



Raspberry Pi : utiliser un LCD 4×20

Aujourd'hui on va rajouter un petit périphérique très simple mais fort pratique : un petit LCD low-cost 4 lignes / 20 caractères par ligne, prévu à l'origine pour Arduino.



Comme on peut le voir sur la photo, cet écran intègre un module I²C en 5V, ce qui nous permet de l'utiliser avec le Raspberry avec seulement 2 fils.

[Vous pouvez l'acheter ici en nous faisant gagner quelques centimes...](#)

Au niveau des branchements, c'est exactement la même chose que dans notre [tuto sur l'horloge RTC](#). Les deux modules peuvent même être chaînés, le bus I²C c'est fait pour ça.

Attention par contre aux niveaux logiques :

Notre Raspberry Pi fonctionne sur une logique 3.3V. Le LCD quant à lui fonctionne sur une logique 5V. Si les résistances de pull-up présentes sur les pin I²C du Pi suffisent pour l'horloge RTC, elle ne suffiront certainement pas pour l'écran.

Nous allons donc rajouter un convertisseur logique, qui sera d'ailleurs fort utile pour brancher tout un tas d'autre modules I²C, sans plus jamais se poser la question des niveaux logiques (pour le bus I²C en tous cas)... mais sinon le tuto ressemblera for-

tement à celui sur l'horloge RTC...

1- Nécessaire

- un écran 4×20 I²C ([de ce genre](#))
- un convertisseur logique 3.3v-5v à 70 cents ([exemple](#))

2- Branchements

On va partir à chaque fois du convertisseur, ce sera le plus simple pour expliquer.

coté 5V :

connectez AVCC au +5v du Pi

connectez AVCC au +5v de l'écran

connectez AGND au GND du Pi

connectez AGND au GND de l'écran

connectez ASCL au SCL de l'écran

connectez ASDA au SDA de l'écran

coté 3.3V :

connectez BVCC au +3.3v du Pi

connectez BGND au GND du Pi

connectez BSCL au SCL du Pi

connectez BSDA au SDA du Pi

Grâce à ce montage, il est possible d'ajouter autant de composants/modules I²C qu'on le désire (enfin presque), que ce soit en 3.3V ou en 5V : les modules 5V se connectent du coté A du convertisseur, les modules 3.3V (dont le Pi) se connectent du coté B.

2- Activation et configuration du module logiciel I²C

Ici, je vais surtout me répéter... On a déjà tout préparé dans le tuto sur l'horloge RTC.

On ajoute les modules au démarrage :

```
sudo nano /etc/modules
```

Ajouter ces 2 lignes :

```
i2c-bcm2708  
i2c-dev
```

On redémarre :

```
sudo reboot
```

On installe les outils nécessaires :

```
sudo apt-get install python-smbus i2c-tools
```

On supprime les modules I²C de la blacklist (si c'est le cas) :

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

la ligne

```
blacklist i2c-bcm2708
```

devient

```
#blacklist i2c-bcm2708
```

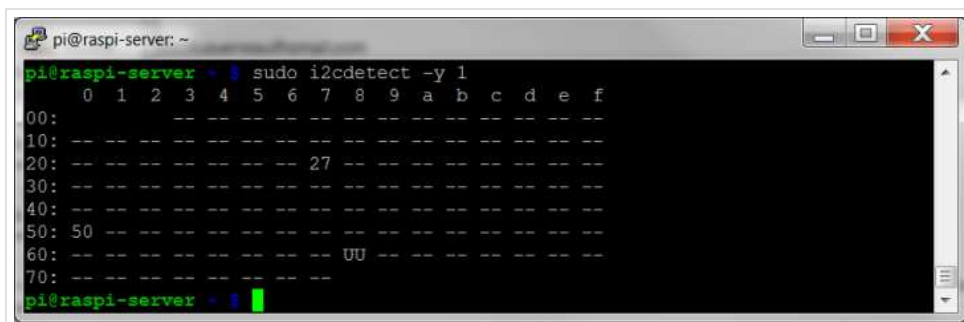
On redémarre, et le Raspi devrait être prêt à utiliser les périphériques I²C.

3- Test de l'interface I²C

On va utiliser la commande `i2cdetect`, qui permet de voir la liste des périphériques I²C reliés au Pi. La commande est différente si vous utilisez un Raspberry Pi Rev 1 ou Rev 2 (l'adresse du bus I²C est différente), donc choisissez la bonne :

```
sudo i2cdetect -y 0 (pour la Rev 1)
```

```
sudo i2cdetect -y 1 (pour la Rev 2)
```



```
pi@raspi-server: ~  
pi@raspi-server ~$ sudo i2cdetect -y 1  
0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- UU -- -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
pi@raspi-server ~$
```

Sur cette capture, on peut voir que j'ai 3 composants I²C :

- l'écran à l'adresse #27
- la puce AT24C32 de l'horloge RTC à l'adresse #50
- la puce DS1307 de l'horloge RTC à l'adresse #68 (elle est notée « UU » car elle est en cours d'utilisation)

Il est possible que votre écran ait une adresse différente, j'en ai déjà vu en 24 et 28. Notez cette adresse si elle est différente de 27, on en aura besoin plus tard pour configurer le driver.

4- Installation du driver

Nous allons utiliser un petit pilote en python pour communiquer avec l'écran. Vous l'aurez compris, si on veut communiquer avec l'écran, ce sera en python 😊

Dans un dossier, nous allons donc créer 2 fichiers :

```
sudo nano i2c_lib.py
```

```
import smbus  
from time import *  
  
class i2c_device:  
    def __init__(self, addr, port=1):  
        self.addr = addr  
        self.bus = smbus.SMBus(port)  
  
    # Write a single command  
    def write_cmd(self, cmd):  
        self.bus.write_byte(self.addr, cmd)  
        sleep(0.0001)
```

```
# Write a command and argument
def write_cmd_arg(self, cmd, data):
    self.bus.write_byte_data(self.addr, cmd, data)
    sleep(0.0001)

# Write a block of data
def write_block_data(self, cmd, data):
    self.bus.write_block_data(self.addr, cmd, data)
    sleep(0.0001)

# Read a single byte
def read(self):
    return self.bus.read_byte(self.addr)

# Read
def read_data(self, cmd):
    return self.bus.read_byte_data(self.addr, cmd)

# Read a block of data
def read_block_data(self, cmd):
    return self.bus.read_block_data(self.addr, cmd)
```

```
sudo nano lcddriver.py
```

```
import i2c_lib
from time import *

# LCD Address
ADDRESS = 0x27

# commands
LCD_CLEARDISPLAY = 0x01
LCD_RETURNHOME = 0x02
LCD_ENTRYMODESET = 0x04
LCD_DISPLAYCONTROL = 0x08
LCD_CURSORSHIFT = 0x10
LCD_FUNCTIONSET = 0x20
LCD_SETCGRAMADDR = 0x40
LCD_SETDDRAMADDR = 0x80

# flags for display entry mode
LCD_ENTRYRIGHT = 0x00
LCD_ENTRYLEFT = 0x02
LCD_ENTRYSHIFTINCREMENT = 0x01
LCD_ENTRYSHIFTDECREMENT = 0x00
```

```
# flags for display on/off control
LCD_DISPLAYON = 0x04
LCD_DISPLAYOFF = 0x00
LCD_CURSORON = 0x02
LCD_CURSOROFF = 0x00
LCD_BLINKON = 0x01
LCD_BLINKOFF = 0x00

# flags for display/cursor shift
LCD_DISPLAYMOVE = 0x08
LCD_CURSORMOVE = 0x00
LCD_MOVERIGHT = 0x04
LCD_MOVELEFT = 0x00

# flags for function set
LCD_8BITMODE = 0x10
LCD_4BITMODE = 0x00
LCD_2LINE = 0x08
LCD_1LINE = 0x00
LCD_5x10DOTS = 0x04
LCD_5x8DOTS = 0x00

# flags for backlight control
LCD_BACKLIGHT = 0x08
LCD_NOBACKLIGHT = 0x00

En = 0b00000100 # Enable bit
Rw = 0b00000010 # Read/Write bit
Rs = 0b00000001 # Register select bit

class lcd:
    #initializes objects and lcd
    def __init__(self):
        self.lcd_device = i2c_lib.i2c_device(ADDRESS)

        self.lcd_write(0x03)
        self.lcd_write(0x03)
        self.lcd_write(0x03)
        self.lcd_write(0x02)

        self.lcd_write(LCD_FUNCTIONSET | LCD_2LINE | LCD_5x8DOTS | LCD_4
        self.lcd_write(LCD_DISPLAYCONTROL | LCD_DISPLAYON)
        self.lcd_write(LCD_CLEARDISPLAY)
        self.lcd_write(LCD_ENTRYMODESET | LCD_ENTRYLEFT)
        sleep(0.2)

    # clocks EN to latch command
    def lcd_strobe(self, data):
        self.lcd_device.write_cmd(data | En | LCD_BACKLIGHT)
        sleep(.0005)
        self.lcd_device.write_cmd(((data & ~En) | LCD_BACKLIGHT))
        sleep(.0001)
```

```
def lcd_write_four_bits(self, data):
    self.lcd_device.write_cmd(data | LCD_BACKLIGHT)
    self.lcd_strobe(data)

# write a command to lcd
def lcd_write(self, cmd, mode=0):
    self.lcd_write_four_bits(mode | (cmd & 0xF0))
    self.lcd_write_four_bits(mode | ((cmd << 4) & 0xF0))

# put string function
def lcd_display_string(self, string, line):
    if line == 1:
        self.lcd_write(0x80)
    if line == 2:
        self.lcd_write(0xC0)
    if line == 3:
        self.lcd_write(0x94)
    if line == 4:
        self.lcd_write(0xD4)

    for char in string:
        self.lcd_write(ord(char), Rs)

# clear lcd and set to home
def lcd_clear(self):
    self.lcd_write(LCD_CLEARDISPLAY)
    self.lcd_write(LCD_RETURNHOME)
```

Dans ce dernier fichier, vous pouvez ajuster l'adresse de votre écran si ce n'est pas 27 :

```
# LCD Address
ADDRESS = 0x27
```

5- Utilisation de l'écran

C'est le plus facile...

Quand vous voulez utiliser l'écran dans un script, vous n'avez qu'à inclure les 2 fichiers du pilote dans le dossier de votre script. Voici le minimum à utiliser pour exploiter l'écran :

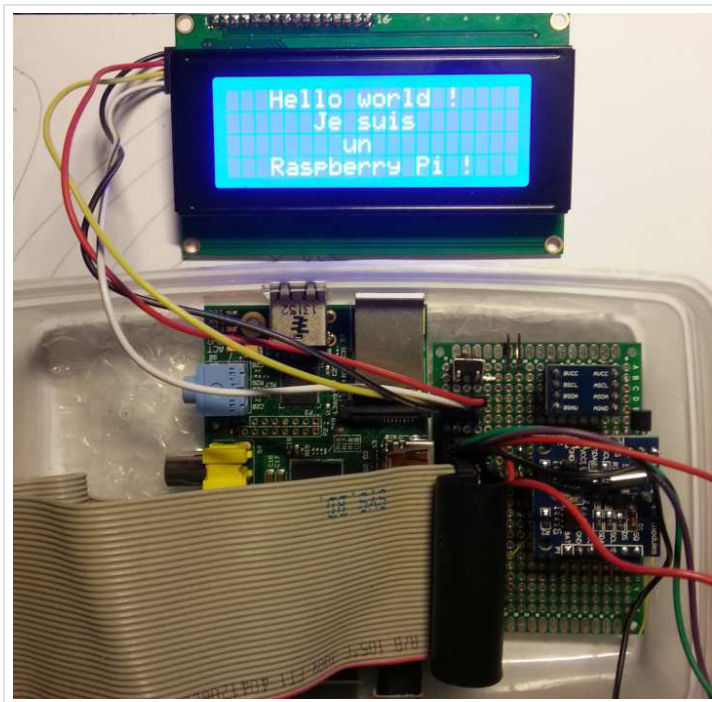
```
# on importe le pilote
import lcddriver
```

```
from time import *

# on initialise le lcd
lcd = lcdriver.lcd()

# on reinitialise le lcd
lcd.lcd_clear()



# on affiche des caracteres sur chaque ligne
lcd.lcd_display_string("  Hello world !", 1)
lcd.lcd_display_string("      Je suis", 2)
lcd.lcd_display_string("      un", 3)
lcd.lcd_display_string("  Raspberry Pi !", 4)
```




Vous noterez que je vous annonce seulement maintenant que ces 3 fichiers sont sur mon github :

https://github.com/CaptainStouf/raspberry_lcd4x20_I2C 😊

Voila, votre Raspberry Pi peut désormais afficher tout un tas de trucs inutiles...

 Tweeter  Partager 5  submit  E-mail

 Imprimer  Plus

Posted by: Captain Stouf // Hardware, Les guides, Linux, Raspberry Pi, Réseau domotique, Software
// 2004, 20x4, IIC/I2C/TWI 2004, I²C, lcd, raspberry pi // mars 14, 2014 [<http://hardware-libre.fr/2014/03/raspberry-pi-utiliser-un-lcd-4x20/>]

3 thoughts on “Raspberry Pi : utiliser un LCD 4×20”

**Kroms***25/03/2014 at 10 h 12 min*

Salutation,

Est il possible d'écrire « Hello world ! » en lui imposant une coordonnée X et Y pour le positionner ou l'on veut sur le LCD ?

Bonne journée

**mackoomba***27/03/2014 at 10 h 33 min*

merci pour ce tuto !!!

@kroms, oui il est possible d'afficher exactement ce qu'on veut, il te suffit de modifier les lignes: (dans le fichier lcd.py)

on affiche des caracteres sur chaque ligne

`lcd lcd_display_string(» Hello world ! », 1)``lcd lcd_display_string(» Je suis », 2)``lcd lcd_display_string(» un », 3)``lcd lcd_display_string(» Raspberry Pi ! », 4)`

les numéros correspondent aux lignes du LCD, ensuite

t'ajoute ou tu retire des espaces

si tu veux automatiser, il faudra fouiller le langage python...

de mon coté le LCD affiche

ligne 1: heure/ ligne 2: date / ligne 3: adresse ip / ligne 4:

temperature ARM