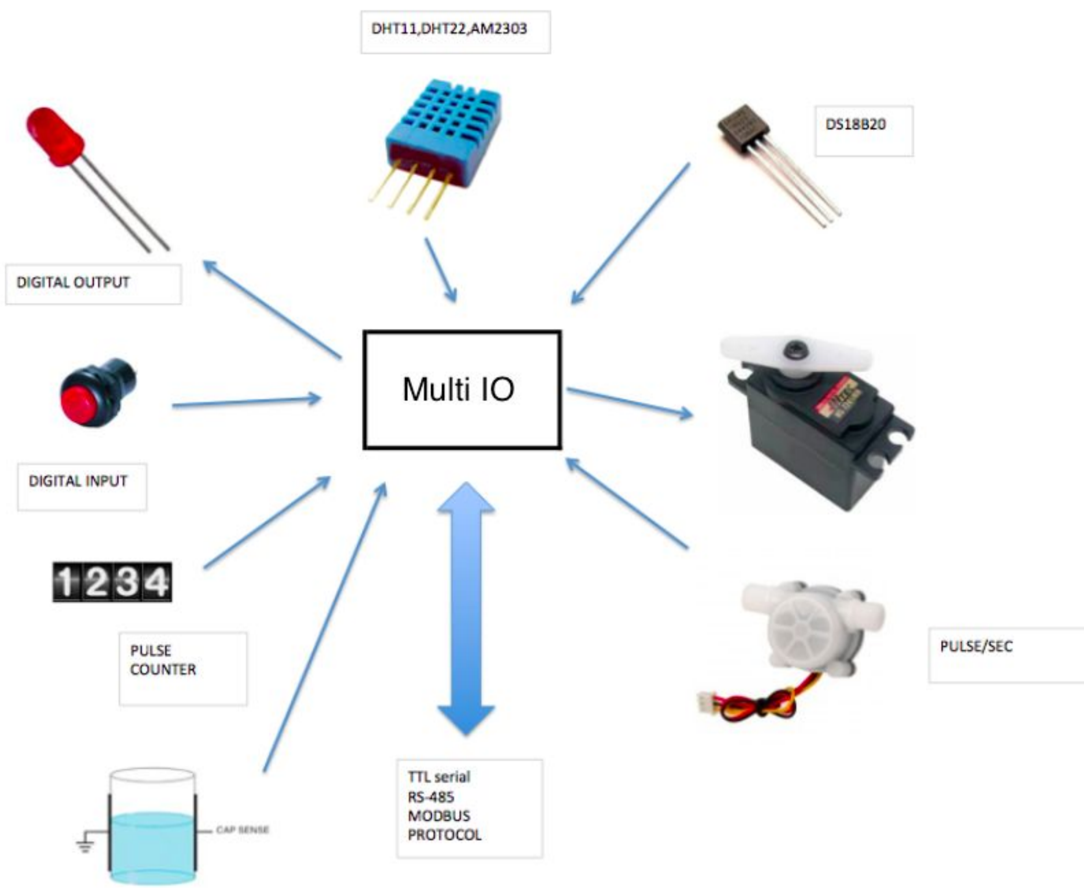
MULTI IO 10

RS-485 IO Interface
with configurable function
using a small Microchip cpu



Daniel Perron
Avril 2014

Preface

Often I need to have a special interface to connect a sensor far away of the computer which will interpret the signal. But instead of re-inventing the wheel all the times, I decide to create a small pcb with build-in screwable terminal with versatile IO configuration. This way I could change, by software, the functionalities of the pins.
I was tired of using an arduino, compile a special code and solder wire between the sensor and the arduino header.

I decide to use the RS-485 multipoint system which allow me to connect more than one module on the same cable and not been limited by short distance cable. The RS-485 cable could be extend up to 1200 meter without repeater.

Since Modbus protocol library was already available for Python and ‘C’, I decide to make the module compatible with Modbus. In theory, up to 247 modules could reside on the same bus but because of driver constraint, 64 modules is more or less the maximum.

The 18 pins DIP PIC16F1827 will be the core cpu with 10 inputs or outputs signal with differents type of configuration possible,

- A/D 10 bits with 4 levels of voltage reference.
- DHT11, DHT22 ou AM2303 sensor interface.
- DS18B20 sensor interface.
- Internal capacitive frequency reader.
- Frequency and pulse counter reader.
- Simple input or output signal.
- 4 PWM output using 10 bits resolution.
- R/C PWM servo control.

Step 1 - Raspberry Pi setup

An USB to Rs-485 interface it the easiest way to communicate with the Raspberry Pi. It is possible to use the internal serial ttl console but it is a little more complex since you need a way to control the direction of the signal.

If you want to use the built-in serial /dev/ttyAMA0 you will need to disable the debug and the serial console.

P.S. If you are using a USB adapter go to step 3.

1 - Disable the serial debug on /dev/ttyAMA0. (/dev/ttyS0 Pi3)

- Type ‘sudo raspi-config’.
- Select interface option.
- Select P6 serial.
- Select No for login shell.
- Select Yes to enable serial.
- Finish
- Reboot

    ~~**pi@raspberrypi ~ $** cd /boot~~

- ~~Make a backup of cmdline.txt just in case.~~
    ~~**pi@raspberrypi /boot $** sudo cp cmdline.txt cmdline.txt.bk.1~~
- ~~Edit the file cmdline.txt and remove everything related to~~
    ~~**pi@raspberrypi /boot $** sudo nano cmdline.txt~~
    ~~Remove **console=ttyAMA0,115200 kgdboc=ttyAMA0,115200**~~
- ~~Overwrite the file by pressing **ctrl-O** and exit **ctrl-X** .~~

~~2 - Edit /etc/inittab and disable /dev/ttyAMA0 login console.~~

- ~~Edit /etc/inittab.~~
    ~~**pi@raspberrypi /boot $** cd~~
    ~~**pi@raspberrypi ~ $** sudo nano /etc/inittab~~
- ~~Find the line with /dev/ttyAMA0 (The last one).~~
- ~~Add the character ‘#’ at the first column of the line.~~
    ~~#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100~~
- ~~Press **ctrl-O** to save and **ctrl-X** to exit.~~
- ~~Reload inittab using the command **init q** or just reboot.~~
    ~~**pi@raspberrypi ~ $** sudo init q~~

3 - Python minimalmodbus installation.

- Update debian

  pi@raspberrypi ~ $ sudo apt-get update

- Install python-pip.

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

- Install minimalmodbus.

  pi@raspberrypi ~ $ sudo pip install -U minimalmodbus
  pi@raspberrypi ~ $ sudo pip-3.2 install -U minimalmodbus

4 - Install d' intelhex. This is needed to program the cpu using the Pi.

sudo python3 -m pip install intelhex

- ~~Get the source code.~~

  ~~**pi@raspberrypi ~ $** wget http://www.bialix.com/intelhex/intelhex-1.5.zip~~

- ~~Unzip it.~~

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

- ~~Install the Python module.~~

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

5 - Install libmodbus

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

6- Now Let’s get the code to burn the PIC18F27 cpu.

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

7- Download PIC_MULTI_10_IO from github.

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

8- Reboot

sudo reboot

Step 2 - Program the cpu

The PIC16F1827 cpu from Microchip has 4K words of program in flash. It is possible to program it directly from the Raspberry PI in LVP mode (Low Voltage programming). You will need to disable SPI and the 1 wire driver. The burnLVPx application will burn the code into the cpu

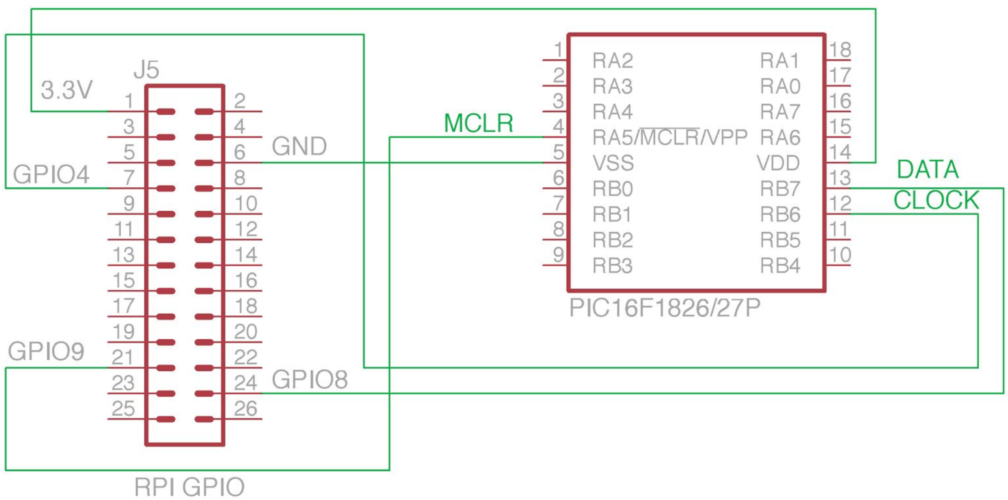
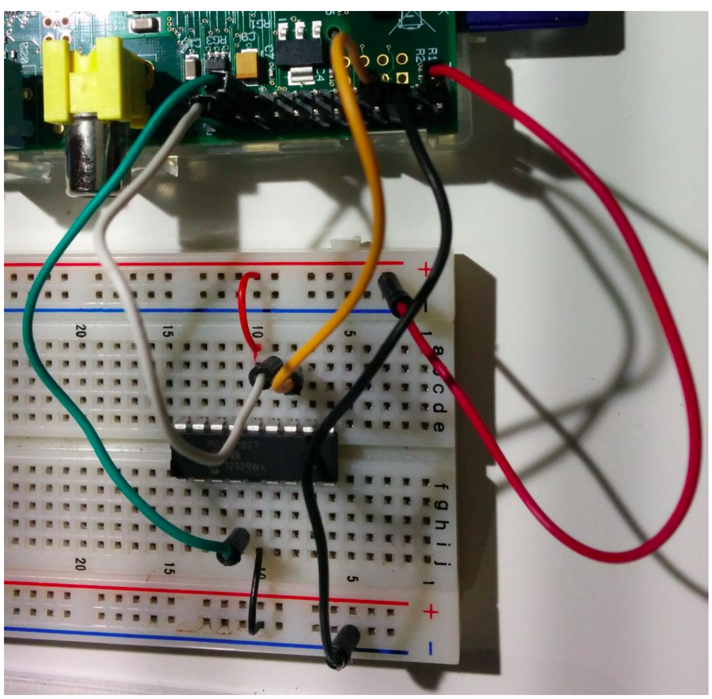


Figure 1. Connection schematic to burn the cpu with the Raspberry Pi.



Picture 1. The cpu ready to be burned.

On the PCB it is possible to use the standard ICSP ,In circuit serial programming, connector. This connector will enable us to use external programmer like the PICKIT from Microchip and it is handy to re-program the cpu.

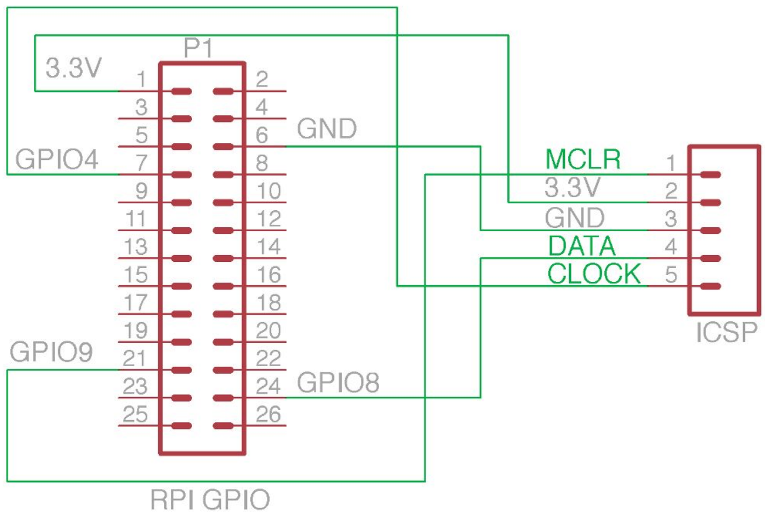
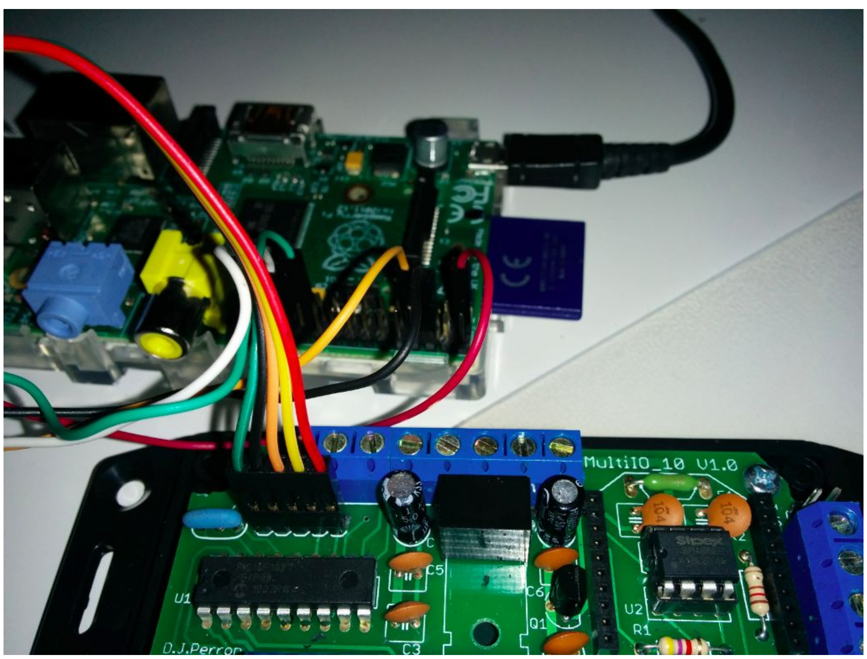


Figure 2. ICSP schematic connection.



Picture 2. ICSP connection with the Raspberry PI.

N.B. You need to disconnect everything on IO3 and IO4 terminal. They are used by the ICSP. Please just use the 3.3V for direct connection with the Raspberry PI.

1 - Use burnLVPx.py to program the cpu.

- Connect the cpu according to Picture 1 or 2.

- Start burnLVPx with the current firmware.

```
pi@raspberrypi ~ $ cd
pi@raspberrypi ~ $ sudo ./burnLVP/burnLVPx.py PIC_MULTI_10_IO/FirmwareV1.04_57600Baud.hex
```

The output should look like this.

```
File " PIC_MULTI_10_IO/FirmwareV1.04_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!
```

- Disconnect and remove the cpu or the ICSP.

- Add a sticker to mark the cpu.

Step 4 - CPU verification

Once the cpu is programmed, it is time to test it with the Raspberry PI using the build-in serial port. A direct connection allow us to check the programmed cpu. The RS-485 interface is not needed .

The current firmware , **FirmwareV1.04_57600Baud.hex,** need an external 8Mhz ceramic resonator.

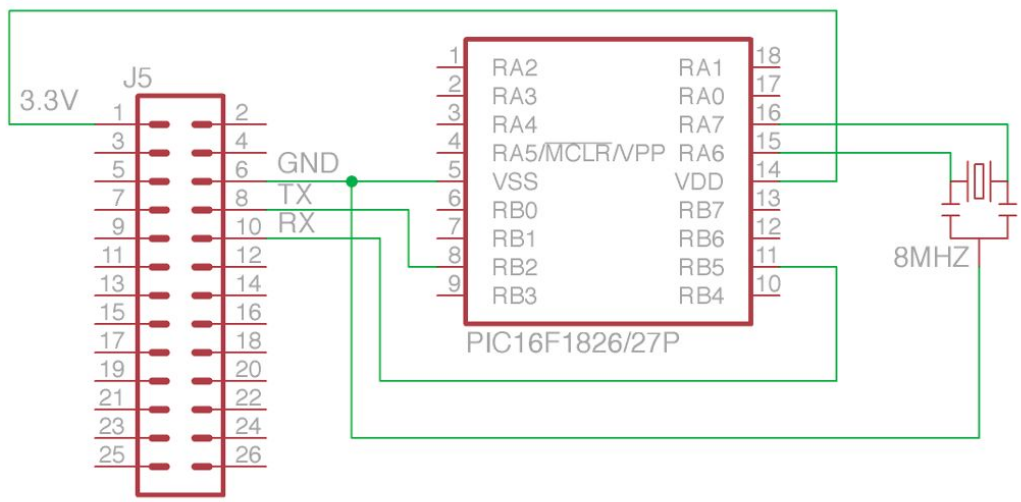
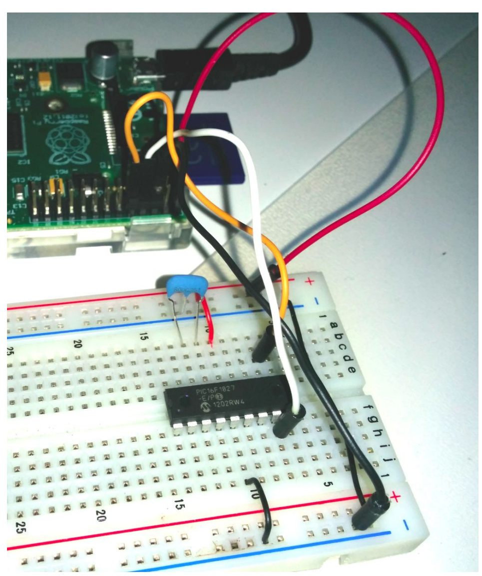


Figure 3. Direct TTL serial connection.



Picture 3. Direct cpu connection.

The “configPIC” will enable to test the cpu. You will need to compile it.

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

P.S. put everthing on 1 line.

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

Let’s run the software with the correct parameters.

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

-d Device port specification.
-b Baud rate.
-t   Waiting time out in microsecond.
For Xbee set timer to -t 200000

This is the application output response ,

```
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
```

Press ‘M’! This will scan the RS-485 bus and it should find one module at the
Press ‘M’. This command will scan the RS-485 bus and it should find one module at the address 127.

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

Now we will change the modbus address to something else. This way we could have more modules. Each module need to have a different modbus address.

Ok let;s change the address 127, the default, to address 1.

Press ‘A’ to select the module 127.

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

It should respond with

```
Selected module :127
```

Press ‘C’ followed by 1 and [Enter].

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

The configPIC application will config each IO pins. The key 0 to 9 will select the IO pin.

ex: To change I/O 0 to be an 2V analog converter (A/D).

1 - Select the specific module . Press ‘A’ follow by the modbus address.
2 - Select the specific IO.   Press the numeric key ‘0’...’1’.
   The application will display a list of all mode possible the specific IO.
   Press 0 and 4 for 2V A/D.

```
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
```

The key ‘L’ will display all the I/O configuration.

```
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
```

I/O specification and modbus

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

Figure 4. Configuration table

The modbus address and the I/Os configuration are stored in flash memory of the cpu. They are permanently store.

This is the configuration constant list in Python.

```
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
```

A/D MODE

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

A 10 bits value will be returned.

The formula is,

```
Voltage read =  A/D value * Reference Voltage / 1023;
```

Python

```
readSensor(Pin)       => this function came from PicModule.py
```

and it will called

```
  module.read_registers(Pin * 16 , 1, 4)[0]
```

Read register[16 * Pin] with modbus function 4

Language C

```
uint16_t MB_Register[3];
if(modbus_read_input_registers(mb,Pin*16,1,MB_Register)>=0)
  {
    // The  MB_Register[0] will hold the 10 bits conversion
  }
```

ex: Read Temperature on IO0 module 1.

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

#ioconfig are set once and store into the eerom
# 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 eerom writing

AnalogValue= remote1.readSensor(0)
print("TMP35 : {:5.1f}°C".format(AnalogValue/10.0))
```

On the A/D conversion system, two special registers could be read to get power voltage and build-in diode temperature sensor.

Modbus register 0x1000   => Return A/D conversion of 2.048V using VDD has Vref.

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)
   {
      // register MB_Register[0] hold 10 bits A/D.
   }
```

ex: Read the CPU Voltage.

```
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))
```

The same method could be used to read the thermal diode.

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)
   {
      // register MB_Register[0] hold 10 bits A/D.
   }
```

DIGITAL INPUT MODE

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

The IOCONFIG_INPUT_PULLUP is only available for IO0,IO1,IO2,IO3 et IO4. One small pull-up current will hold the input to VDD.

Each I/O in digital mode are define has 0 or 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. Digital IO could be read or set using an 16 multiplier offset.

ex: To read IO7

```
#Let's assume that the configuration is already set to digital input.

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

print("IO0: {}",remote1.readIO(0))
```

DIGITAL OUTPUT MODE

```
IOCONFIG_OUTPUT         =6
```

This mode will set IO to digital output signal.

Python

writeIO(Pin, valeur)

module.write_bit(Pin,valeur)

Language C

```
if(modbus_write_bit(mb,Pin,Value)>0)
 {
    … it is done!
 }
```

ex: Create three pulse on IO1 module 2 with an USB to RS-485 adapter.

```
import time
import PicModule
remote2 = PicModule.PicMbus(2,Baud=57600,Device='/dev/ttyUSB0')
#Let's assume that the configuration is already set.

for pulse in range(3):
   remote2.writeIO(1,1)
   time.sleep(0.1)
   remote2.writeIO(1,0)
   time.sleep(0.1)
```

CAPACITIVE MODE

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

A running oscillator will clock at the determine frequency will depends on capacitive load. This mode will read count. Four mode of power are available with one with no generator.
The format is 32 bits (4 registers) and the time base is 0.1 sec.

Python

readSensor(Pin)

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

ex: Read Frequency on IO0 et IO1

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

#assume that  IOCONFIG1 et IOCONFIG0  are in mode
IOCONFIG_CAP_SENSE_HIGH

print("Cap sense IO0 : {}".format(remote1.ReadSensor(0))
print("Cap sense IO1 : {}".format(remote1.ReadSensor(1))
```

Dielectric constant differs from the type of material . With this mode it is possible to calculate level from a container. You will need to get the count readout any multiple position and compensate for temperature .

DHT SENSOR MODE

```
IOCONFIG_DHT11                  =16
IOCONFIG_DHT22                  =17    (Also AM2303 type)
```

Read DHT11 or DHT22 temperature sensor.

Two registers will return Temperature and humidity.

Python

```
readDHT(Pin)
       return  None => No Sensor reading.
       return  [humidity, temperature]

module.read_register(Pin * 16,3,4)   #this is a generic function
       return registers
             [0] 0xFFFF => No sensor  1=> sensor value OK
             [1] Humidity raw value
             [2] Temperature raw value
```

Language C

```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: Read DHT22 sur IO 0 Module 1

```
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. Care have to be taken when you power up the DHTXX sensor. It won’t work if it is not let’s Idle high on power up. Better to config the sensor first, connect the sensor and power it up.

Also to use to much counter and R/C servo. This will increase the interrupt lag and made the sensor unreadable.

THE DS18B20 MODE

```
IOCONFIG_DS18B20                  =32
```

Read DS18B20 temperature sensor.

N.B. IO5, IO6, IO7, IO8 and IO9 will need a pull-up resistor (~4k7)

Python

readDS18B20(Pin)

return   None => No sensor
retourn The temperature

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]);
         printf("Temp: %5.1f Celsius",Temperature);
       }
       else printf("Error");
     }
    else
     printf("Unable to read  Sensor");
    printf("\n");
}
```

ex: Read DS18B20 sur IO0 Module 1

```
import time
import PicModule
remote1 = PicModule.PicMbus(1)
result = remote1.readDS18B20(0)
if result is None:
  print(“Error ON DS18B20  Sensor! Bad Config or sensor not connected”)
else:
  print(“Temperature:{6.2f}°C”.format(result))
```

The R/C SERVO MODE

```
IOCONFIG_SERVO            =64
```

Control R/C servo module.

Set the register in micro-second step.

To turn OFF servo, siply set te register to 0.

The default range for R/C servo are 500 to 2500µs

It is possible to play between 1 to 20000 but the behavior could be really bad.

Python

```
RCServo(Pin,value)
```

Language C

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

ex: Set ALL R/C servo from module in center position (1000us).

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

THE COUNTER MODE

```
IOCONFIG_COUNTER                    =128
```

Count digital pulse in 32 bits resolution. Also return a Pulse/sec frequency couter (16 bits).

Python

readSensor(Pin)

[0]= MSB 16 bits Unsigned int counter.
[1]= LSB 16 bits Unsigned int counter.
[2]= Pulse per second counter. (The frequency).

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  Frequence;
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]);
   Frequence = MB_Register[2];
 }
```

It is possible to reset the counter.

python

```
resetCounter(Pin)

module.write_register(Pin,0,0,6)   #this is the generic function called by resetCounter()
```

Language C

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

A webpage using 3 modules of 8 relays.

This is a small project using a webpage powered by webiopi to control 24 relays using three modules.

Install webiopi

```
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
```

Set the Raspberry-pi to enable webiopi on power up. Raspberry Pi.

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

The first time we will need to reboot or start webiopi manually.

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

Let's change the script to include our own python script to access the modbus module and let's set the system to set WebRelais.html has the default webpage.

```
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 is in the github PICMulti_10_IO. you just need to copy it over.

```
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
```

And let's transfert de webpage to /usr/share/webiopi/htdocs

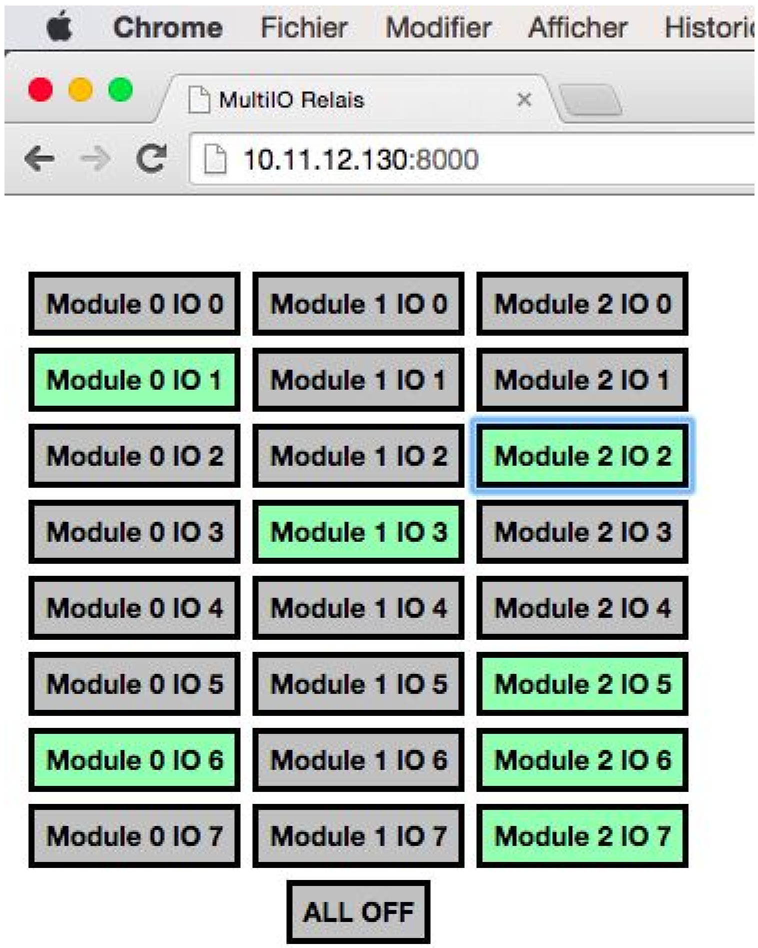
```
pi@raspberrypi ~ $ cd /usr/share/webiopi/htdocs
pi@raspberrypi ~ $ sudo cp /home/pi/PIC_MULTI_10_IO/WebRelais.html .
```

Restart webiopi

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

The webpage should be available at the Raspberry PI IP on PORT 800.

You could change or remove the password by editing or deleting /etc/webiopi/passwd.



Ok IO0 to IO7 are connected to relay. We could use IO8 to check the temperature with a DS18B20. We will need to add a 4k7 pull-up resistor.

From a previous project, I do have code to display temperature using RDTOOL and HighChart.

InstallHighChart and 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
```

Do a test to check the install

http://raspberrypi.local/charts/index.htm   (Not “html” but “htm”).

raspberrypi.local is the Raspberry Pi. You could use avachi to set a local name

install libnss-mdns

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

This way you will have access to the Raspberry PI using it’s identificatio name + “.local”.
P.S. It won’t work with adroid device.

If you want to change the name of your raspberry PI to WebPi for example

sudo hostname WebPi
sudo nano /etc/hosts   **and change raspberrypi to WebPi**
sudo nano /etc/hostname   **and change raspberrypi to WebPi**

and “reboot”

Now that HightCharts et rrdtool are installed , ww will modify WebRelais.py and add 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 .
```

To prevent premature wear of the SD card, we will create a ramdisk to handle temporary file create by rrdtool.

in /etc/fstab, add the line in yellow,

```
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
```

The ramdisk will be link at /usr/share/webiopi/htdocs/temperature.

Restart the RPi to check if the ramdisk will be mounted properly.

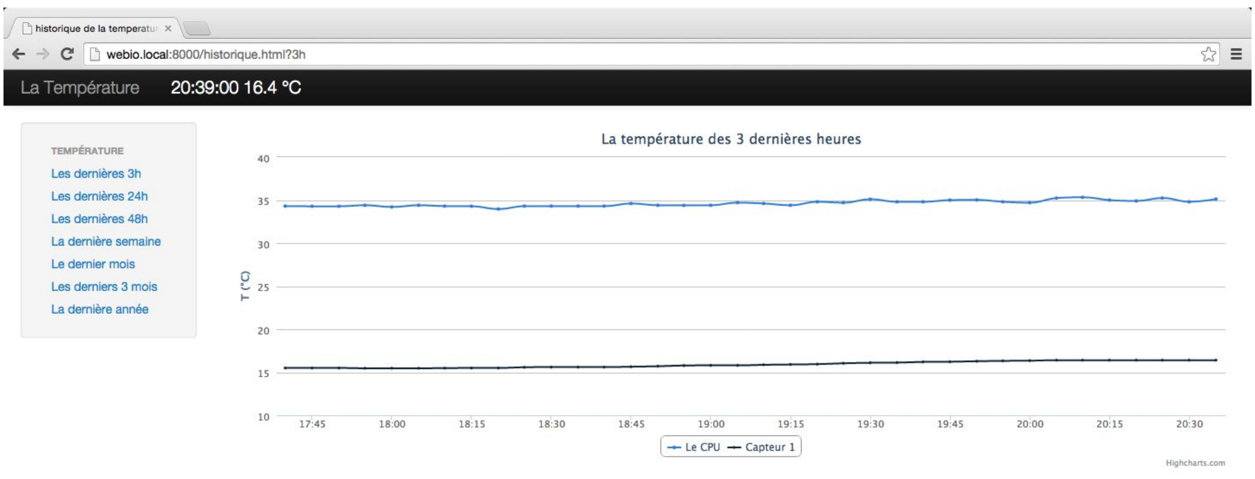
A already made script will create the rrdtool database. But first let's create /home/pi/data

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

And restart webiopi

pi@raspberrypi ~ $ sudo service webiopi start

And Check the webpage   http:://RaspberryPi.local:8000/historique.html



APPENDIX

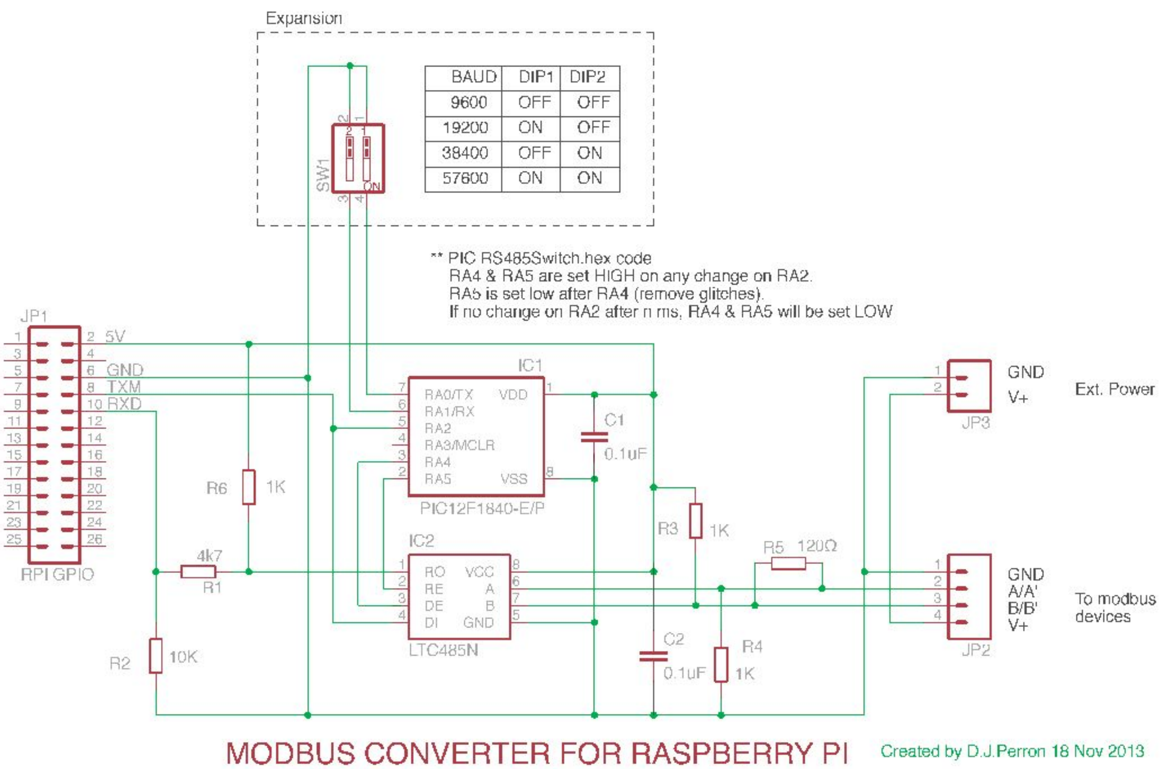


Figure 6- RS-485 I.C. setup with Raspberry Pi

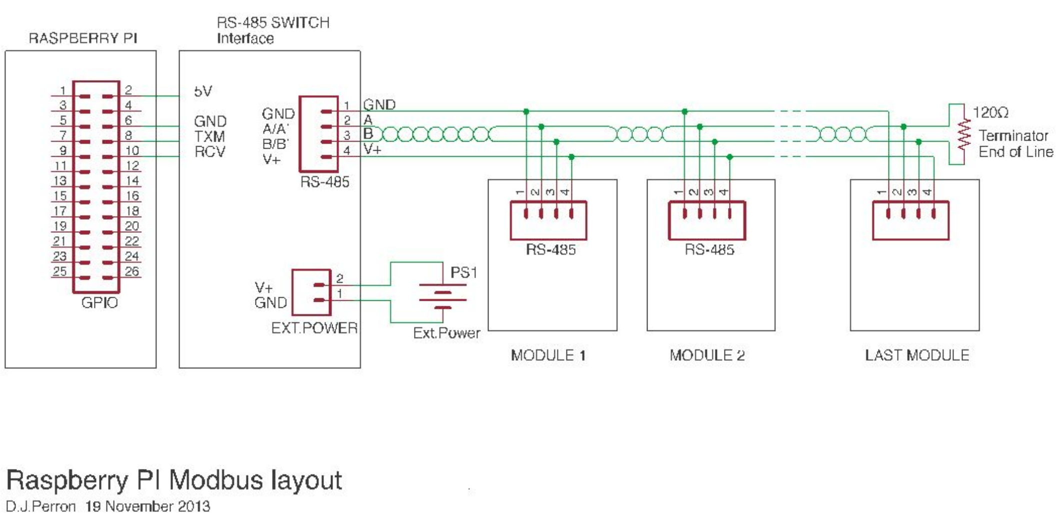


Figure 7 - Three modules in modbus example.

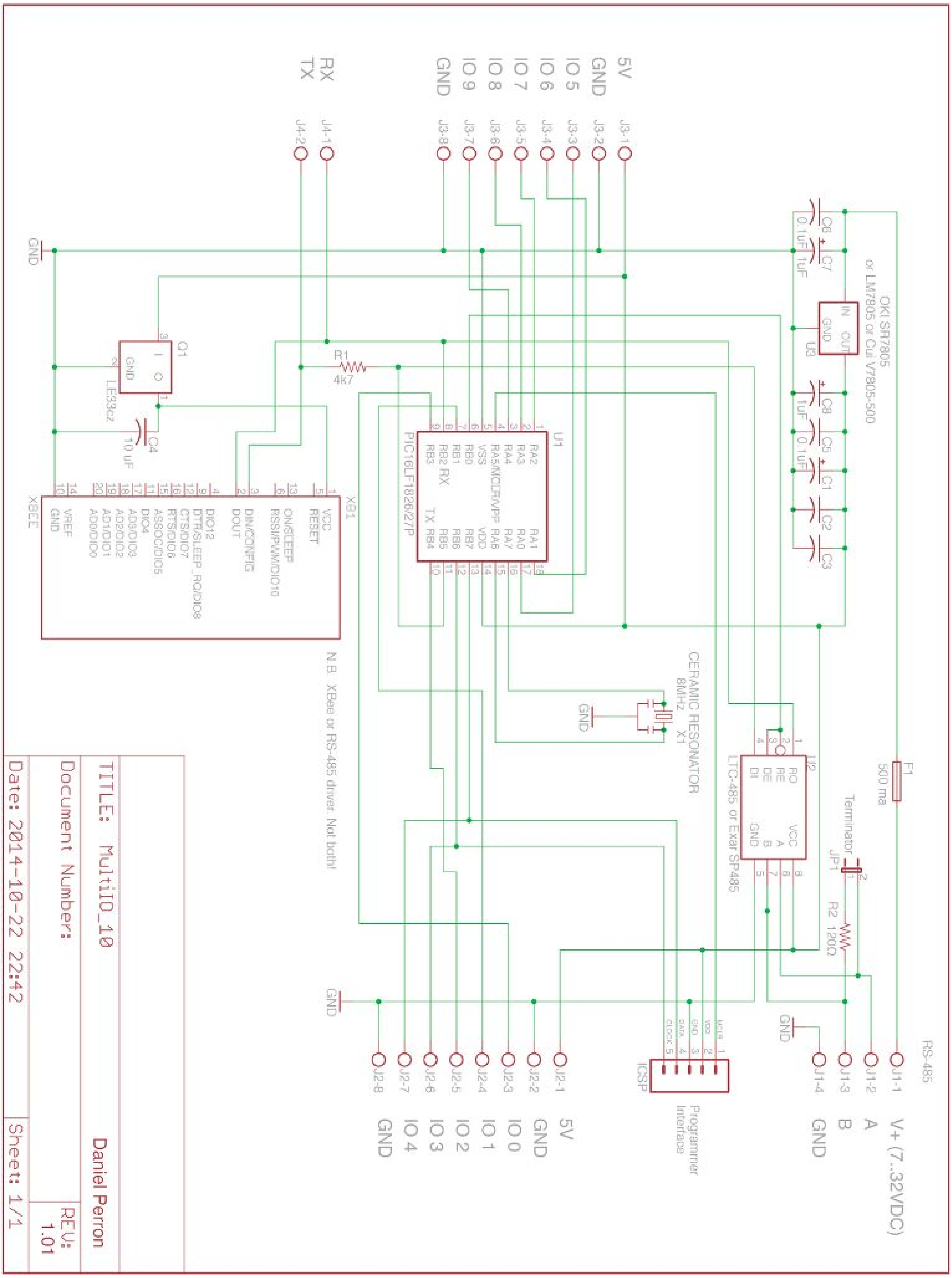


Figure 8. PCB schematic multi IO 10

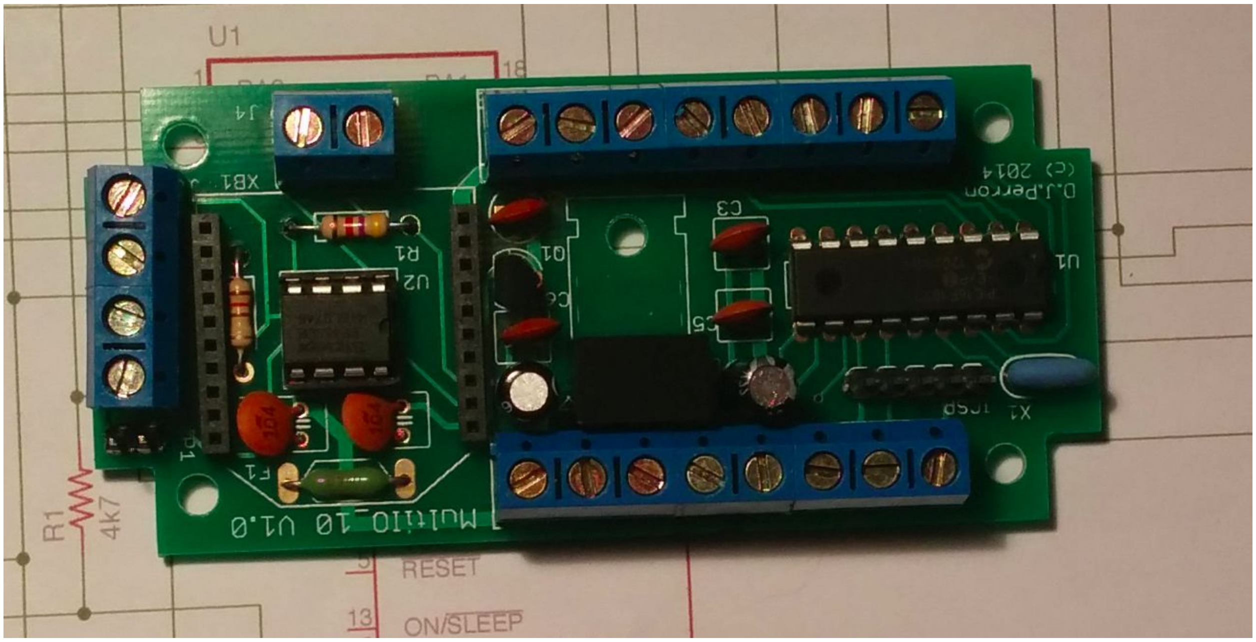


Photo 5. The PCB Multi IO 10 assembly

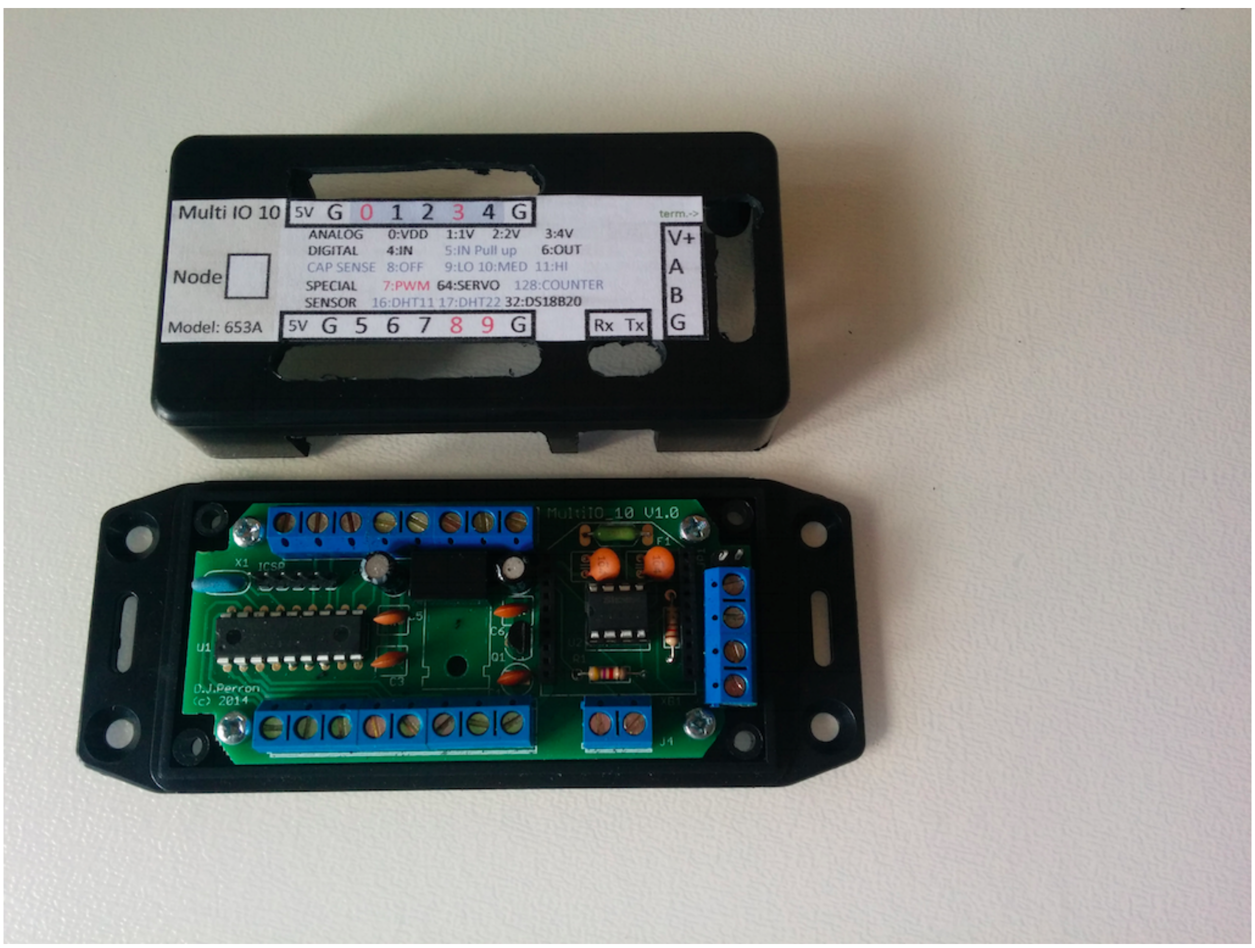


photo 6. The PCB in the Hammond 1591ASFLBK box.

Reference

Modbus Protocol Reference Guide
Modicom PI-MBUS-300 Rev. H
AEG SCHNEIDER
AUTOMATION
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