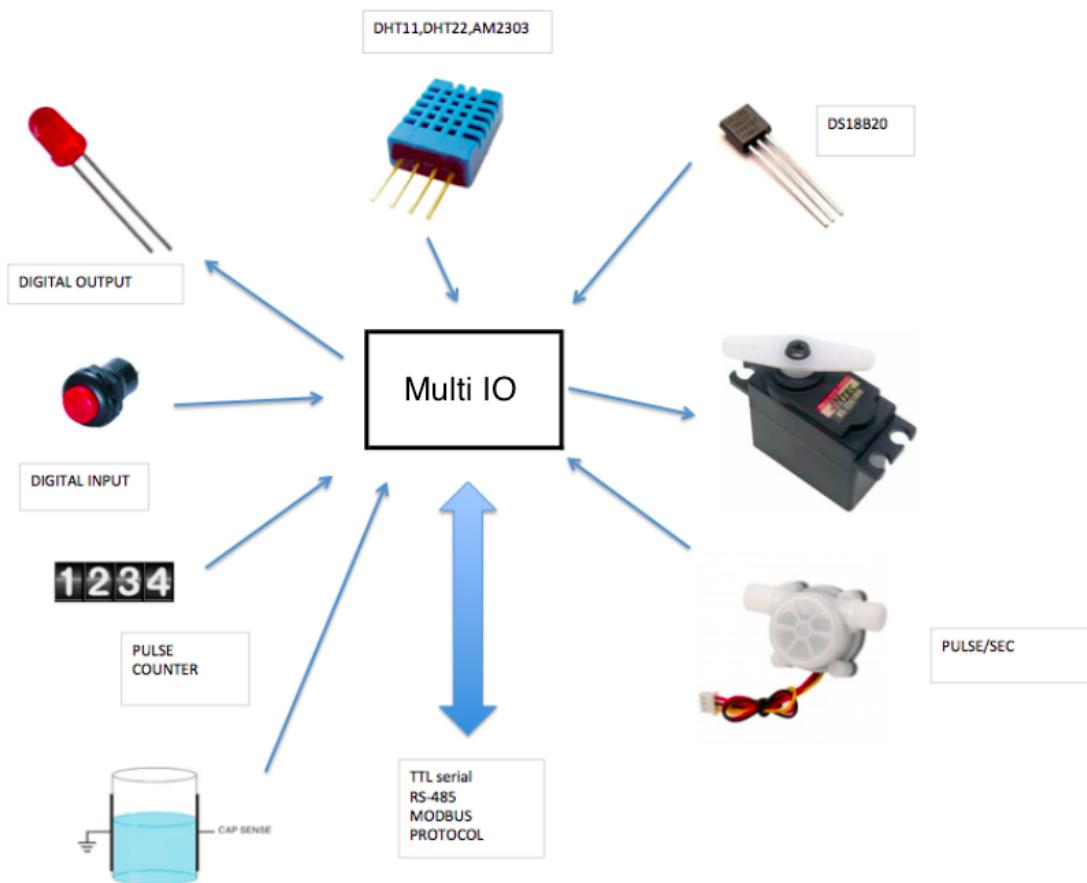


# Extension RS-485 avec Entrées/Sorties configurable depuis un petit cpu PIC



Daniel Perron  
Avril 2014

# Préface

Souvent nous avons besoin de brancher des Entrées ou sorties qui sont hors de portée d'un simple signal TTL. Alors il faut convertir le signal pour qu'il puisse voyager sur une plus longue distance. J'ai donc choisi le mode RS-485 avec le protocole modbus. De cette façon il est possible d'utiliser un seul câble qui va d'une périphérique à l'autre ce qui réduit considérablement la complexité du système. Il est possible, en théorie , d'avoir 247 modules attachés sur un serveur. En réalité, il y a une limite physique qui est dépendante du choix de la puce RS-485 utilisée. Un maximum de 64 modules est donc recommandé mais il est possible d'en avoir plus.

Un cpu à 18 broches, PIC16F1827, permet de réduire considérablement la dimension et la consommation en énergie du module.

Une possibilité de 10 entrées et sorties qui peuvent être configurées selon nos besoins spécifiques.

- Conversion A/D 10 bits, avec 4 différents niveaux de référence.
- Lecture de température avec des capteurs DHT11, DHT22 ou AM2303.
- Lecture de température avec des capteurs DS18B20.
- Lecture de fréquence utilisant l'oscillateur interne avec effet capacitif.
- Compteur et lecture de fréquence du signal logique.
- Simple entrée ou sortie digitale.
- Quatre sorties PWM physiques à 10 bits de résolution.
- Sortie pour R/C servo.

Une compréhension du protocole Modbus n'est pas nécessaire mais elle est suggérée. J'ai une référence pour Modbus à la fin du manuel.

# Step 1 - Préparation du Raspberry Pi

La communication série peut se faire par un adaptateur USB ou par l'interface interne du Raspberry PI. Pour l'utilisation série interne du Raspberry PI nous avons besoin de désactiver la console tty série et aussi celle du mode "débogue" lors de l'amorçage du Raspberry PI.

P.S. Pour les adaptateurs USB , passez à l'étape 3.

1 - Modifions le fichier `/boot/cmdline.txt` pour enlever la console `/dev/ttAMA0`.

- Désactivons le port série.

```
pi@raspberrypi ~ $ sudo raspi-config
```

- Interfacing option
- Serial
- Non (pour pas de console)
- Oui pour activer le port série.
- OK
- Finish
- Oui pour reboot

3 - Installation du module Python minimalmodbus.

- Faire une mise à jour.

```
pi@raspberrypi ~ $ sudo apt-get update
```

- Installons python3-pip.

```
pi@raspberrypi ~ $ sudo apt-get install python3-pip
```

- Installons minimalmodbus.

```
pi@raspberrypi ~ $ sudo pip3 install -U minimalmodbus
```

#### 4 - Installation intelhex.

- Méthode facile

```
pi@raspberrypi ~ $ sudo pip3 install intelhex
```

- Méthode alternative,

- Allons chercher le code source.

```
pi@raspberrypi ~ $ wget
```

```
http://www.bialix.com/intelhex/intelhex-1.5.zip
```

- Décompressons.

```
pi@raspberrypi ~ $ unzip intelhex-1.5.zip
```

- Installons le module.

```
pi@raspberrypi ~ $ cd intelhex-1.5/
```

```
pi@raspberrypi ~/intelhex-1.5 $ sudo python setup.py install
```

```
pi@raspberrypi ~/intelhex-1.5 $ cd
```

#### 5 - Installation libmodbus

```
pi@raspberrypi ~ $ sudo apt-get install libmodbus5 libmodbus-dev
```

6- Allons chercher BurnLVP. Le code Python pour programmer le cpu avec le Raspberry PI.

```
pi@raspberrypi ~ $ git clone https://github.com/danjperron/burnLVP.git
```

7- Téléchargeons PIC\_MULTI\_10\_IO de github

```
pi@raspberrypi ~ $ git clone https://github.com/danjperron/PIC_MULTI_10_IO.git
```

8- Reboot

```
sudo reboot
```

## Step 2 - Programmer le cpu

La puce PIC16F1827 de Microchip a une mémoire non volatile de 2 x 4 kilo octets. Il est possible d'utiliser une faible tension pour brûler le programme en mode (LVP). Nous allons utiliser le Raspberry Pi pour insérer le code binaire directement dans le cpu. L'application python burnLVPx permet d'utiliser les GPIOs du Raspberry Pi pour programmer les cpus de Microchip.

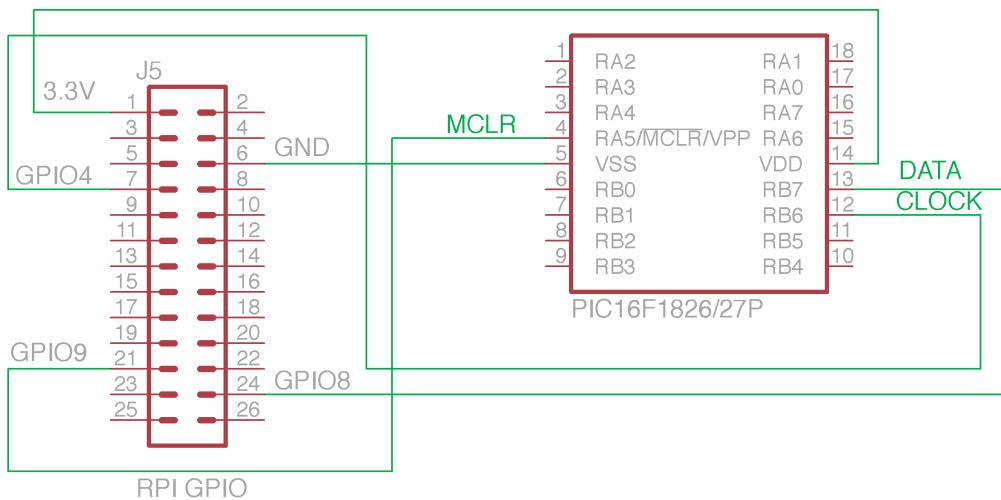


Figure 1. Schéma de connexion pour programmer le cpu avec le Raspberry Pi.

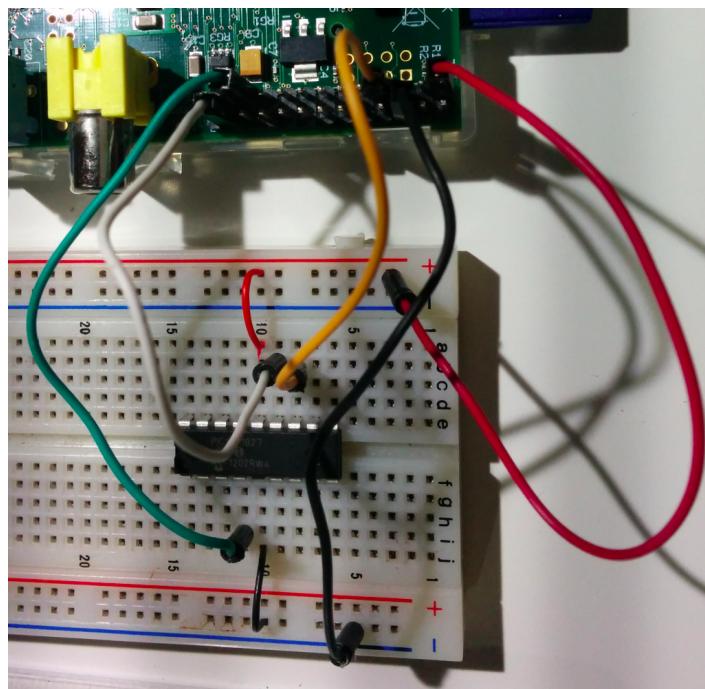


Photo 1. Le PIC16F1840 prêt à être programmé.

Il y a aussi le connecteur spécial ICSP(In circuit serial programming). Ce connecteur est très commun pour les processeurs PIC puisqu'il permet de programmer le cpu après montage.

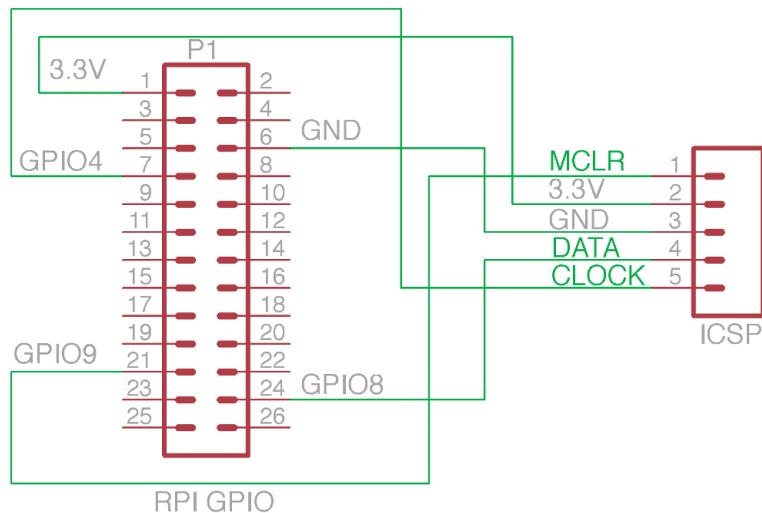


Figure 2. Schéma de connexion pour le connecteur ICSP.

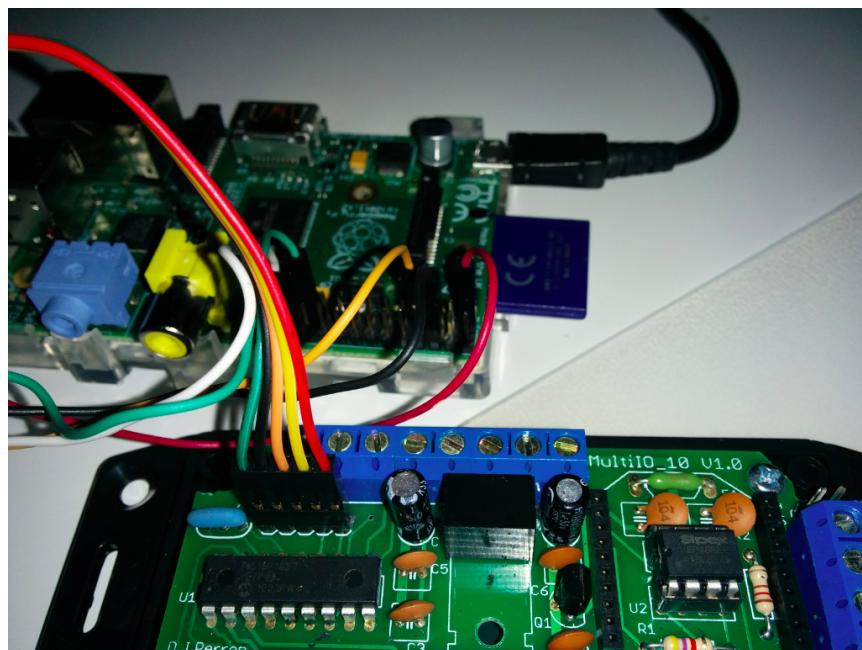


Photo 2. Connection sur le connecteur ICSP.

N.B. Pour le PCB multi IO 10 , aucune autre source d'alimentation. Les connecteurs pour IO3 et IO4 doivent être libres.

1 - Pour programmer le CPU , nous allons utiliser l'application python burnLVPx.py

- Branchez le cpu avec le Raspberry Pi selon la photo 1 ou 2.
- Démarrez l'application burnLVPx avec le fichier hex correspondant au programme désiré.

```
pi@raspberrypi ~ $ cd  
pi@raspberrypi ~ $ sudo ./burnLVP/burnLVPx.py PIC_MULTI_10_IO/FirmwareV1.02_57600Baud.hex
```

La sortie console de l'application devrait être

```
File " PIC_MULTI_10_IO/FirmwareV1.02_57600Baud.hex " loaded  
Scan CPU  
Check PIC12/16...  
Cpu Id = 0x0  
Revision = 0x5  
Found PIC16F1827 from Cpu Family PIC12/16  
Cpu Id: 0x27a0 revision: 5  
Bulk Erase Program , Data. .... done.  
Program blank check.....Passed!  
Data Blank check.....Passed!  
Writing Program.....Done.  
Program check .....Passed!  
Writing Data.Done.  
Data check .Passed!  
Writing Config..Done.  
Config Check..Passed!  
Program verification passed!
```

- Débranchez la tension et enlevez le cpu du protoboard ou débranchez l'ICSP.
- Ajoutez une étiquette sur le cpu pour marquer qu'il est programmé.

## Step 4 - Vérification du cpu

Une fois le cpu programmé, il est maintenant possible d'utiliser le Raspberry Pi avec son port série interne pour vérifier la programmation. Alimenté avec 3.3V et connecté sur les broches de communication série, le cpu devrait répondre au protocole Modbus . L'interface RS-485 n'est pas nécessaire puisqu'il y a seulement un cpu donc nous n'avons pas besoin de débrancher la connexion.

Selon le programme utilisé dans le cpu , il est peut être nécessaire d'ajouter un résonateur. C'est le cas pour le fichier **Firmware\_noX\_V1.06\_57600Baud.hex**

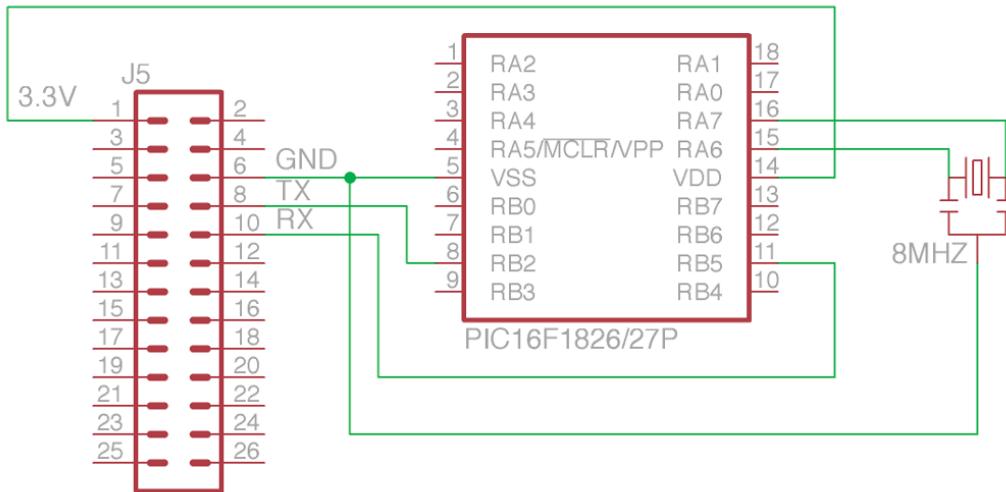


Figure 3. Cpu en mode test directement sur /dev/ttyAMA0.

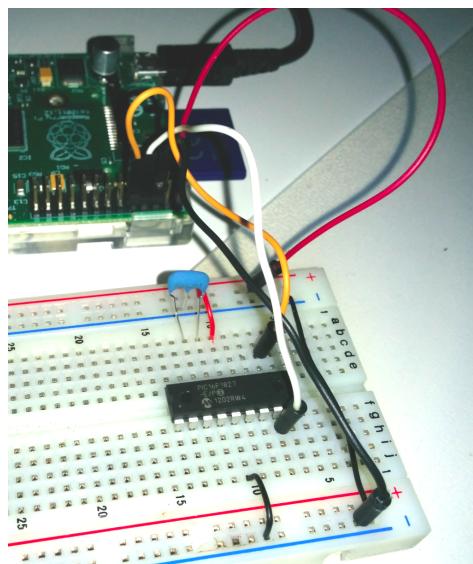


Photo 3. Le cpu en mode de communication directe.

L'utilitaire "configPIC" va permettre de vérifier le cpu. Pour cela il faudra le compiler.

```
pi@raspberrypi ~/PIC_MULTI_10_IO $ gcc -I /usr/include/modbus \
> configPIC.c -o configPIC -l modbus
```

P.S. le tout peut être sur une ligne

```
gcc -I /usr/include/modbus configPIC.c -o configPIC -l modbus
```

Maintenant, exécutons le programme avec les paramètres appropriés

```
pi@raspberrypi ~/PIC_MULTI_10_IO $ ./configPIC -d /dev/serial0 -b 57600
```

-d spécifie le port de communication.

-b spécifie la baud rate.

-t spécifie le délai d'attente en microseconde. (time out)

pour la communication XBee, il est préférable d'ajouter -t 200000

L'application devrait répondre ,

```
PIC multi-purpose I/O MODBUS configuration
Version 1.0 (c) Daniel Perron, April 2014
device:/dev/ttyAMA0 Baud Rate:57600
```

```
M) MODBUS scan
F) FLUSH buffer
A) Select Slave Module
C) Change Slave Address
0..9) Set IO mode
Q) Quit
```

Selected module :None

Appuyer sur la touche 'M' du clavier génère un scan complet du système et si tout est OK, la réponse donnera ceci,

```
==== Scanning MODBUS
127 : Type 653A Multi Purpose 10IO
```

Et maintenant il nous reste à changer l'adresse Modbus du module pour ne pas être en conflit avec d'autres modules. Il serait temps de détailler la configuration des modules.

Nous allons changer l'adresse 127, qui est la configuration par défaut pour 1. Avec l'application configPIC toujours en fonction, nous allons choisir le module 127 en appuyant sur la touche 'A' et en tapant 127 suivi de la touche retour.

```
Select Module  
Enter Slave Address ?127
```

L'application répondra avec la sélection choisie.

```
Selected module :127
```

Maintenant changeons l'adresse modbus pour 1. Appuyons la touche 'C' et tapons 1 suivie de la touche de retour (ENTER).

```
=====Change Address  
Enter new Slave Address for this module (1..127) ?1  
Module is now on Address 1
```

L'application permet de configurer chaque E/S selon le mode voulu. Les touches 0 à 9 indiquent quelle broche.

ex: pour modifier E/S IO0 pour une lecture analogique 2V.

- 1 - Choisir le module désiré. La touche 'A' suivi de l'adresse du module.
- 2 - Choisir l'E/S désiré. La touche numérique.

L'application donnera une liste de possibilités et il restera à choisir.

J'appuie sur '0' et ensuite je choisis 4 suivis de la touche retour.

```
Selected module :1  
===== Change IO0 mode  
0) ANALOGVDD      1) ANALOG1V      2) ANALOG2V  
3) ANALOG4V       4) INPUT        5) INPUT PULLUP  
6) OUTPUT         7) PWM          8) CAP SENSE OFF  
9) CAP SENSE LOW   10) CAP SENSE MEDIUM  11) CAP SENSE HIGH  
16) DHT11         17) DHT22       32) DS18B20  
64) R/C SERVO     128) COUNTER  
Slave address 1 current IO0 mode is 4 : INPUT  
Enter new configuration ?2  
Module 1 IO0 set to 2: ANALOG2V
```

La touche ‘I’ permet d'afficher l'information des E/S d'un module sélectionné.

```
Selected module :127
=====
127 : Type 653A Multi Purpose 10IO
IO0: DHT22          Temp: 24.2 Celsius  Humidity: 57.1%
IO1: CAP SENSE HIGH [107 (0x006B)] [27481 (0x6B59)]
IO2: COUNTER         [0 (0x0000)] [0 (0x0000)] [0 (0x0000)]
IO3: COUNTER         [2 (0x0002)] [26823 (0x68C7)] [50 (0x0032)]
IO4: INPUT PULLUP    [1 (0x0001)]
IO5: ANALOG1V        [1023 (0x03FF)]
IO6: ANALOG2V        [672 (0x02A0)]
IO7: OUTPUT           [0 (0x0000)]
IO8: R/C SERVO       [1000 (0x03E8)]
IO9: DS18B20          Temp: 23.8 Celsius
```

# Les modes Entrée/Sorties et modbus

IO	PORT	Sortie 6	Entrée 4	Entrée + pullup 5	Analogique 0,1,2,3	DHT11/DHT22 16,17	DS18B20 32	PWM 7	Effet capacatif 8,9,10,11	R/C Servo 64	Compteur 128
0	RB3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1	RB1	✓	✓	✓	✓	✓	✓		✓	✓	✓
2	RB4	✓	✓	✓	✓	✓	✓		✓	✓	✓
3	RB6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	RB7	✓	✓	✓	✓	✓	✓		✓	✓	✓
5	RA0	✓	✓		✓		✓			✓	
6	RA1	✓	✓		✓		✓			✓	
7	RA2	✓	✓		✓		✓			✓	
8	RA3	✓	✓		✓		✓	✓		✓	
9	RA4	✓	✓		✓		✓	✓		✓	

Figure 4. Table de configuration

La configuration des E/S est stockée dans le flash interne du cpu. La configuration ne sera pas perdue lors de la mise hors tension.

Voici la liste des constantes Python reliées à la configuration.

```

IOCONFIG_ANALOGVDD= 0
IOCONFIG_ANALOG1V= 1
IOCONFIG_ANALOG2V= 2
IOCONFIG_ANALOG4V= 3
IOCONFIG_INPUT= 4
IOCONFIG_INPUT_PULLUP= 5
IOCONFIG_OUTPUT= 6
IOCONFIG_PWM= 7
IOCONFIG_CAP_SENSE_OFF= 8
IOCONFIG_CAP_SENSE_LOW=9
IOCONFIG_CAP_SENSE_MEDIUM=10
IOCONFIG_CAP_SENSE_HIGH=11
IOCONFIG_DHT11= 16
IOCONFIG_DHT22= 17
IOCONFIG_DS18B20= 32
IOCONFIG_SERVO= 64
IOCONFIG_COUNTER= 128
IOCONFIG_COUNTER_PULLUP= 129

```

Lors de la lecture des registres dédiés à chaque IO, les valeurs retournées seront appropriées au mode de configuration choisi. C'est pour cela que les registres lues seront sur une base identification multipliée par 16 puisque certaines configurations vont retourner plus d'un paramètre.

L'exception sera toujours la configuration digitale d'un simple IO. L'identité de chaque IO est direct. Cela va pour les fonctions modbus 1,2 et 5.

P.S. Base d' identification => IO0 = 0, IO1= 1 , ..., IO9=9

## Le mode de conversion d'entrée analogique.

IOCONFIG_ANALOGVDD	=0	( VRef= VDD)
IOCONFIG_ANALOG1V	=1	( VRef=1.024V)
IOCONFIG_ANALOG2V	=2	( VRef=2.048V)
IOCONFIG_ANALOG4V	=3	( VRef=4.096V)

Une valeur de 10 bits sera convertie selon la référence désignée.

La formule est

Voltage de lecture = Valeur A/D \* Voltage de référence / 1023;

### Python

```
readSensor(Pin)      => cette fonction provient du module PicModule.py

Ceci appelle la fonction
module.read_registers(Pin * 16 , 1, 4)[0]
```

Lire un registre à la position (16 \* Pin) avec la fonction Modbus 4

### Language C

```
uint16_t MB_Register[3];
if(modbus_read_input_registers(mb,Pin*16,1,MB_Register)>=0)
{
    // le registre MB_Register[0] contient la valeur 10 bits analogue
}
```

ex: Pour lire un capteur de température sur I00 avec un module à l'adresse 1.

```
import time
import PicModule
remote1 = PicModule.PicMbus(1,Baud=57600,Device='/dev/serial0')

#ioconfig are set once and store into the eeprom
# we just need to check it and change it if it is not ok
ioconfig = remote1.readConfig(0)
if ioconfig != remote1.IOCONFIG_ANALOG1V:
    remote1.config(0,remote1.IOCONFIG_ANALOG1V)
    time.sleep(0.3) #need to wait for eeprom writing

AnalogValue= remote1.readSensor(0)
print("TMP35 : {:.1f}°C".format(AnalogValue/10.0))
```

Deux registres spéciaux utilisent la conversion analogique pour récupérer la tension de l'alimentation ainsi que la température du cpu.

Modbus registre 0x1000 => retourne la valeur numérique de la référence 2.048V avec la tension VDD comme référence.

### Python

```
readVRef2V()  
  
module.read_registers(0x1000,1,4)[0]
```

### language C

```
uint16_t MB_Register[3];  
if(modbus_read_input_registers(mb,0x1000,1,MB_Register)>=0)  
{  
    // le registre MB_Register[0] contient la valeur 10 bits analogue  
}
```

ex: Lire la Tension de l'alimentation CPU.

```
import PicModule  
remote1 = PicModule.PicMbus(1,Device='/dev/serial0')  
A2D_VRef = remote1.readVRef2V()  
print("VDD : {:.5.2f}V".format(2.048*1023.0/A2D_VRef))
```

C'est la même méthode pour lire la diode thermique dans le cpu

### Python

```
readDiode()  
  
module.read_registers(0x1000,1,4)[0]
```

### language C

```
uint16_t MB_Register[3];  
if(modbus_read_input_registers(mb,0x1001,1,MB_Register)>=0)  
{  
    // le registre MB_Register[0] contient la valeur 10 bits analogue  
}
```

## Le mode d'entrée numérique

```
IOCONFIG_INPUT          =4  
IOCONFIG_INPUT_PULLUP   =5  ( Weak pull up from cpu)
```

Le mode IOCONFIG\_INPUT\_PULLUP est seulement disponible pour les IO0,IO1,IO2,IO3 et IO4. Un faible courant maintient les entrées sur VDD.

Chaque IO est défini comme une entrée numérique, 0 ou 1.

### Python

```
readIO(Pin)  
  
module.read_bit(Pin)
```

### Language C

```
uint16_t MB_Register[3];  
if(modbus_read_input_registers(mb,Pin*16,MB_register)>=0)  
{  
    ...= MB_Register[0]; // retourne 0 ou 1  
  
}
```

N.B. Exception - L'IO en mode entrée ou sortie peut être lu avec la fonction Modbus 4 mais il faut multiplier par 16 l'identifiant.

ex: Pour lire IO7

```
#Assumons que la configuration est déjà en mode d'entrée numérique  
  
import PicModule  
remote1 = PicModule.PicMbus(1,Baud=57600,Device='/dev/serial0')  
  
print("IO0: {}",remote1.readIO(0))
```

## Le mode de sortie numérique

```
IOCONFIG_OUTPUT          =6
```

Ce mode permet d'utiliser les IO en sortie.

### Python

```
writelO(Pin, valeur)  
  
module.write_bit(Pin,valeur)
```

### Language C

```
if(modbus_write_bit(mb,Pin,Value)>0)  
{  
    ...c'est fait!  
}
```

ex: Créer un trois pulsations sur IO1 avec le modules dont l'adresse est 2 et branché sur un adaptateur RS-485 USB.

```
import time  
import PicModule  
remote2 = PicModule.PicMbus(2,Baud=57600,Device='/dev/ttyUSB0')  
#Assumons que la configuration est déjà en mode sortie  
for pulse in range(3):  
    remote2.writelO(1,1)  
    time.sleep(0.1)  
    remote2.writelO(1,0)  
    time.sleep(0.1)
```

## Le mode de captation capacitif

IOCONFIG_CAP_SENSE_OFF	=8
IOCONFIG_CAP_SENSE_LOW	=9
IOCONFIG_CAP_SENSE_MEDIUM	=10
IOCONFIG_CAP_SENSE_HIGH	=11

Un oscillateur est généré à chaque IO et la fréquence de l'oscillation dépend de la valeur capacitive engendrée par la connexion au IO. Quatre modes de puissance sont disponibles dont un n'a aucune oscillation. La valeur retourné correspond au nombre de pulsations, au format 32 bits non signé, à chaque 1/10 de seconde.

### Python

```
readSensor(Pin)  
  
module.read_long(Pin * 16,4,False)
```

ex: Lire la valeur de l'oscillation de IO0 et IO1

```
import time  
import PicModule  
remote1 = PicModule.PicMbus(1)  
  
#assumons IOCONFIG1 et IOCONFIG0 sont en mode IOCONFIG_CAP_SENSE_HIGH  
  
print("Cap sense I00 : {}".format(remote1.ReadSensor(0))  
print("Cap sense I01 : {}".format(remote1.ReadSensor(1))
```

La constante diélectrique varie beaucoup d'un matériel à l'autre. Ce mode peut être utilisé pour calculer le niveau d'un réservoir. Il s'agira de calibrer l'information. Il faudra toutefois compenser pour le changement de température.

## Le mode du capteur DHT

IOCONFIG_DHT11	=16
IOCONFIG_DHT22	=17 (Also AM2303 type)

Ce mode permet de lire un capteur de température du type DHT22 et DHT11. Deux registres sont retournés. Soit la température et l'humidité.

### Python

```
readDHT(Pin)
    retourne None => pas de capteur
    retourne [humidité, Température]

module.read_register(Pin * 16,3,4)
    retourne
        [0] 0xFFFF => pas de capteur 1=> capteur OK
        [1] Humidité en valeur numérique et non en pourcentage
        [2] Température en valeur numérique et non en celsius.
```

### Language C

```
void ReadDHT22(modbus_t * mb,int _io)
{
    float Factor,Temperature,Humidity;
    uint16_t MB_Register[3];
    if(modbus_read_input_registers(mb,_io*16,3,MB_Register)>=0)
    {
        if(MB_Register[0]==0)
            printf("Buzy");
        else if (MB_Register[0]==1)
        {
            if(MB_Register[2] & 0x8000)
                Factor = (-0.1);
            else
                Factor = (0.1);
            Temperature = (MB_Register[2] & 0x7fff) * Factor;
            Humidity = (MB_Register[1] * 0.1);
            printf("Temp: %5.1f Celsius   Humidity: %5.1f%%",Temperature,Humidity);
        }
        else printf("Error");
    }
    else
        printf("Unable to read DHT Sensor");
    printf("\n");
}
```

ex: Lire un DHT22 sur IO 0

```
import time
import PicModule
remote1 = PicModule.PicMbus(1)

result = remote1.readDHT(0)
if result is None:
    print("Error ON DHT Sensor! Bad Config or sensor not connected")
else:
    print("Humidity: {5.1f}%".format(result[0]))
    print("Temperature:{5.1f}°C".format(result[1]))
```

P.S. Le capteur DHT11/22 est capricieux et il ne fonctionnera pas si l'IO n'est pas en tension élevée lors de la connexion initiale. Il est préférable de configurer le module et ensuite de mettre tout hors tension avant d'installer le capteur DHT11/22.

La lecture du capteur DHT11/22 peut être problématique si la fonction de compteur et de R/C servo est enclenché puisque trop d'interruptions va perturber la temporisation du système.

## Le capteur DS18B20

IOCONFIG\_DS18B20 =32

Ce mode retourne la température du capteur DS18B20 attaché au IO. Seulement un capteur peut être attaché sur chaque IO.

N.B. Pour les IO5, IO6, IO7, IO8 et IO9 il faut ajouter un résistance en pull-up, environ 4k7, pour créer une tension positive en relâchement du capteur.

### Python

```
valeur, status = readDS18B20(Pin)
```

Le status retourne 1 si la valeur est correcte.

La signification du status est,

0 = occupé.

1 = valeur valide.

2 = erreur sur le CRC.

0xffff = mauvaise communication avec le capteur.

### Language C

```
void ReadDS18B20(modbus_t * mb,int _io)
{
    float Factor=0.0625;
    float Temperature;
    int mask;
    short Temp;
    uint16_t MB_Register[3];
    if(modbus_read_input_registers(mb,_io*16,3,MB_Register)>=0)
    {
        if(MB_Register[0]==0)
        {
            printf("Buzy");
        }
        else if (MB_Register[0]==1)
        {
            Temperature = Factor * ((short)MB_Register[1]);
            if((Temperature >= (-55)) && (Temperature <= 125))
                printf("Temp: %.1f Celsius",Temperature);
            else
                printf("Error");
        }
        else printf("Error");
    }
    else
        printf("Unable to read Sensor");
    printf("\n");
}
```

ex: Lire le capteur DS18B20 sur IO0

```
import time
import PicModule
remote1 = PicModule.PicMbus(1)
value, status = remote1.readDS18B20(0)
if status != 1:
    print("Error ON DS18B20 Sensor! Bad Config or sensor not connected")
else:
    print("Temperature:{6.2f}°C".format(result))
```

## **Le mode R/C Servo**

IOCONFIG\_SERVO =64

Ce mode permet de brancher des R/C servos.

La valeur du registre de positionnement est en microseconde.

Une valeur de 0 met à OFF le servo.

Les valeurs standard pour un minimum et maximum sont 500 et 2500µs  
il est possible de jouer entre 1 et 20000µs mais je ne garantie pas le bon fonctionnement.

### **Python**

```
RCServo(Pin,valeur)
```

### **Language C**

```
modbus_write_register(mb,Pin * 16, valeur);
```

ex: Bouger le R/C servo à la position centrale (1000us).

```
import PicModule
remote1 = PicModule.PicMbus(1)
remote1.RCServo(0,1000)
```

## Le mode compteur

```
IOCONFIG_COUNTER          =128  
IOCONFIG_COUNTER_PULLUP  =129
```

Ce mode permet de compter des pulsations numériques. Il y a aussi un registre d'indication de fréquence.

### Python

```
readSensor(Pin)  
  
[0]= MSB 16 bits Unsigned int totalisateur  
[1]= LSB 16 bits Unsigned int totalisateur  
[2]= Nombre de Pulsion par seconde. La fréquence.
```

ex: Read COUNTER on IO0

```
import PicModule  
remote1 = PicModule.PicMbus(1)  
#assume ioconfig on IO0 to be IOCONFIG_COUNTER  
result = remote1.readSensor(0)  
print("Pulse/sec = {}".format(result[2]));  
print("Total Count ={}".format(result[0] * 65536 + result[1]));
```

### Language C

```
unsigned long totaliser;  
uint16_t Fréquence;  
uint16_t MB_Register[3];  
if(modbus_read_input_registers(mb,Pin*16,3,MB_Register)>0)  
{  
    totaliser = ((unsigned long)MB_Register[0])<<16 | MB_Register[1];  
    Fréquence = MB_Register[2];  
}
```

Il y a moyen de faire une remise à zéro des compteurs. Il s'agit d'écrire les registres avec des zéros.

### python

```
resetCounter(Pin)  
  
module.write_register(Pin,0,0,6)
```

### Language C

```
modbus_write_register(mb,Pin,0);
```

# Une Page web pour 3 modules de 8 relais

Un petit projet simple pour asservir beaucoup de gadgets en même temps. Trois petites boîtes noires avec chacune huit relais que nous pouvons allumer ou éteindre en appuyant un bouton sur une page web. Le système consiste en un Raspberry Pi avec un adaptateur USB à RS-485 branché à trois modules qui ont chacun un multIO\_10 et une block the 8 Relais de 12V. Les modules MultIO ont tous les IO en sortie. Chaque relais est branché sur un connecteur 110V , le signal live , pour activer/désactiver un appareil.

Le serveur web est WebIOPI.

Voici la façon de l'installer.

```
pi@raspberrypi ~ $wget http://sourceforge.net/projects/webiopi/files/WebIOPI-0.7.0.tar.gz
pi@raspberrypi ~ $tar -xzvf WebIOPI-0.7.0.tar.gz
pi@raspberrypi ~ $cd WebIOPI-x.y.z
pi@raspberrypi ~ $sudo ./setup.sh
```

Et nous allons le démarrer lors de l'amorçage du Raspberry Pi.

```
pi@raspberrypi ~ $sudo update-rc.d webiopi defaults
```

Un reboot ou un amorçage manuel est nécessaire

```
pi@raspberrypi ~ $sudo service webiopi start
```

Maintenant nous allons changer la configuration pour inclure notre scripts python pour accéder au modules et aussi changer la page d'accueil pour WebRelais.html

```
pi@raspberrypi ~ $cd /etc/webiopi
pi@raspberrypi ~ $sudo nano config
```

```
[SCRIPTS]
# Load custom scripts syntax :
# name = sourcefile
# each sourcefile may have setup, loop and destroy functions and macros
myscript = /home/pi/WebRelais.py

# Use welcome-file to change the default "Welcome" file
welcome-file = WebRelais.html
```

WebRelais.py est dans le github PICMulti\_10\_IO donc nous avons juste à le copier avec PicModule.py

```
pi@raspberrypi ~ $ cp /home/pi/PIC_MULTI_10_IO/WebRelais.py /home/pi
pi@raspberrypi ~ $ cp /home/pi/PIC_MULTI_10_IO/PicModule.py /home/pi
pi@raspberrypi ~ $ chmod +x /home/pi/WebRelais.py
pi@raspberrypi ~ $ chmod +x /home/pi/PicModule.py
```

Et la page web sera transférée à /usr/share/webiopi/htdocs

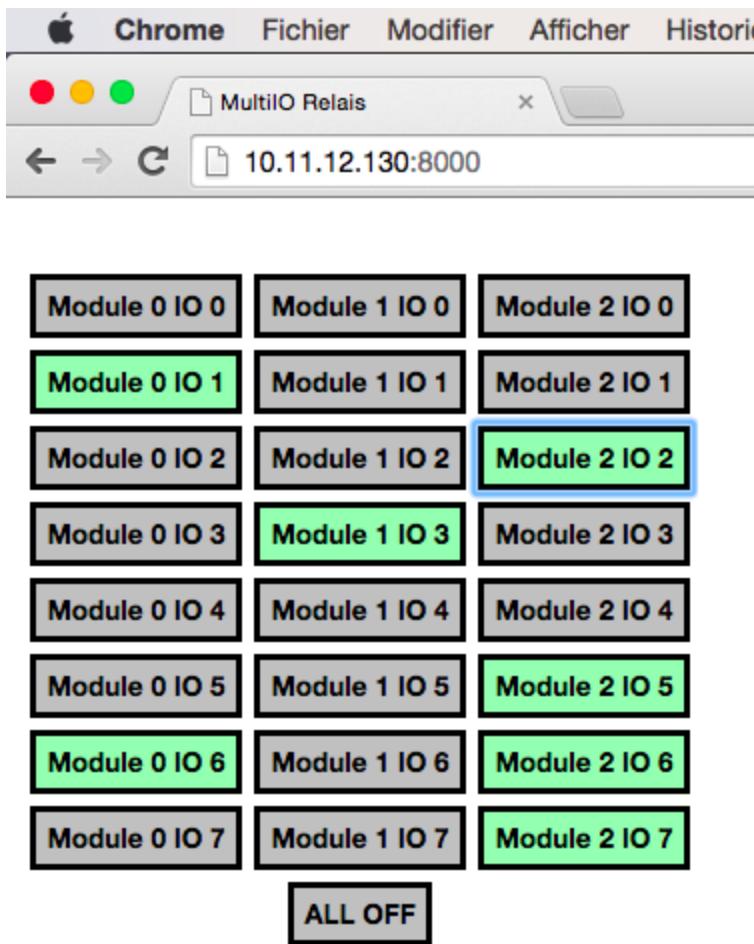
```
pi@raspberrypi ~ $ cd /usr/share/webiopi/htdocs
pi@raspberrypi ~ $ sudo cp /home/pi/PIC_MULTI_10_IO/WebRelais.html .
```

Et redémarrons webiopi

```
pi@raspberrypi ~ $ sudo service webiopi restart
```

La page web devrait être accessible sur L'IP du Raspberry PI sur le port 8000.

Nous pouvons aussi enlever le mot de passe en effaçant le fichier /etc/webiopi/passwd.



Puisque les modules ont 10 E/S , J'ai décidé d'ajouter un capteur de température DS18B20 sur le IO 8 avec le module 11. Il faut ajouter une résistance de 4700 ohm entre 5V et le signal.(Pull UP).

J'ai déjà du code venant d'un autre projet pour l'historique de température. Je vais donc modifier légèrement le code pour l'adapter au module.

Voici le lien au projet Rpi en time lapse.

[https://docs.google.com/document/d/1512wpeCToWxMwa8njtgMs\\_3\\_tu6Xpd77ZPfYTJvYB2M/edit?usp=sharing](https://docs.google.com/document/d/1512wpeCToWxMwa8njtgMs_3_tu6Xpd77ZPfYTJvYB2M/edit?usp=sharing)

Webiopi a l'usage exclusif du port série qu'utilise les modules modbus par l'entremise du script WebRelais.py . Avec l'ajout de code dans la fonction "loop()" nous allons pouvoir lire le capteur à

chaque minute et créer à partir de RRDTOOL et HIGHCHART un graphique représentant un historique de la température.

Installons HighChart et rrdtool.

```
pi@raspberrypi ~ $ sudo apt-get install rrdtool
pi@raspberrypi ~ $ wget http://code.highcharts.com/zips/Highcharts-3.0.9.zip
pi@raspberrypi ~ $ sudo mkdir /usr/share/webiopi/htdocs/charts
pi@raspberrypi ~ $ sudo unzip Highcharts-3.0.9.zip -d /usr/share/webiopi/htdocs/charts
```

Et vérifions l'installation avec l'url suivant

<http://raspberrypi.local/charts/index.htm> (attention pas .html mais .htm)

raspberrypi.local est l'adresse IP du Raspberry Pi. Il est possible d'utiliser avachi pour voir le nom du serveur sur le réseau.

Il s'agit d'installer libnss-mdns

```
pi@raspberrypi ~ $ sudo apt-get install libnss-mdns
```

De cette façon nous avons accès au Rpi avec RaspberryPi.local en mode local seulement.

Si vous voulez changer le nom du RPi. WebPI par exemple

```
sudo hostname WebPi
sudo nano /etc/hosts      et changeons raspberrypi to WebPi
sudo nano /etc/hostname et changeons raspberrypi to WebPi
et "reboot"
```

Maintenant que HightCharts et rrdtool sont installés, nous allons modifier WebRelais.py et ajouter historique.html.

```
pi@raspberrypi ~ $ sudo service webiopi stop
pi@raspberrypi ~ $ sudo rm /home/pi/WebRelais.py
pi@raspberrypi ~ $ cp /home/pi/PIC_MULTI_10_IO/WebRelais.2.py /home/pi/WebRelais.py
pi@raspberrypi ~ $ cd /usr/share/webiopi/htdocs
pi@raspberrypi ~ $ sudo cp /home/pi/PIC_MULTI_10_IO/historique.html .
```

Pour protéger la carte SD nous allons créer un disque de mémoire vive (ramdisk) pour manipuler les fichiers temporaire de rrdtool. Nous allons ajouter dans /etc/fstab une entrée pour un disque de 10 mégaoctets en mode tmpfs (RAM)

```
pi@raspberrypi ~ $ sudo mkdir /usr/share/webiopi/htdocs/temperature
pi@raspberrypi ~ $ sudo nano /etc/fstab
```

```
proc      /proc      proc  defaults      0      0
/dev/mmcblk0p1 /boot      vfat  defaults      0      2
/dev/mmcblk0p2 /      ext4  defaults,noatime 0      1
```

```
tmpfs /usr/share/webiopi/htdocs/temperature tmpfs defaults,noatime,nosuid,mode=0755,size=1m 0 0
```

La partition ram pointera sur /usr/share/webiopi/htdocs/temperature.

Démarrons le Rpi pour créer le nouveau répertoire

Nous allons ajouter la température du cpu dans la banque de données de rrdtool.

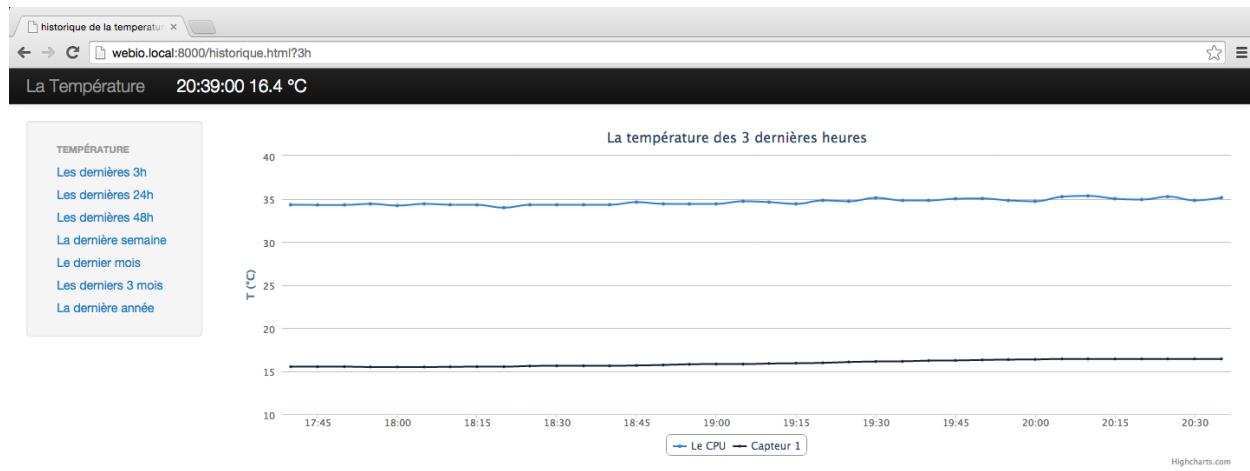
Créons le fichier de données temperature.rrd dans le répertoire /home/pi/data avec le script createdata.sh dans le répertoire /home/pi/**PIC\_MULTI\_10\_IO** ([le github](#))

```
pi@raspberrypi ~ $ mkdir data  
pi@raspberrypi ~ $ /home/pi/PIC_MULTI_10_IO/createdata.sh
```

Et finalement re-démarrons webiopi

```
pi@raspberrypi ~ $ sudo service webiopi start
```

Et utilisons un navigateur web pour regarder l'historique de la température.



# APPENDIX

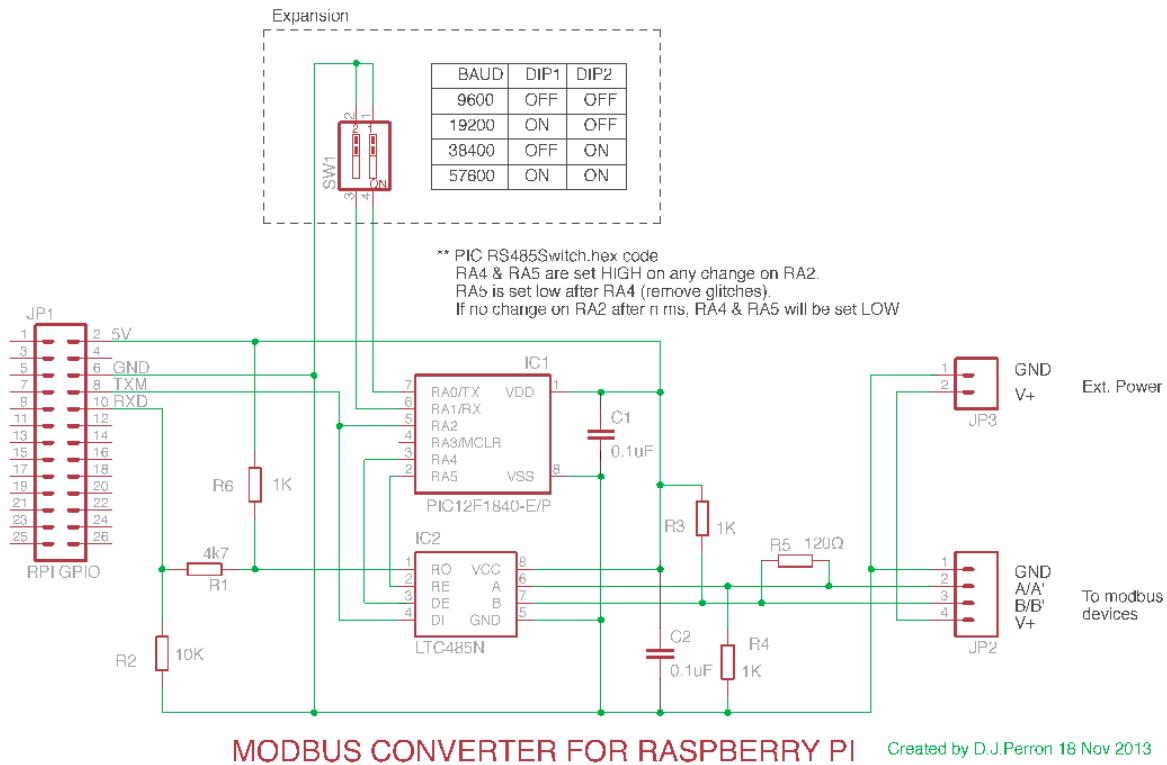
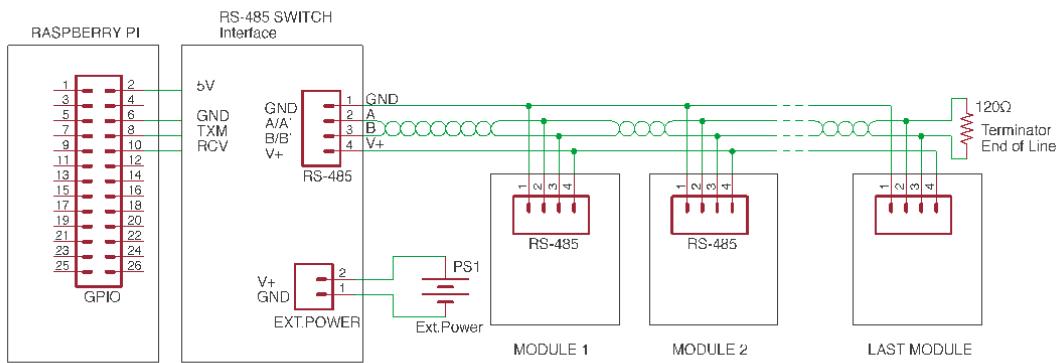
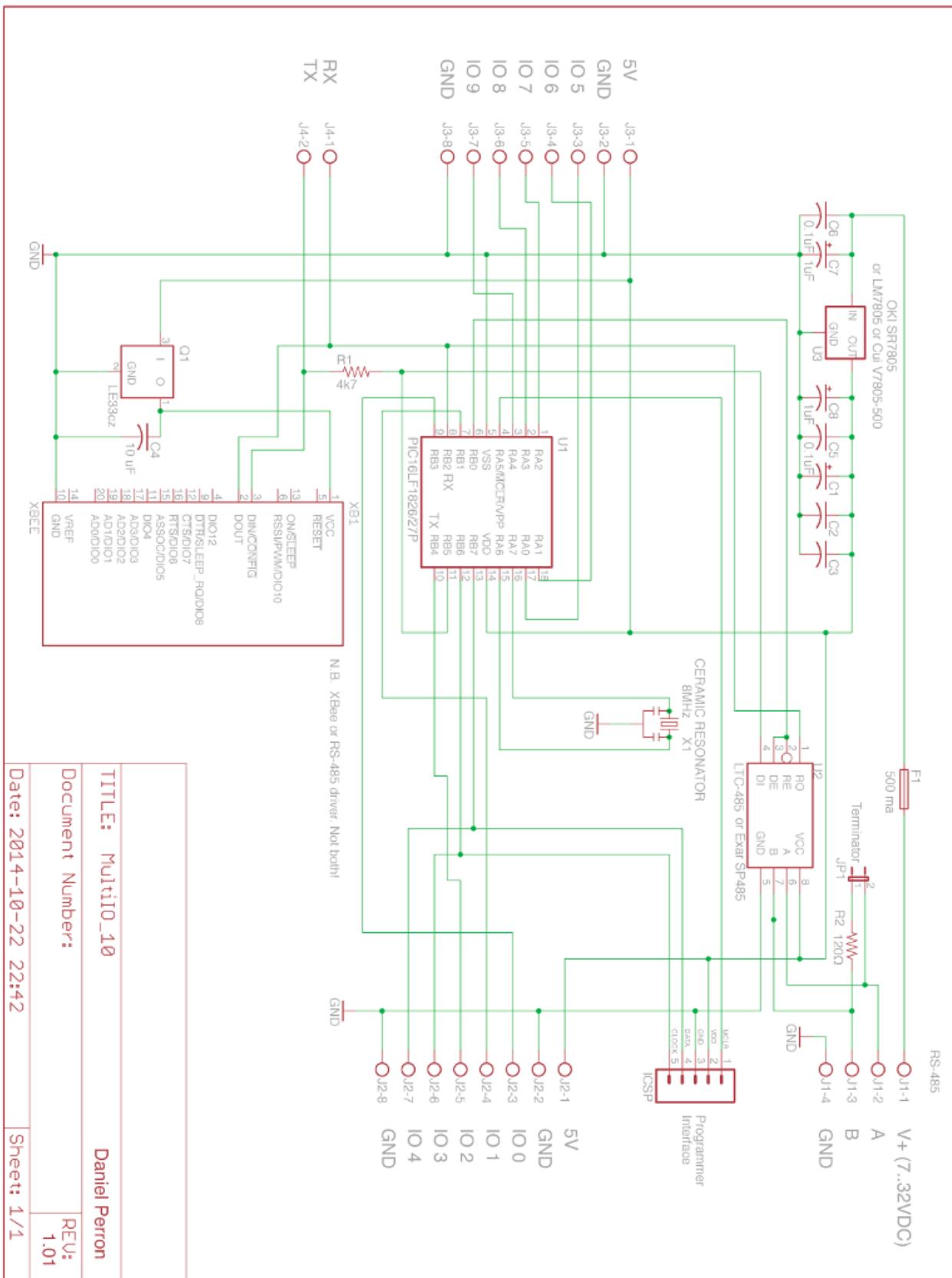


Figure 6- Interface RS-485 pour le Raspberry Pi



Raspberry Pi Modbus layout  
D.J.Perron 19 November 2013

Figure 7 - Trois modules modbus .



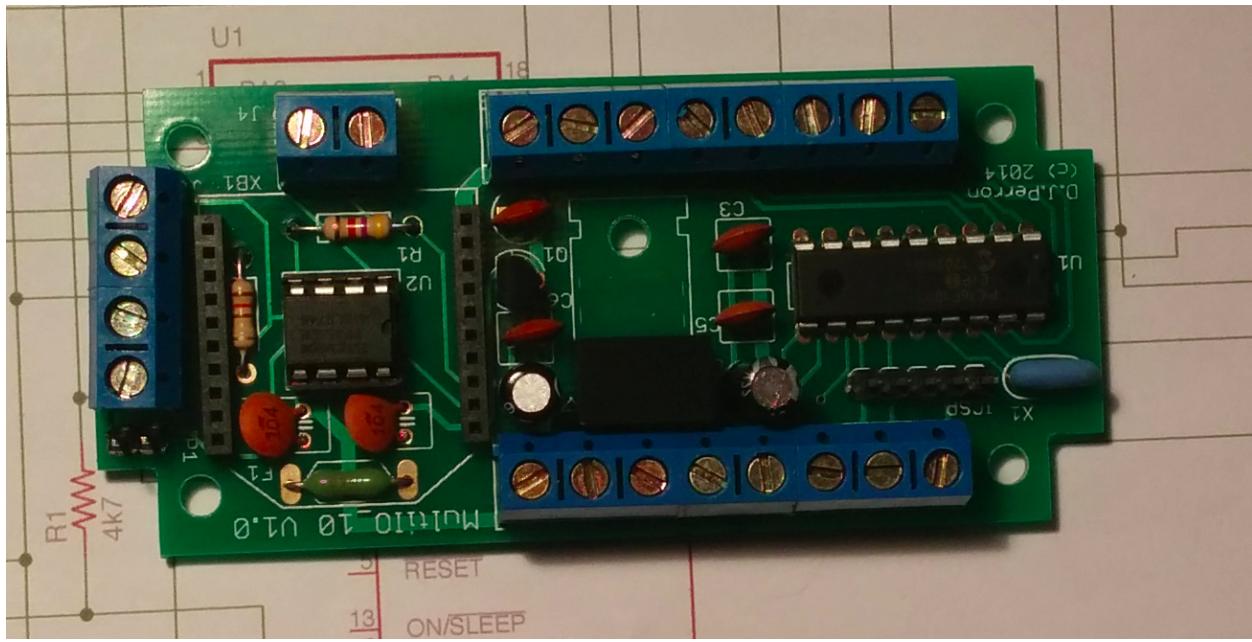


Photo 5. Le PCB Multi IO 10 assemblé

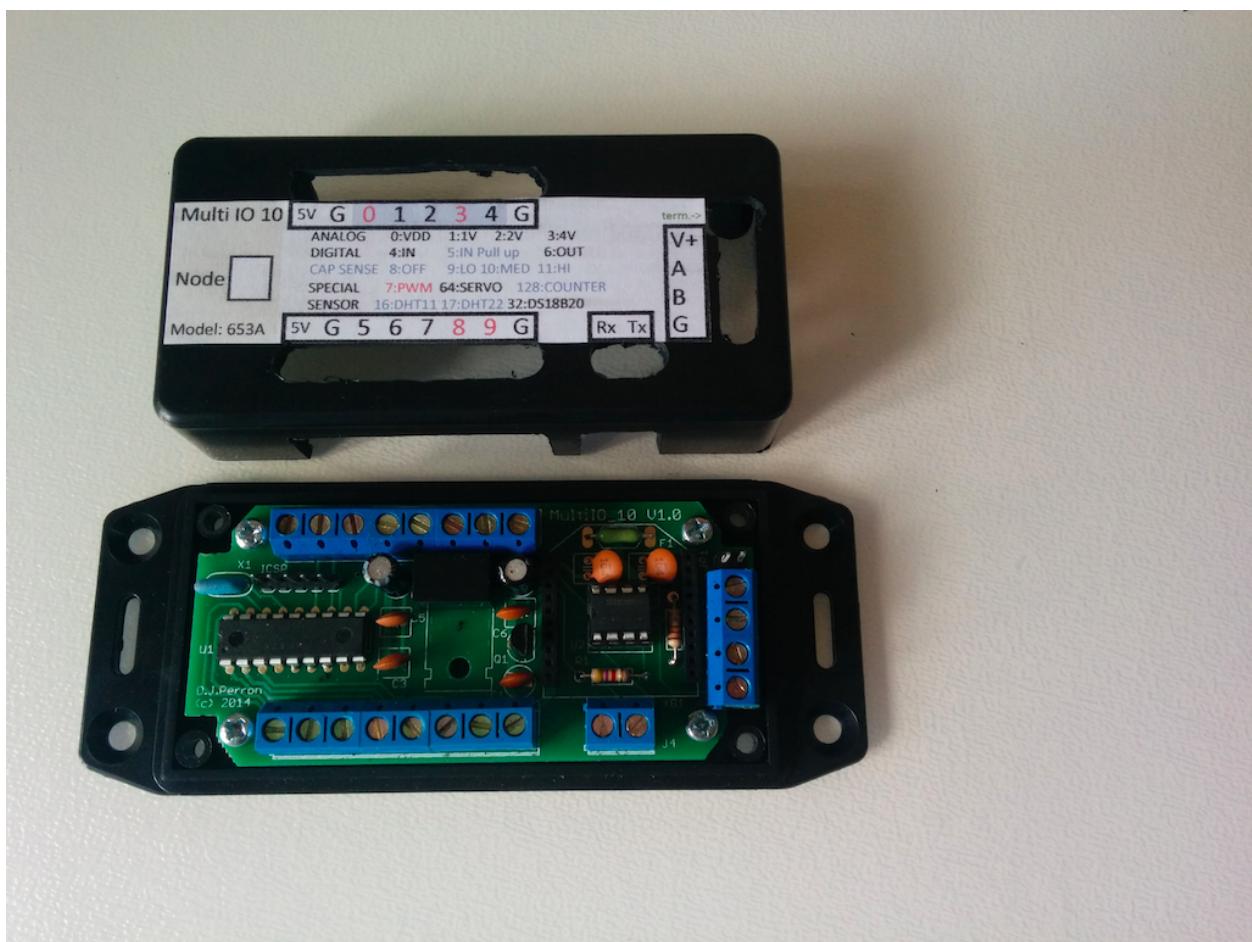


photo 6. Le PCB dans un boîtier.

## Réference

Modbus Protocol Reference Guide

Modicom PI-MBUS-300 Rev. H

AEG SCHNEIDER

AUTOMATION

[http://web.eecs.umich.edu/~modbus/documents/PI\\_MBUS\\_300.pdf](http://web.eecs.umich.edu/~modbus/documents/PI_MBUS_300.pdf)

PIC12F1840

Microchip

8-Pin Flash Microcontrollers with XLP Technology

<http://ww1.microchip.com/downloads/en/DeviceDoc/40001441D.pdf>

LTC485

Linear Technology

Low power RS485 Interface transceiver

<http://cds.linear.com/docs/en/datasheet/485fi.pdf>