == NULL) //if directory does not exist =>create file
without directory
            {
                //The path = just fileName
                strcpy(path, fileName);

                //Try to create file without directory
                file = fopen(fileName, "ab+");

                if(file == NULL)
                    {
                    fprintf(stderr,"\n ********* %d ******** \n\n" );
                        printf("Error opening file\n");
                        exit(1);
                    }
            }

            //If the file is empty, write title of the 2 variables
            if(isEmpty){
                fprintf(stderr,"\n Create file in /%s \n\n" ,path );
                fprintf(file,"Value,AverageOf10LastValues\n");
                isEmpty = false;
            }

            //Connection
           if (modbus_connect(ctx) == -1) {
                fprintf(stderr, "Connection failed:
%s\n",modbus_strerror(errno));
                modbus_free(ctx);
                return -1;
            }

            /* Allocate and initialize the memory to store the registers */
            nb_points = (UT_REGISTERS_NB > UT_INPUT_REGISTERS_NB) ?
                UT_REGISTERS_NB : UT_INPUT_REGISTERS_NB;

            //Tab register
            tab_rp_registers = (uint16_t *) malloc(nb_points *
sizeof(uint16_t));
```

```
            memset(tab_rp_registers, 0, nb_points * sizeof(uint16_t));

            //Read input registers
            rc = modbus read registers(ctx, 200,UT INPUT REGISTERS NB,
                                       tab_rp_registers);

            printf("Modbus_read_input_registers: ");

            if (rc != UT_INPUT_REGISTERS_NB) {
                printf("FAILED (nb points %d)\n", rc);
                goto close;
            }


            //Values traitment
            for (i=0; i < UT_INPUT_REGISTERS_NB; i++) {

                //If tab of values is full
                if(n>9){
                    tabFull =true;
                    n=0;
                }

                //Keep the 10 last values of registers in a tab
                TAB_RP_REG[n]= tab_rp_registers[i];
                n++;



                //if tab is full => Calcul average of the 10 last values
                if( tabFull ){
                    average = CalculAverage(TAB_RP_REG, 10);
                //if tab is not full => Calcul average with the values
already obtained
                } else{
                    average = CalculAverage(TAB_RP_REG, (n));
                }


                //Print
                printf("value = %d\n",tab_rp_registers[i]);
                printf("average = %d\n",average);
                //write data on csv file
                fprintf(file,"%d,%d\n",tab_rp_registers[i],average );

            }

            printf("OK\n");

            /* Close the connection */
            modbus_close(ctx);

            //New connection
             ctx = modbus_new_tcp("192.168.1.119", 502);


           if (modbus_connect(ctx) == -1) {

                fprintf(stderr, "Connection failed:
%s\n",modbus_strerror(errno));
                modbus_free(ctx);
```

```
                return -1;
            }
               //Write the average in the register
            rc = modbus_write_register(ctx, 400,average);


            sleep(1);

            /* Close the connection */
            modbus_close(ctx);
            fclose(file);

        }

    close:
        /* Free the memory */

        free(tab_rp_registers);
        free(TAB_RP_REG);
        free(date);
        free(fileName);
        /* Close the connection */
        modbus_close(ctx);
        modbus_free(ctx);


 return out;

}

/*=====================================================================*/
/* Fin de fichero ModbusTCPTest.cpp
*/
/*=====================================================================*/
```

WeMos D1

The WeMos D1 is a card based on the ESP8266 module and with Wi-Fi. It works like an Arduino system, with the Arduino IDE. To configure the Arduino IDE, I followed this tutorial : http://www.instructables.com/id/Programming-the-WeMos-Using-Arduino-SoftwareIDE/

The main steps are :

- Look the differences in pin assignments between Arduino Uno and WeMos D1.
- Connect the WeMos to the computer
- Telling Arduino where to find the WeMos Library
- Install the WeMos board
- Configure the parameters

For the Modbus library, it's from Github of Andre Sarmento, but it's a library for ESP8266:

https://github.com/itead/ITEADLIB_Arduino_WeeESP8266

Firstly, the WeMos must be connected to the Raspberry with Wi-Fi, specifying the SSID and password. After that, the WeMos is on the same network of the Raspberry, with a static ip (192.168.1.119 for me) and so, we can launch the Modbus communication.

It's like on Arduino, you have to define the Input/Output, the registers used and to write / read the values with the Modbus protocol.

```
//Librairies
#include <ESP8266WiFi.h>
#include <Modbus.h>
#include <ModbusIP_ESP8266.h>

// Modbus Registers Offsets (0-9999)
const int HREG_1 = 200;
const int HREG_2 = 400;

//Used Pins
const int sensorPin = A0;
const int outputPin = D7;

//ModbusIP object
ModbusIP mb;

//Time
long ts;

void setup() {
  //Serial config
  Serial.begin(115200);
  //Connection to Wi-Fi
  mb.config("RPi2_Test", "raspberry");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  //register 1
  mb.addHreg(HREG_1);
  //register 2
  mb.addHreg(HREG_2);

  //Input
  digitalWrite(sensorPin, LOW);
  digitalWrite(sensorPin, INPUT_PULLUP);  // set pullup on analog pin 0
  //Output
  pinMode(outputPin, OUTPUT);
  //Take time
  ts = millis();

}

void loop() {

   //Call once inside loop() - all magic here
   mb.task();

   //Read the average of the last values of potentiometer [0 - 1024]
provided by the Raspberry in the register 2
  //And use this value for the LED
   analogWrite(outputPin, (mb.Hreg(HREG_2)/4) );


    if (millis() > ts +500 ) {
       ts = millis();

       //Add the values of potentiometer in the register 1
       mb.Hreg(HREG_1, analogRead(sensorPin));
       //Display
       Serial.print("\nval = ");
       Serial.print(analogRead(sensorPin));
       Serial.print("\naverage = ");
       Serial.print(mb.Hreg(HREG_2));
       Serial.print("\n");

   }

}
```

The application is like with Ethernet communication. When the Raspberry is powered on, the Wi-Fi is launching and the WeMos connects. Each 500ms, the value of potentiometer is writing in a register. The Raspberry read this register, stock the values in a table and writes them in a .CSV File. The average of the 10 last values is calculated and is writing in another register. The WeMos reads this average and sends the value to turn on the LED.

Conclusion

This project is interesting to learn how to work the Modbus communication, over TCP/IP, and how to use Raspberry and Arduino systems. The Modbus protocol is not hard to implement and can be used for a lot of application (industrial IT, Automaton systems, connected objects and more). It's the same for the Raspberry and Arduino, the applications can be limitless and the community on the web is rich ; they are many tutorials, project and this allows to improve and share our skills on forums. Moreover, Raspberry and Arduino are increasingly used in company to develop prototypes. Their affordable prices allow them to be accessible to all, for personal or professional uses.

Sources

- http://www.ostafichuk.com/raspberry-pi-projects/modbus-on-the-pi/modbus-on-the-pi-part-1-compiling-a-basic-example/
- https://en.wikipedia.org/wiki/Modbus
- http://cs.smith.edu/dftwiki/index.php/Tutorial:_Client/Server_on_the_Raspberry_Pi
- https://www.worldofgz.com/electronique/communication-modbus-tcp-arduino/
- http://educ8s.tv/arduino-esp8266-tutorial-first-look-at-the-wemos-d1-arduino-compatible-esp8266-wifi-board/
- http://www.instructables.com/id/Programming-the-WeMos-Using-Arduino-SoftwareIDE/
- http://www.pihomeserver.fr/2013/05/24/raspberry-pi-home-server-donner-une-ip-statique-a-votre-machine/
- http://raspberrypihq.com/how-to-add-wifi-to-the-raspberry-pi/
- http://www.dericbourg.net/2015/07/04/utiliser-un-raspberry-pi-comme-point-dacces-wifi/
- http://www.framboise314.fr/mettez-un-routeur-dans-votre-raspberry-pi-ou-linverse/
- https://pimylifeup.com/raspberry-pi-wireless-access-point/
- http://jamod.sourceforge.net/kbase/protocol.html
- http://libmodbus.org/
- https://github.com/andresarmento/modbus-arduino
- https://github.com/andresarmento/modbus-esp8266