**Top !** simius.be/modules-test/afficheur-sur-bus-i2c

Contenu

[http://ares.asso-ensea.net/robotics/realisation\_d\_un\_afficheur\_lcd\_i2c/ 3](#_Toc404248771)

[Réalisation d’un afficheur LCD I2C 3](#_Toc404248772)

[Introduction 3](#_Toc404248773)

[Fonctionnement du PCF8574P 3](#_Toc404248774)

[Fonctionnement d’un afficheur LCD 3](#_Toc404248775)

[Description des broches 4](#_Toc404248776)

[Le jeu d’instruction 5](#_Toc404248777)

[Affichage 7](#_Toc404248778)

[Mode 4 bits 7](#_Toc404248779)

[Schéma 8](#_Toc404248780)

[Le typon 8](#_Toc404248781)

[Programme 9](#_Toc404248782)

[Photos 12](#_Toc404248783)

[Avertissement 12](#_Toc404248784)

[http://www.simius.be/modules-test/afficheur-sur-bus-i2c.html 13](#_Toc404248785)

[Afficheur sur bus I2c 13](#_Toc404248786)

[Schéma interne du PCF8574 13](#_Toc404248787)

[Adressage 14](#_Toc404248788)

[http://www.tobozzo.free.fr/html/pages-electron/saa1064.htm 18](#_Toc404248789)

[http://www.pobot.org/Piloter-des-afficheurs-7-segments.html?lang=fr 19](#_Toc404248790)

[http://skyduino.wordpress.com/2011/11/10/tutoriel-arduino-communication-par-spi-hard-et-soft-et-par-i2c/ 20](#_Toc404248791)

[Communication par SPI (hard et soft) et par I2C 20](#_Toc404248792)

[http://arduino103.blogspot.fr/2013/03/connecter-des-arduino-ensembles-un.html 25](#_Toc404248793)

[Connecter des Arduino ensembles - Un tutoriel I2C pour Arduino 25](#_Toc404248794)

[http://www.mon-club-elec.fr/pmwiki\_reference\_arduino/pmwiki.php?n=Main.LibrairieWire 28](#_Toc404248795)

[Description 28](#_Toc404248796)

[Connexions 28](#_Toc404248797)

[Les fonctions de la librairie Wire 28](#_Toc404248798)

[Fonctions d'initialisation 28](#_Toc404248799)

[Fonctions "mode maître" 28](#_Toc404248800)

[Fonctions "mode esclave" 29](#_Toc404248801)

[Fonctions obsolètes 29](#_Toc404248802)

[Principe d'utilisation 29](#_Toc404248803)

[Note 29](#_Toc404248804)

[Exemples d'utilisation : 29](#_Toc404248805)

[Sur le site Arduino anglais : 29](#_Toc404248806)

[Sur le site www-mon-club-elec.fr : 30](#_Toc404248807)

[http://www.pihomeserver.fr/2013/08/13/raspberry-pi-home-server-arduino-lier-les-deux-via-bus-i2c/ 31](#_Toc404248808)

[Raspberry Pi – Arduino – Lier les deux via le bus I2C 31](#_Toc404248809)

[Définir l’Arduino comme esclave 33](#_Toc404248810)

[Activer la gestion I2C sur le Raspberry Pi 34](#_Toc404248811)

[Communiquer avec l’Arduino 35](#_Toc404248812)

[http://www.pobot.org/Tutoriel-complet-sur-I2C.html?lang=fr 39](#_Toc404248813)

[Tutoriel complet sur I2C 39](#_Toc404248814)

[Le matériel 39](#_Toc404248815)

[Code Arduino 39](#_Toc404248816)

[Commandes Linux / Raspberry Pi 40](#_Toc404248817)

[Astuce pour PC Linux 41](#_Toc404248818)

# <http://ares.asso-ensea.net/robotics/realisation_d_un_afficheur_lcd_i2c/>

[ARES ENSEA](http://ares.asso-ensea.net/robotics)

###### Association Enséarque des passionnés d'électronique, de mécatronique, de télécommunication, d'informatique...

## Réalisation d’un afficheur LCD I2C

By [*Matthieu Patteyn*](http://ares.asso-ensea.net/robotics/author/mattpatt/) On 15 janvier 2014 · [*6 Comments*](http://ares.asso-ensea.net/robotics/realisation_d_un_afficheur_lcd_i2c/#wp-comments)

## Introduction

Lorsque l’on travaille avec un micro contrôleur on est vite limité en broches, par exemple un afficheur LCD en mode 4 bits utilise à lui seul 7 pins ( presque 1 port ), et le routage n’est pas très simple.

C’est pourquoi j’ai voulu déporter ce port. A l’aide d’un simple PCF8574P qui lit ou écrit le port complet qui coûte moins de 4€. Certains regarderont les versions commerciales et seront étonnés de voir une telle différence de prix, mais les versions commerciales ont un avantage: elles sont réalisées avec un micro contrôleur donc fonctionnent plus rapidement.

## Fonctionnement du PCF8574P

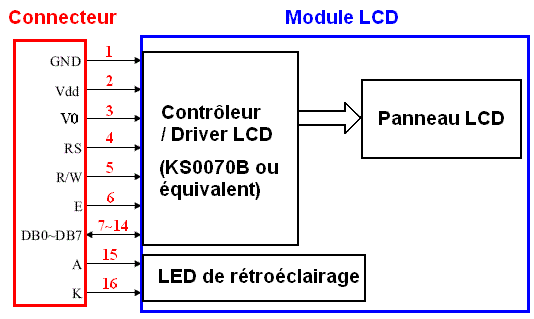
Le PCF8574P est un composant I2C ( fonctionne sur le BUS I2C qui est un bus à 2 fils)  
[Petit cour sur l’I2C](http://fr.wikipedia.org/wiki/I2C)

Pour faire simple le PCF8574P se configure en entrée ou en sortie (pour tout le port). Il sera donc impossible de connaître l’état de l’afficheur, on le commandera sans avoir de réponse, c’est pourquoi la version commerciale est plus rapide.

Le BUS I2C comporte 3 fils ( un 4 ème si on souhaite ramener l’alimentation), la masse, SCL (horloge) et SDA (donnée). Le pic 16F876 possède un module I2C, mais si votre micro contrôleur n’en possède pas, on peux utiliser 2 broches et le réaliser en programme ( je l’ai déjà fait avec un 68HC11 lors de mes études. PS : évitez ce micro contrôleur il y a 100 fois mieux pour moins cher )

## Fonctionnement d’un afficheur LCD

Un afficheur LCD est composé de plusieurs parties:

[](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/LCDdriver.gif)

\* Un contrôleur ( driver LCD) qui communique avec l’extérieur et gère l’affichage. Les contrôleurs sont compatibles entre eux ( ce n’est pas le cas avec les afficheurs LCD graphiques)

\* Led de rétro éclairage ( pour activer le rétro éclairage ou non )

\* Contraste: contraste de l’affichage ( valeur analogique comprise entre 0 et 5 v , pour les LCD graphiques, 0 et -5v)

Il existe plusieurs tailles d’écran, pour ma part , j’utilise un 16 x 2 (2 lignes de 16 caractères). Le module devrait être compatible avec les autres modèles.

### Description des broches

|  |  |  |  |
| --- | --- | --- | --- |
| **Numéro de broche du connecteur** | **Symbole** | **Type** | **Description** |
| 1 | GND | alimentation | masse (0V) |
| 2 | Vdd | alimentation | + 5 V (alimentation du contrôleur interne) |
| 3 | V0 | alimentation | alimentation du panneau LCD Ici a 0 v : contraste maximum |
| 4 | RS (Register Select) | entrée | \* RS = 1 : sélection du registre de données \* RS = 0 : o en mode écriture : sélection du registre d’instruction o en mode lecture : sélection du drapeau BUSY et du compteur d’adresse |
| 5 | R/W (Read / Write) | entrée | \* R/W = 1 : mode lecture \* R/W = 0 : mode écriture |
| 6 | E (Enable) | entrée | Entrée de validation |
| 4 | RS (Register Select) | entrée | \* RS = 1 : sélection du registre de données \* RS = 0 : o en mode écriture : sélection du registre d’instruction o en mode lecture : sélection du drapeau BUSY et du compteur d’adresse |
| 5 | R/W (Read / Write) | entrée | \* R/W = 1 : mode lecture \* R/W = 0 : mode écriture |
| 6 | E (Enable) | entrée | Entrée de validation |
| 7 | DB0 (Data bit 0) | entrée/sortie | Bus de données (8 bits)\* DB0 : bit de poids faible (LSB) \* DB7 : bit de poids fort (MSB)\* En mode écriture : bus configuré en entrée \* En mode lecture : bus configuré en sortie |
| 8 | DB1 (Data bit 1) | entrée/sortie |
| 9 | DB2 (Data bit 2) | entrée/sortie |
| 10 | DB3 (Data bit 3) | entrée/sortie |
| 11 | DB4 (Data bit 4) | entrée/sortie |
| 12 | DB5 (Data bit 5) | entrée/sortie |
| 13 | DB6 (Data bit 6) | entrée/sortie |
| 14 | DB7 (Data bit 7) | entrée/sortie |
| 15\* | A (Anode) | alimentation | alimentation du système de rétro éclairage |
| 16\* | K (Cathode) | alimentation |

\* seulement pour les modules LCD équipés de LED de rétro éclairage.

A noter: le contraste est toujours au maximum, si je réalise une V2, j’ajouterai le contraste.  
La résistance pour le rétro éclairage n’est pas toujours obligatoire, je ne l’ai pas mise. Et je n’ai pas mis d’interrupteur, si je réalise une V2, un bornier permettra d’activer ou non le rétro éclairage pour économiser de la batterie…

### Le jeu d’instruction

Le jeu d’instruction permet de communiquer avec l’afficheur en parlant le même langage. Tout n’est pas forcément utilisé/utile , mais si vous souhaitez faire clignoter quelque chose , je ne l’ai pas fait, il vous faudra donc le faire en fonction de ce tableau…

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Description | Durée (KS0070B à 270 kHz) |
| Display clear | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | \* Efface l’écran \* Place le curseur dans la position d’origine (1ère position à gauche, 1ère ligne : compteur d’adresse à 0x00) | 1,53 ms |
| Return home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | Place le curseur dans la position d’origine (1ère position à gauche, 1ère ligne : compteur d’adresse à 0x00) | 1,53 ms |
| Entry mode set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | I/D = 1 : Déplacement du curseur d’une position à droite (Incrémentation du compteur d’adresse)I/D = 0 : Déplacement du curseur d’une position à gauche (Décrémentation du compteur d’adresse)S(Shift) = 1 : Décalage de l’affichage dans le sens de déplacement du curseurS = 0 : L’affichage n’est pas décalé | 39 µs |
| Display On/Off control | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | \* D = 1 : affichage visible \* D = 0 : affichage invisible \* C = 1 : curseur visible \* C = 0 : curseur invisible \* B = 1 : curseur clignotant \* B = 0 : curseur fixe | 39 µs |
| Cursor or display shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | X | X | \* S/C = 1 : Décalage de l’affichage \* S/C = 0 : Déplacement du curseur \* R/L = 1 : Décalage à droite \* R/L = 0 : Décalage à gaucheN.B. Les données de la DDRAM\* ne sont pas modifiées. | 39 µs |
| Set Function | 0 | 0 | 0 | 0 | 1 | DL | N | F | X | X | \* DL = 1 : interface 8 bits \* DL = 0 : interface 4 bits \* N = 1 : 2 lignes d’affichage \* N = 0 : 1 ligne d’affichage \* F = 1 : matrice 5 x 11 points \* F = 0 : matrice 5 x 8 points | 39 µs |
| Set CGRAM address | 0 | 0 | 0 | 1 | a5 | a4 | a3 | a2 | a1 | a0 | Positionne le compteur d’adresse de la CGRAM\*\* | 39 µs |
| Set DDRAM address | 0 | 0 | 1 | a6 | a5 | a4 | a3 | a2 | a1 | a0 | Positionne le compteur d’adresse de la DDRAM | 39 µs |
| Write data to CGRAM or DDRAM | 1 | 0 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 | Write data to CGRAM : écrit la donnée (d7 …d0) à l’adresse de la CGRAM définie par l’instruction précédente Set CGRAM addressWrite data to DDRAM : écrit la donnée (d7 …d0) à l’adresse de la DDRAM définie par l’instruction précédente Set DDRAM address | 43 µs |
| Read Busy flag and address counter | 0 | 1 | BF | ac6 | ac5 | ac4 | ac3 | ac2 | ac1 | ac0 | Lit le drapeau BF (Busy Flag) et le compteur d’adresse (ac6 … ac0).Le drapeau Busy est disponible sur la sortie DB7.Quand le drapeau est au niveau 1, cela veut dire que le contrôleur est occupé à traiter une instruction.Le contrôleur n’accepte aucune nouvelle instruction tant que le drapeau est au niveau 1 : il faut donc attendre que le drapeau passe au niveau 0. | - |
| Read data from CGRAM or DDRAM | & | 1 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 | Read data from CGRAM : lit la donnée (d7 …d0) à l’adresse de la CGRAM définie par l’instruction précédente Set CGRAM addressRead data from DDRAM : lit la donnée (d7 …d0) à l’adresse de la DDRAM définie par l’instruction précédente Set DDRAM address | 43 µs |

X : 0 ou 1

\* DDRAM = Display Data RAM

\*\* CGRAM = Character Generator RAM

Les 2 derniers sont impossibles car on ne peut lire le port.

### Affichage

Pour afficher les caractères, il faut rentrer les valeurs ASCII correspondantes dans la DDRAM. Chaque case correspond à 1 caractère.

La première ligne commence à $00 et la seconde à $40. On utilise les caractères ASCII.

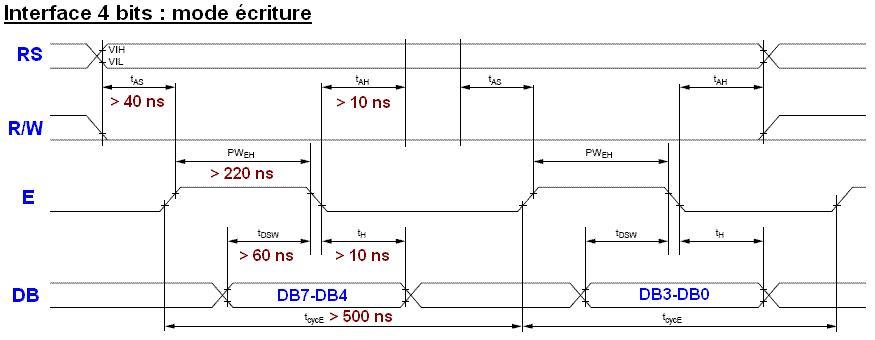
Des caractères sont personnalisables mais je ne les ai jamais utilisés.

### Mode 4 bits

Il existe 2 modes : le mode 8 bits et le mode 4 bits. Nous utiliserons le mode 4 bits qui permet de contrôler l’afficheur au complet sans avoir à utiliser plus d’un port.

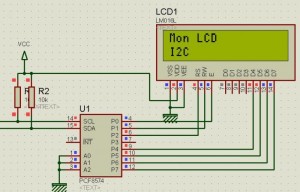
Il faut tout d’abord le mettre en mode 4 bits. (Utiliser la fonction startlcd)

Ensuite, il y a des règles pour communiquer en mode 4 bits, on transmet les bits de poids fort ensuite les bits de poids faible (c’est pourquoi le mode 4 bits est plus lent que le mode 8 bits)

[](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/write_timing_diagram.jpg)

Pour le programme , je ne me suis pas soucier des temps, l’I2C prend en moyenne 500 µs pour envoyer 1 octet. Donc normalement, je peux supprimer tous les delays.

## Schéma

[](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/Schema_LCD_I2C.jpg)

Comme on peut l’observer, l’adresse du PCF8574P est : 0b01000000 , mais on peut la modifier ( en modifiant le typon )

Ce système est très bien quand il nous manque 1 port et évite d’avoir beaucoup de fils ou de grosses nappes qui se baladent…

## Le typon

[Le typon format PDF](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/Typon_afficheur_lcd_I2C.pdf)

Nomenclature  
1 afficheur LCD 2 lignes  
Un composant I2C PCF8574P  
2 résistances de 10 K (facultatif) car le Bus I2C utilise ses entrées à collecteur ouvert  
2 bornier double (les remplacer par des autres types de connecteurs peux être ingénieux …)  
1 connecteur HE14 1 rangée femelle (à placer sur la carte)  
1 connecteur HE14 1 rangée mâle (à placer sur l’afficheur)

## Programme

**Le programme a été réalisé pour les pics avec le compilateur CSS. Pour un autre compilateur , ou un autre composant , il faudra modifier le programme**

#include &lt;16f876A.h&gt;

#fuses HS, NOWDT, NOPROTECT, NOPUT, NOBROWNOUT, NOLVP

#use delay (clock=20000000)

#use I2C(master, sda=PIN\_C4, scl=PIN\_C3)

#define addresse\_LCD 0b01000000

//////////////////////////////////////////////////////////////////////////////

//Programme réalisé par Matthieu

//Janvier 2008

//

//Les fonctions :

//

//       startlcd() // initialise l'afficheur ( à placer au départ)

//       clear() // efface le contenu de l'afficheur

//       ligne\_suivante()//passage à la ligne suivante

//       affich("Message")// affiche message.

//       création de lcd\_putc(message)

//             f clear\_lcd

//             ligne\_up

//

//

//

//       Règle de câblage :

//          D0&gt; E

//          D1&gt; RS  NEW R/W

//          D2&gt; R/W NEW : RS

//          D3&gt; NC

//          D4&gt; D4

//          D5&gt; D5

//          D6&gt; D6

//          D7&gt; D7

//

//////////////////////////////////////////////////////////////////////////////

void envoi(unsigned char data)

{

i2c\_start();

i2c\_write(addresse\_LCD);//adresse du composant

i2c\_write(data);

i2c\_stop();

}

void valid\_e(unsigned char data)

{

data=data|0b00000001;//E est a 1

envoi(data);

delay\_ms(1);

data=data-1;//E=0

envoi(data);

delay\_ms(1);

}

void startlcd()

{

//On travaille sur la commande donc RS=0 R/W=0 écriture

valid\_e(0x00);

valid\_e(0x10);

valid\_e(0x00);

valid\_e(0x00);

valid\_e(0x10);

//Passage en mode 8 bit

valid\_e(0x30);

valid\_e(0x30);

// on passe en mode 4 bits

valid\_e(0x30);

valid\_e(0x20);

//mode 4 bits, 2 lignes, caractères 5x7 (28h)

valid\_e(0x20);

valid\_e(0x80);

//affichage en fonction, pas de curseur (0Ch)

valid\_e(0x00);

valid\_e(0xC0);

//le curseur se déplace vers la gauche (06h)

valid\_e(0x00);

valid\_e(0x60);

//on efface l'afficheur

valid\_e(0x00);

valid\_e(0x10);

}

void clear\_lcd()

{

valid\_e(0x00);

valid\_e(0x10);

}

void ligne\_up()

{

//40h

valid\_e(0xC0);

valid\_e(0x00);

delay\_ms(1);

}

void affich(char mess)

{

//Le message passe lettre par lettre , donc cette fonction passe en boucle

char data;

data=mess&amp;0xF0;

data=data+4; //RS a 1

valid\_e(data);

data=mess&amp;0x0F;

data=data&lt;&lt;4;

data=data+4;//RS a 1

valid\_e(data);

}

//Attention CSS ne respecte pas les conventions sur les chaines de caractères en vrai il faudrait faire quelque chose du style:

/\*

void new\_lcd\_putc(char \* c){

for(c[0]!=0){

lcd\_putc(c);

c++;

}

}

\*/

void lcd\_putc(char c)

{

switch(c)

{

case'\f':

clear\_lcd();

break;

case'\n':

ligne\_up();

break;

default:

affich(c);

break;

}

}

void main()

{

startlcd();

while(1)

{

lcd\_putc("\fMon LCD\nI2C");

delay\_ms(1000);

lcd\_putc("\f");

lcd\_putc("Fonctionne");

delay\_ms(1000);

}

}

## Photos

[](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/photo_LCDI2C1.jpg) [](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/photo_LCDI2C2.jpg)  
[](http://ares.asso-ensea.net/robotics/wp-content/uploads/2013/12/photo_LCDI2C3.jpg)

Une amélioration qui pourrait être bien: agrandir le typon et pouvoir ajouter des vis pour donner moins de choc aux fixations. On pourrait aussi facilement visser notre LCD I2C sur notre robot à l’aide d’une équerre.

## Avertissement

L’I2C fonctionne à une certaine vitesse, si vous utilisez n’importe quel câble vous risquez de vous faire avoir. Par exemple un câble RJ11 droit ne dois pas être trop long, car 2 fils l’un à coté de l’autre forment un condensateur, et un condensateur forme un joli filtre passe bas.

Un système qui fonctionne bien : 4 fils avec un bornier.

Les fils de couleurs ou des détrompeurs peuvent vous éviter de griller vos composants ( c’est idiot , mais sous la précipitation ou tard le soir , c’est parfois utile)

Si vous souhaitez améliorer le système vous êtes prié d’en faire profiter le plus de monde possible.

# <http://www.simius.be/modules-test/afficheur-sur-bus-i2c.html>

## Afficheur sur bus I2c

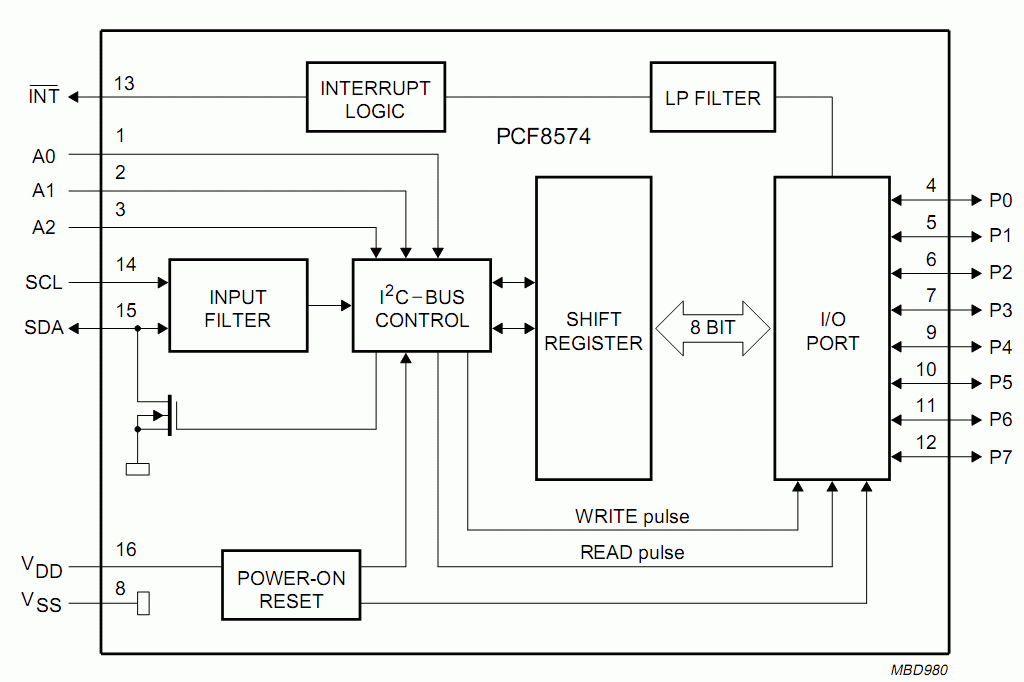
Il est toujours plus facile, lorsque l'on fait du développement sur des circuits à microcontrôleurs, d'afficher certaines variables ou état sur un [afficheur LCD](http://www.simius.be/component/content/?id=7&Itemid=10) pour permettre un débogage plus facile, mais dans le cas des afficheurs les plus courants et les moins cher, ceux-ci ne sont pourvu que de lignes parallèles (bus de donnée et lignes de contrôles) ce qui impliques beaucoup de connections ainsi qu'une grande consommation de ligne E/S sur le microcontrôleur.

Une solution simple existe afin de réduire fortement ces connexions, l'utilisation du bus I2c, ce bus que la plupart des microcontrôleurs possèdent et qui est assez facile à mettre en œuvre car il ne comporte que deux lignes (SCL, SDA) ainsi que l'alimentation (+5V, GND).

Dans notre cas, nous allons donc réaliser une interface parallèle de 8 lignes vers un bus I2c. Cette interface aura la possibilité de 8 E/S lesquelles pourront être programmées via le bus I2C et ainsi nous aurons, par le biais de cette interface, la possibilité de contrôler un afficheur, un clavier, des relais, des LEDs et encore bien d'autre chose.

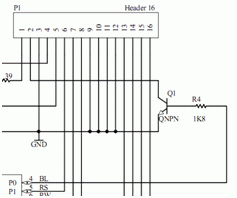
Pour ce faire, nous allons donc utiliser un circuit intégré du type PCF8574 qui existe en plusieurs versions avec des plages d'adresse différentes. Ce qui va nous permettre de placer sur le bus I2c, plusieurs de ces modules afin de contrôler plusieurs afficheurs ou d'autres périphériques interfacés.

## **Schéma interne du PCF8574**



Sur cette interface, il y a la possibilité de sélectionner sa plage d'adresse par de petit pontage à réaliser par un court-circuit de soudure sur la platine afin de mettre ou pas à la masse une de ces lignes. Nous avons aussi la possibilité de faire la même chose avec les résistances de Pull Up qui se trouvent sur les lignes I2C dans le cas ou le microcontrôleur ne serait pas équipé de ces résistances.

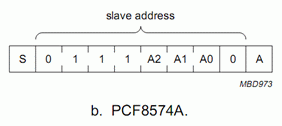
Il y a aussi sur cette platine, une commande par transistor pour actionner le rétroéclairage qui se trouve aussi sur certain afficheur LCD, ou une LED supplémentaire.



Si vous voulez plus d'information sur le PCF8574, vous pouvez télécharger son [datasheet](http://www.simius.be/images/datasheet/pcf8574.pdf).

## **Adressage**

**L'adressage du PCF8574 est fait comme suit:**



Ce qui veut dire que le PCF8574 a une plage d'adresse qui peut aller de 40 à 4E selon les niveaux des divers lignes d'adresses et le PCF8574A a une plage allant de 70 à 7E ce qui nous permet d'utiliser 8 adresses différentes sans que les interfaces se gênent entre eux.

***huit adresses, Pourquoi ??***

il ne faut pas oublier que l'adressage du Bus I2C se fait sur 7 bits et non 8, car le bit le plus à droite est le bit du choix de lecture ou d'écriture sur le bus (voir paragraphe I2C).

Sur notre platine l'adressage du PCF8574 est câblé comme suit:



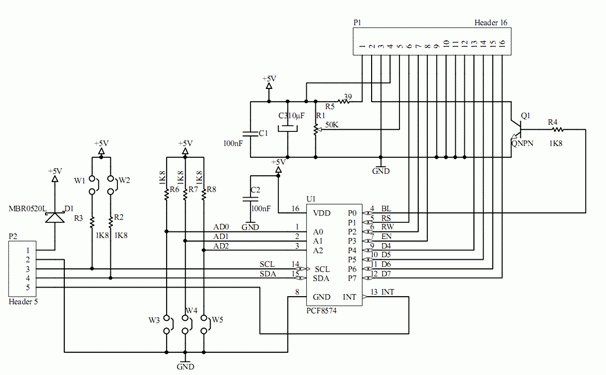
W3, W4 et W5 ne sont pas réaliser, donc les trois adresses sont au VCC.



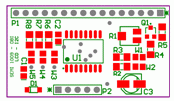
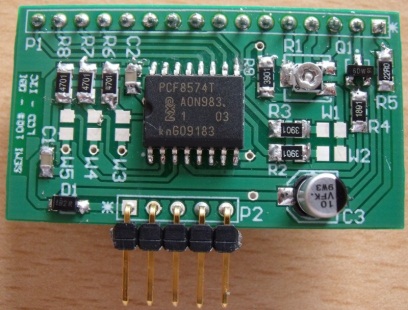
Ce qui fait donc A2=1, A1=1 et A0=1, ce qui nous donne l'adresse 01001110 soit 4E en hexadécimal.

Donc dans notre programme l'adresse 0x4E sera utilisée pour accéder à cette interface.

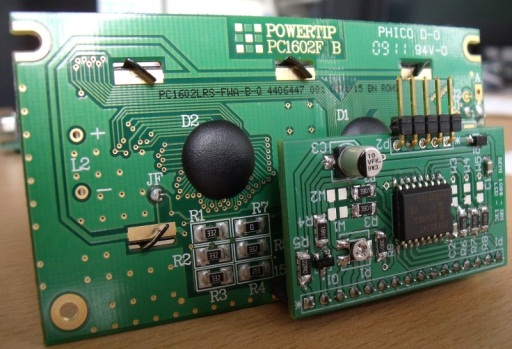
**Schéma complet de la platine**

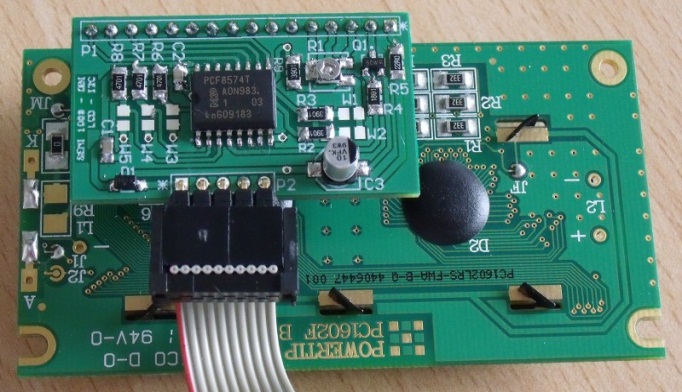


**Implantation des composants**

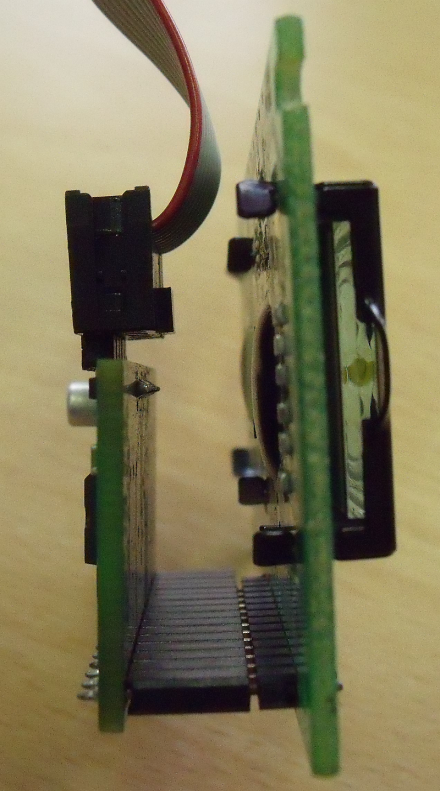


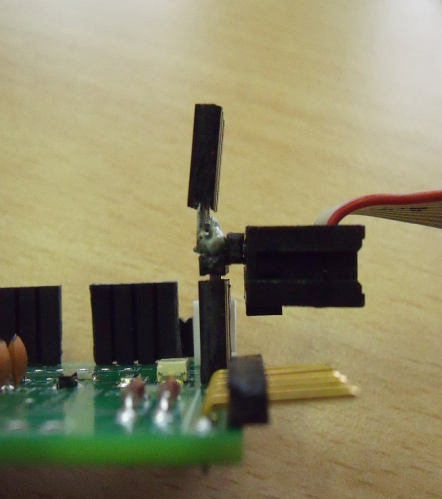
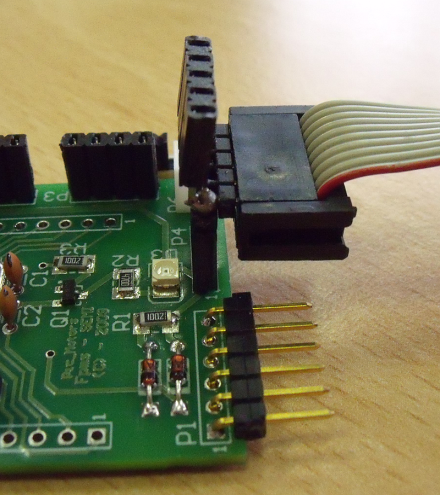
**Fonctionnement**





**Connexions sur la carte MiniSiMiUS**



# <http://www.tobozzo.free.fr/html/pages-electron/saa1064.htm>

Je vous présente ici un module d'affichage à base d'afficheurs 7 segments: Celui ci est pilotable via une interface I2c, connectable en externe:

Le but de ce montage, n'est pas de révolutionner l'électronique, mais de mettre à disposition des internautes un circuit facilement réalisable.

Les afficheurs 7 segments utilisés sont des HDSP- 5301 HP 813E ou LTS547aR .

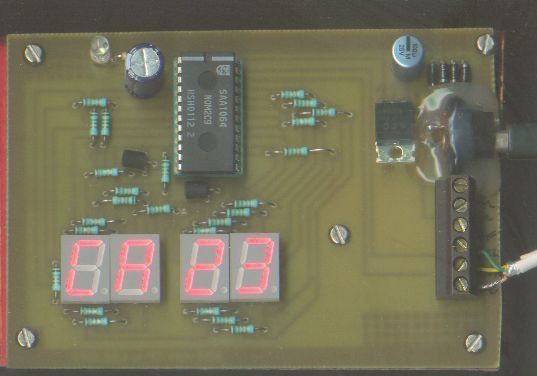
(1) Voici le [schéma électrique](http://www.tobozzo.free.fr/html/pages-electron/images/saa1064/020525A.GIF) complet ainsi que le [databook](http://www.tobozzo.free.fr/telech/electron/databook/saa1064.pdf) du Saa1064

Une alimentation externe est nécessaire. Une jack d'alimentation (DC 8V mini, AC 6V mini) femelle est prévue à cet effet

(2) Le typon au [format gif](http://www.tobozzo.free.fr/html/pages-electron/images/saa1064/020525C.GIF), au [format PDF](http://www.tobozzo.free.fr/html/pages-electron/images/saa1064/projet-saa1064.pdf) http://www.tobozzo.free.fr/images/new.gif

, et la [vue des composants](http://www.tobozzo.free.fr/html/pages-electron/images/saa1064/020525B.GIF) classiques

(3) Une photo de la carte cablée:



Tout est provisoire pour l'instant, et, je n'ai mis aucun soft en ligne

# <http://www.pobot.org/Piloter-des-afficheurs-7-segments.html?lang=fr>

Pas intéressant (pour l’instant)

# <http://skyduino.wordpress.com/2011/11/10/tutoriel-arduino-communication-par-spi-hard-et-soft-et-par-i2c/>

## Communication par SPI (hard et soft) et par I2C

Le but de ce tutoriel est de comprendre comment communiquer par SPI et par I2C avec des périphériques externes, le tout avec une carte arduino.

Pour les curieux :  
[La page wikipédia sur le bus I2C](http://fr.wikipedia.org/wiki/I2C)  
[La page wikipédia sur le bus SPI](http://fr.wikipedia.org/wiki/Serial_Peripheral_Interface)

Voici la liste des composants nécessaire pour chaque partie du tutoriels :  
1ere partie, Horloge I2C + afficheur 7 segments :  
1x carte arduino (uno pour moi),  
1x PCF8575 + support cms -> DIL,  
2x résistances de 10K,  
2x réseau de résistances de 1K (!! réseau DIL (dual in line) pas SIL (single in line)),  
2x afficheurs 7 segments à anode commune (CA), par exemple SA52-11EWA ou un HDSP-5501,  
1x module RTC sur base de DS1307 (exemple : module RTC de sparkfun, adafruit, ...),  
1x condensateur de 100nF (optionnel),  
et beaucoup de fils (~40) !  
Schéma : [Format JPG](http://skyduino.wordpress.com/?attachment_id=2068)

2eme partie, SPI Hardware :  
1x carte arduino (uno pour moi),  
1x "Rotary Encoder LED Ring" de Mayhew Labs,  
1x Protoshield + breadbard de sparkfun,  
et 8 fils.  
Schéma : voir vidéo

3eme partie, SPI software, shiftOut :  
1x carte arduino (uno pour moi),  
1x 74HC595,  
2x résistances de 10K,  
1x condensateur de 100nF (optionnel),  
1x réseau de résistances de 1K (SIL),  
8 x led rouge 5mm,  
et une 20aine de fils.  
Schéma et détails : [Schéma arduino.cc](http://www.arduino.cc/en/Tutorial/ShiftOut)

4eme partie, SPI software, shiftIn :  
1x carte arduino (uno pour moi),  
1x CD4021,  
1x condensateur de 100nF (optionnel),  
1x réseau de résistances de 1K (SIL),  
1x "mini switch" à 8 interrupteurs,  
et une 20aine de fils.  
Schéma et détails : [Schéma arduino.cc](http://www.arduino.cc/en/Tutorial/ShiftIn)

Les liens vers la référence des librairies et fonctions « importante » de ce tutoriel :  
[Librairie Wire (I2C)](http://arduino.cc/en/Reference/Wire)  
[Librairie SPI](http://arduino.cc/en/Reference/SPI)  
[Référence de ShiftOut](http://arduino.cc/en/Reference/ShiftOut)  
[Référence de ShiftIn](http://arduino.cc/en/Reference/ShiftIn)

Et pour finir bien évidement le tutoriel :

Pour ceux qui est des différents sketches de ce tutoriel :

Le sketch de la partie I2C :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | // Inclusion de la librairie I2C  #include <Wire.h>    // Adresse I2C du module RTC  #define DS1307\_ADDRESS 0x68  // Adresse I2C du PCF8575  #define PCF8575\_ADDRESS 0x20    // Conversion BCD -> decimal  byte bcdToDec(byte val)  {    return ((val/16\*10) + (val%16));  }    // Tableau de correspondance chiffre -> 7-segments  byte digits[10] = {    B10000001 ,    B11101101 ,    B01000011 ,    B01001001 ,    B00101101 ,    B00011001 ,    B00010001 ,    B11001101 ,    B00000001 ,    B00001101  };    void setup()  {    Wire.begin(); // Configuration du bus I2C  }    void loop()  {    byte secondes = recupSecondes(); // Récupération des secondes depuis le module RTC    afficheNombre(secondes); // Affichage sur le double 7-segments    delay(1000); // Attente de 1s  }    // Retourne le nomre de secondes de l'heure courante stocké dans le module RTC  int recupSecondes()  {    Wire.beginTransmission(DS1307\_ADDRESS); // Engage la communication    Wire.send(0); // Démande l'adresse 0 (-> secondes)    Wire.endTransmission(); // Termine la communication      Wire.requestFrom(DS1307\_ADDRESS, 1); // Demande une réponse de 1 octet en provenance du module RTC    return bcdToDec(Wire.receive()); // Converti le résultat (en bcd) en décimal (base 10)  }    // Affiche un nombre décimal de 0 à 99 sur le double afficheur 7-segments  void afficheNombre(byte nombre)  {    byte x = nombre % 10; // chiffre des unités    byte y = nombre / 10; // chiffre des dizaines    Wire.beginTransmission(PCF8575\_ADDRESS); // Engage la communication    Wire.send(digits[x]); // Envoie le 1er octet (bits 0 ~ 8 )    Wire.send(digits[y]); // Envoie le 2eme octet (bits 9 ~ 16)    Wire.endTransmission(); // Termine la communication  } |

+ Bonus le script python pour générer les valeurs des digits :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | byteOrder = "gfabedc."  digits = [ "abcdef" , "bc" , "abged" , "abcdg" , "bcfg" , "afgcd" , "acdefg" , "abc" , "abcdefg" , "abcfg" ]    print "byte digits[10] = {"  for digit in digits:    nstr = "B"    for letter in byteOrder:      if letter in digit:        nstr +=  "0"      else:        nstr +=  "1"    print nstr,","  print "};" |

Le sketch de la partie SPI hardware :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // Inclusion de la librairie SPI hardware  #include <SPI.h>    // Tableau d'effets lumineux (ici un simple scroll avec un led)  unsigned int effect[16] = {1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768};    void setup()  {    SPI.begin(); // Configuration du port SPI hardware  }    void loop()  {    for(byte i = 0 ; i < 16 ; i++) // Pour chacune des 16 leds    {      unsigned int n = effect[i]; // Récupération de "l'image" correspondant à i dans la tableau des effets lumineux      SPI.transfer(n & 0xFF); // Envoi du LSB (1er octet, bits 0 ~ 8 )      SPI.transfer((n & 0xFF00) >> 8); // Envoi du MSB (2eme octet, bits 9 ~ 16)      delay(250); // Avec un petit effet K2000 ce sera plus marant =P    }  } |

Le sketch de la partie ShiftOut :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #define CS\_PIN 10  #define DS\_PIN 11  #define CLK\_PIN 12    void setup()  {    Serial.begin(9600); // Configuration du port série    pinMode(CS\_PIN, OUTPUT); // broche CS -> Chip Enable    pinMode(DS\_PIN, OUTPUT); // broche DS -> Data Serial    pinMode(CLK\_PIN, OUTPUT); // Broche CLK -> Clock  }    void loop()  {    if(Serial.available() > 2) // Attente de 3 caractéres sur le port série    {      int nombre = (Serial.read() - '0') \* 100; // Récupération du nombre en décimal sur 3 chiffres      nombre += (Serial.read() - '0') \* 10;      nombre += Serial.read() - '0';      nombre &= 255; // Protection contre le nombre >255 qui pourrait entrainer un bug de shiftOut        digitalWrite(CS\_PIN, LOW); // On sélectionne le 74HC595 -> en attente d'un octet      shiftOut(DS\_PIN, CLK\_PIN, MSBFIRST, (byte)nombre); // On envoi l'octet (cast en byte pour forcer l'envoi d'un seul octet)      digitalWrite(CS\_PIN, HIGH); // On déselectionne le 74HC595 -> Affichage de l'octet    }  } |

Le sketch de la partie ShiftIn :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #define CS\_PIN 10  #define DS\_PIN 11  #define CLK\_PIN 12    void setup()  {    Serial.begin(9600); // Configuration du port série    pinMode(CS\_PIN, OUTPUT); // broche CS -> Chip Enable    pinMode(DS\_PIN, INPUT); // broche DS -> Data Serial    pinMode(CLK\_PIN, OUTPUT); // Broche CLK -> Clock  }    void loop()  {    digitalWrite(CS\_PIN, HIGH); // On place CS à HIGH pour collecter les données paralléles de Q0 à Q7    delayMicroseconds(20);    digitalWrite(CS\_PIN, LOW); // A partir de maintenant le CD021 devrait avoir collecté les états de Q0 à Q7      byte n = shiftIn(DS\_PIN, CLK\_PIN, MSBFIRST); // On récupére l'octet depuis le CD4021      // Affichage des valeurs des deux encodeurs    Serial.print("V1 & V2 :");    Serial.println(n, BIN); // Bug bit 8 ?    delay(1000);  } |

+ Bonus le sketch SPI – Read :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <SPI.h>    void setup()  {    Serial.begin(9600); // Configuration du port série    SPI.begin(); // Configuration du port SPI    SPI.setClockDivider(SPI\_CLOCK\_DIV128);    pinMode(10, OUTPUT);  }    void loop()  {    digitalWrite(10, LOW);    byte n = SPI.transfer(0); // On récupére l'octet depuis le CD4021    digitalWrite(10, HIGH);      // Affichage des valeurs des deux encodeurs    Serial.print("V1 & V2 :");    Serial.println(n, BIN);    delay(1000);  } |

# <http://arduino103.blogspot.fr/2013/03/connecter-des-arduino-ensembles-un.html>

### Connecter des Arduino ensembles - Un tutoriel I2C pour Arduino

Nous venons de terminer [un Tutoriel en Français sur I2C](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro) accessible sur notre Wiki.  
*Voici quelque-unes des informations que nous avons collectées dans notre tutoriel*.  
  
Arduino supporte le [protocole I2C](http://en.wikipedia.org/wiki/I2C) (Inter Integrated Circuit)... aussi appelé protocole 2 fils.  
  
Ce dernier permet d'établir des connexions asynchrones entre plusieurs composants intelligent pour partager des informations via un "bus commun".  
  
Si ce protocole est utilisé pour permettre à un Arduino de communiquer avec des senseurs (un réseau de senseur), il permet aussi à des Arduino de communiquer directement ensembles... ou de communiquer avec un système plus évolué tel d'un Raspberry Pi.

|  |
| --- |
| [http://1.bp.blogspot.com/-SHAEt2vRl6A/UTz4t6fMrAI/AAAAAAAADI4/Ecj3St_32iE/s400/Arduino-I2C-Intro-Montage2Arduino.jpg](http://1.bp.blogspot.com/-SHAEt2vRl6A/UTz4t6fMrAI/AAAAAAAADI4/Ecj3St_32iE/s1600/Arduino-I2C-Intro-Montage2Arduino.jpg) |
| [Connexion entre Arduino via le bus I2C](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro) |

Le maître et les esclaves  
Sur un bus I2C, il y a un Maître et plusieurs Esclaves. Le maître gère la communication sur le bus et interroge les différents esclave.  
  
Les esclaves n'ont droit a aucune initiative de communication, il doivent simplement attendre une requête en provenance du maître.  
  
[Ce concept est plus largement détaillé dans notre tutoriel](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro).  
  
Le concept "Master Reader"  
Le "maître lit".  
Dans ce concept, le maître se connecte sur un esclave de son choix et envoi une commande/demande à l'esclave pour recevoir de l'information   
**Ce concept est pratique pour collecter les informations de différents senseurs.**

|  |
| --- |
| [http://3.bp.blogspot.com/-6nT0rjmkEg4/UTz3SgGDGuI/AAAAAAAADIw/Zbbd-Vmvs7M/s400/Arduino-I2C-Intro-MasterReader.jpg](http://3.bp.blogspot.com/-6nT0rjmkEg4/UTz3SgGDGuI/AAAAAAAADIw/Zbbd-Vmvs7M/s1600/Arduino-I2C-Intro-MasterReader.jpg) |
| Source: [Tutoriel I2C sur le Wiki de MCHobby](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro) |

Pour être précis, le nom complet de ce concept est "Master Reader/Slave Sender" (le maître lit et les esclaves envoient/parlent) .  
  
Le concept "Master Writer"  
Le "maître écrit".  
Dans ce concept, le maître écrit/envoi des informations vers de multiples esclaves. **Ce concept est bien pratique pour commander différents actionneurs ou répartir des ordres vers les différents éléments physiques** (comme, par exemple, les servos moteurs I2C d'un robots).   
  
Pour être précis, le nom complet de ce concept est "Master Writer/Slave Receiver".  
  
Raccordement I2C  
**Brochage I2C**  
Le Bus I2C utilise les broches A4 et A5 sur une Arduino Uno. Il suffit de raccorder les broches A4 ensemble et A4 ensemble (et partager la même masse) pour constituer le Bus I2C.  
  
Notez que ces broches ne sont pas identiques pour toutes les plateformes, en voici un résumé.  
  
Arduino Uno

* SDA sur la broche I2C Data: Pin A4
* SCL sur la broche I2C Clock: Pin A5

Arduino Mega

* SDA sur la broche I2C Data: Pin 20
* SCL sur la broche I2C Clock: Pin21

Arduino Leonardo

* SDA sur la broche I2C Data: Pin Digital 1
* SCL sur la broche I2C Clock: Pin Digital 3

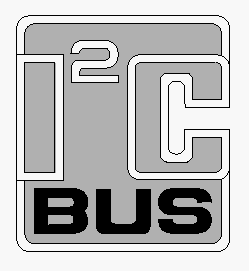
**Deux Arduino Uno sur un Bus I2C**   
Voici comment raccorder deux Arduino sur un Bus I2C.

|  |
| --- |
| [http://3.bp.blogspot.com/-LO3l2DqJ6fQ/UERNod3-iYI/AAAAAAAABpQ/1-pudAh-R7A/s400/Arduino_I2C.jpg](http://3.bp.blogspot.com/-LO3l2DqJ6fQ/UERNod3-iYI/AAAAAAAABpQ/1-pudAh-R7A/s1600/Arduino_I2C.jpg) |
| Bus I2C entre deux Arduino Uno Cliquer pour Agrandir. |

Le code  
Vous trouverez tous [les exemples (avec commentaire en français) relatif à cet sur notre wiki](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro).  
  
**Communiquer sur un bus I2C réclame un code avec un minimum d'explication, c'est pour cela que nous n'avons pas copier/coller ce dernier dans l'article.**  
  
[Ces codes d'exemple sont accessibles sur GitHub](https://github.com/mchobby/I2C_Intro) et également [disponible sur notre wiki](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro).  
  
Ressources

* [Notre article sur le wiki de MCHobby](http://mchobby.be/wiki/index.php?title=Arduino_I2C_Intro).
* [Master Reader/Slave Sender](http://arduino.cc/en/Tutorial/MasterReader)
* [Master Writer/Slave Receiver](http://arduino.cc/en/Tutorial/MasterWriter)
* [Librairie Wire](http://arduino.cc/en/Reference/Wire) (Arduino.CC)
* [Exemples](http://tronixstuff.wordpress.com/2010/10/20/tutorial-arduino-and-the-i2c-bus/) utilisant divers composants utilisant le bus I2C (*Tronix Stuff, anglais*)

# <http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieWire>



## Description

Cette librairie vous permet de communiquer avec les composants utilisant le protocole I2C / TWI (communication série sur 2 fils).

Pour plus de détails sur le bus I2C : <http://fr.wikipedia.org/wiki/I%C2%B2C>

## Connexions

Sur la plupart des cartes Arduino :

* la ligne SDA (ligne de données) est sur la broche analogique 4
* la ligne SCL (ligne d'horloge) est sur la broche analogique 5

Sur la carte Arduino Mega :

* SDA est sur la broche numérique 20
* SCL est sur la broche numérique 21

## Les fonctions de la librairie Wire

### Fonctions d'initialisation

* [begin()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wirebegin) : initialise communication avec Arduino "maître"
* [begin(adresse)](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wirebegin) : initialise communication avec Arduino "esclave"

### Fonctions "mode maître"

* [requestFrom(adresse, quantite)](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.WirerequestFrom) : demande de données à un esclave
* [beginTransmission(adresse)](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.WirebeginTransmission) : débute communication avec un esclave (ouvre stockage données à envoyer avec write)
* [endTransmission()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.WireendTransmission) : envoi des données vers esclave
* [write()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wirewrite) : écrit les données à envoyer vers esclave
* [available()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wireavailable) : test si données disponibles en provenance esclave (cf requestFrom)
* [read()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wireread) : lit les données en provenance de l'esclave

### Fonctions "mode esclave"

* [write()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wirewrite) : envoie les données vers le maître après requête
* [available()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wireavailable) : test si données disponibles en provenance du maître (cf onReceive)
* [read()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wireread) : lit données en provenance maître
* [onReceive(fonction)](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.WireonReceive) : définit la fonction à appeler sur réception de données en provenance du maître
* [onRequest(fonction)](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.WireonRequest) : définit la fonction à appeler sur requête du maître

### Fonctions obsolètes

* [send()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wiresend) - obsolète
* [receive()](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Wirereceive) - obsolète

## Principe d'utilisation

Pour inclure la librairie Wire dans un programme, on ajoutera au début du programme la ligne suivante :

#include <Wire.h>

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieWire?action=sourceblock&num=1)

Pour ce faire, le plus simple est d'utiliser le menu du logiciel Arduino Programme > Importer Librairie > Nom\_librairie

Voir également l'instruction [#include](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Include)

## Note

Il existe deux versions d'adresses I2C en 7 et en 8 bits. Les adresses 7 bits identifient le composant, les adresses 8 bits déterminent si on écrit ou on lit le composant. La librairie Wire utilise les adresses 7 bits. Si vous avez une fiche technique ou un exemple de code qui utilise des adresses 8 bits, vous devrez éliminer le bit de poids faible ( en décalant la valeur de l'adresse d'un bit vers la droite - voir notamment [l'opérateur de décalage à droite >>](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Decalage)), pour obtenir une adresse comprise entre 0 et 127.

## Exemples d'utilisation :

### Sur le site Arduino anglais :

* [SFRRanger\_reader](http://arduino.cc/en/Tutorial/SFRRangerReader): read a Devantech SRFxx ultra-sonic range finder using I2C communication.
* [digital\_potentiometer](http://arduino.cc/en/Tutorial/DigitalPotentiometer): control a AD5171 digital pot using the Wire Library.
* [master reader/slave sender](http://arduino.cc/en/Tutorial/MasterReader): set up two (or more) arduino boards to share information via a master reader/slave sender configuration.
* [master writer/slave reader](http://arduino.cc/en/Tutorial/MasterWriter): allow two (or more) arduino boards to share information using a master writer/slave reader set up.

### Sur le site www-mon-club-elec.fr :

# <http://www.pihomeserver.fr/2013/08/13/raspberry-pi-home-server-arduino-lier-les-deux-via-bus-i2c/>

## Raspberry Pi – Arduino – Lier les deux via le bus I2C

**Certains disent qu’on ne lie pas des chiens avec des saucisses mais des fois pourquoi pas. Alors le Raspberry Pi et l’Arduino ne sont pas forcement aussi opposés, avec même des points communs mais chacun à des avantages qui peuvent être bons de combiner.**

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/RaspArduinoI2C.png)Avec le Raspberry Pi vous pouvez via les ports GPIO contrôler des composants externes et faire de la bidouille en électronique. Mais j’imagine que certains maitrisent déjà bien leur Arduino, qu’ils ont le matériel, etc. Et puis ça fait un tuto en plus :-D

Pour lier les deux, nous allons passer par le protocole I2C. Plusieurs raisons à ça : ne pas utiliser le port USB, le Raspberry Pi étant limité (si vous voulez le faire quand même il y a un article dans le [MagPi #15](http://magpi.finalart.hu/The-MagPi-issue-15-en.pdf) et j’en ferai un moi même plus tard). Et ne pas utiliser le port série. Ici c’est un peu plus personnel, car j’ai une carte [Slice Of Pi](http://www.pihomeserver.fr/2013/07/07/raspberry-pi-home-server-assemblage-de-la-carte-slice-of-pi/) sur un Raspberry pour [la station météo](http://www.pihomeserver.fr/2013/07/24/station-meteo-raspberry-pi-home-server-une-station-meteo-avec-une-interface-web/) et elle utilise le port série pour [les sondes sans fil](http://www.pihomeserver.fr/2013/07/25/station-meteo-raspberry-pi-sondes-xrf-sans-fil-en-action/).

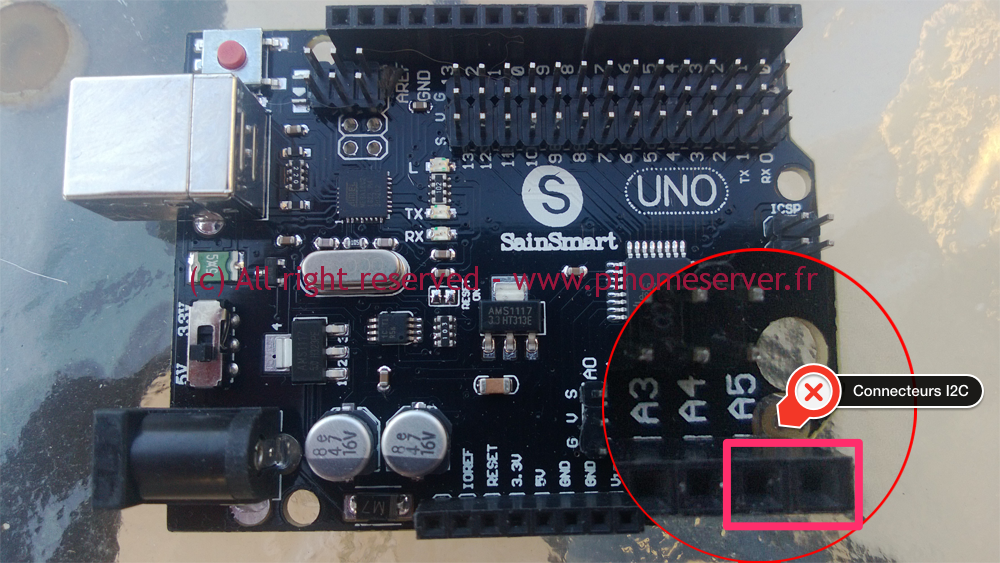
Le protocole I2C, créé par Philips avec une orientation domotique, permet à plusieurs composants de dialoguer entre eux de manière bidirectionnel mais en half-duplex uniquement. Comme d’habitude, plus d’infos chez [Wikipedia](http://fr.wikipedia.org/wiki/I2C).

Il y a un principe de maitre et esclave. Ici nous allons donc utiliser le Raspberry Pi en tant que maitre et l’Arduino en tant qu’esclave.

Avant tout pour connecter les deux appareils il faut savoir quelles broches utiliser !  
Pour le raspberry c’est sur les GPIO 2 et 3 (pour les cartes à partir du modèle B révision 2, ceux avec les trous de fixation dans la carte)

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/raspberry-i2c.png)

Pour l’Arduino, ça se passe sur les broches A4 et A5 pour les cartes de type Arduino Uno. Pour les autres cartes, vous trouverez plus d’informations sur [le site officiel d’Arduino](http://arduino.cc/en/Reference/Wire).

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/arduino-i2c-2.png)Il faudra ajouter une connexion à la masse entre les deux. Donc vous aurez besoin de trois fils.

Avant de connecter nos deux appareils, il faut s’assurer qu’ils vont pouvoir communiquer. Toutes les informations sont sur le site [d’adafruit](http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c) mais en gros, voici ce qu’il faut faire.

## Définir l’Arduino comme esclave

C’est relativement simple en incluant le module Wire dans le programme. J’ai pris l’adresse 0x12 mais vous pouvez choisir de 0x03 à 0x77 (en hexa) :

#include <Wire.h>

#define SLAVE\_ADDRESS 0x12

int dataReceived = 0;

void setup() {

Serial.begin(9600);

Wire.begin(SLAVE\_ADDRESS);

Wire.onReceive(receiveData);

Wire.onRequest(sendData);

}

void loop() {

delay(100);

}

void receiveData(int byteCount){

while(Wire.available()) {

dataReceived = Wire.read();

Serial.print("Donnee recue : ");

Serial.println(dataReceived);

}

}

void sendData(){

int envoi = dataReceived + 1;

Wire.write(envoi);

}

Vous l’aurez compris, le programme initialise l’Arduino avec l’adresse 0x12, définit la méthode de réception des données, la méthode d’envoi des données et c’est parti pour l’envoyer vers la carte avec l’interface.

## Activer la gestion I2C sur le Raspberry Pi

Par défaut, des distributions de type Raspbian désactivent la gestion I2C pour économiser la mémoire utilisée par le module. Pour réactiver, rien de plus simple et c’est dans le fichier /etc/modprobe.d/raspi-blacklist.conf. Il faut mettre en commentaire avec un # la ligne :

blacklist i2c-bcm2708

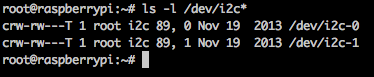
Reste à activer le module lors de chaque démarrage. C’est dans le fichier /etc/modules, en y ajoutant à la fin une nouvelle ligne :

i2c-dev

Ensuite vous installez la suite des outils de gestion I2C

apt-get install i2c-tools

Il va vous falloir redémarrer le Raspberry pour faire apparaitre deux fichiers :

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/1._ssh-2.png)  
Grâce à eux vous allez pouvoir communiquer sur le bus I2C !

Vous pouvez lister les périphériques avec la commande :

i2cdetect -y 1

Pour les modèles 256Mo (comme celui que j’utilise pour mes tests) :

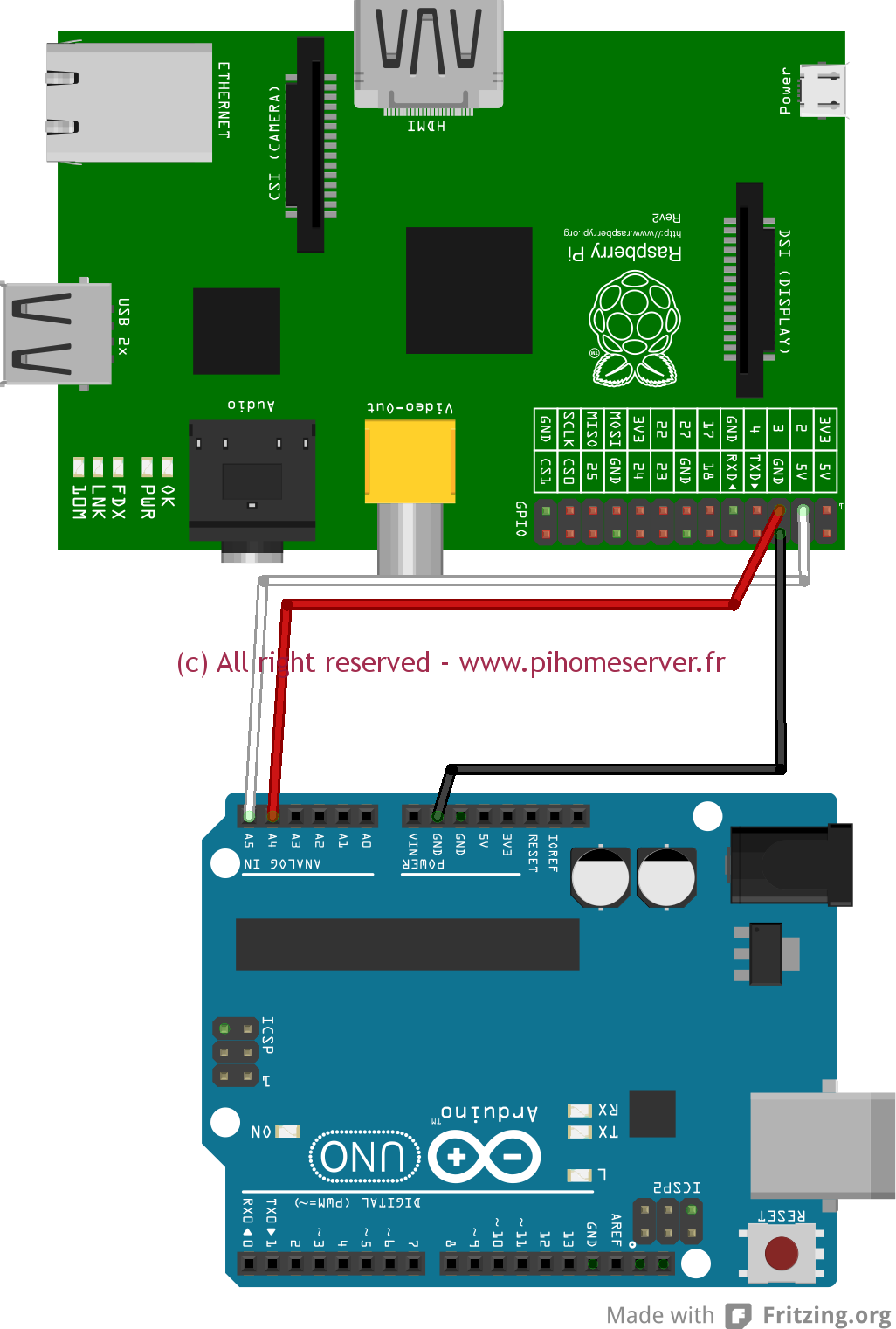
i2cdetect -y 0

Voilà. Etape suivante : connecter l’Arduino et lui parler !

## Communiquer avec l’Arduino

Vous pouvez maintenant relier l’Arduino au Raspberry et allumer tout ce petit monde !

**Attention, sur le schema, le raccordement SDA et SCL est inversé ! Le rouge en A5 et le balnc va en A4 ;-)**

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/ArduinoRaspberry.png)

Une fois fait, vous vous connectez sur le Raspberry Pi et vous relancez la commande i2cdetect avec le bon paramètre :

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/1._ssh-21.png)

Le paramètre défini dans le sketch envoyé à l’Arduino, 0x12, apparait dans les périphériques détectés. On progresse !

Il nous faut maintenant mettre en place le script sur le Raspberry Pi qui va envoyer/recevoir avec l’Arduino. Pour cela, le plus simple c’est d’installer le support d’I2C dans les scripts Python. C’est ce que va faire python-smbus

apt-get install python-smbus

Nous sommes donc maintenant équipé pour parler à notre Arduino. Pour ça un petit programme tout simple. Envoyer un chiffre entre 0 et 8 et nous retourner ce chiffre plus un :

import smbus

import time

# Remplacer 0 par 1 si nouveau Raspberry

bus = smbus.SMBus(0)

address = 0x12

print "Envoi de la valeur 3"

bus.write\_byte(address, 3)

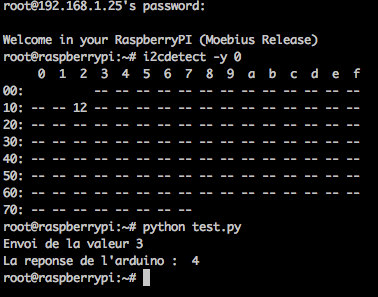
# Pause de 1 seconde pour laisser le temps au traitement de se faire

time.sleep(1)

reponse = bus.read\_byte(address)

print "La reponse de l'arduino : ", reponse

Le résultat de l’exécution :

[](http://www.pihomeserver.fr/wp-content/uploads/2013/08/1._ssh-22.png)

Laisser parler votre imagination pour maintenant diriger l’Arduino en lui envoyant des demandes un peu plus complexes en fonction de ce que vous aurez de branché dessus !

Source et inspiration : un article d’[Oscar](http://blog.oscarliang.net/raspberry-pi-arduino-connected-i2c/)

# <http://www.pobot.org/Tutoriel-complet-sur-I2C.html?lang=fr>

## Tutoriel complet sur I2C

dimanche 12 août 2012, par [Julien Holtzer](http://www.pobot.org/_Julien-Holtzer_.html)

Nous utilisons le **protocole de communication par bus série I2C** depuis longtemps : beaucoup de micro-contrôleurs et même de microprocesseurs utilisent cette technologie.

Il était temps de faire un article complet, en se concentrant sur la **détection du périphérique**, la lecture et l’écriture par la liaison I2C plutôt que sur le composant en lui-même. Consultez les mots-clés en colonne de droite pour avoir accès à une liste d’articles traitant du sujet.

Pour qu’il soit complet, nous allons utiliser un maximum de contrôleurs et de langages différents :  
- PIC  
- AVR  
- Arduino   
- mBed  
- Raspberry Pi  
- un PC Linux

Il y a même **une astuce pour utiliser le port écran (VGA, ...) comme adaptateur I2C**.

### Le matériel

- une carte avec un contrôleur programmable (l’un de ceux cités dessus)  
- les câbles et extensions pour les programmer  
- un composant I2C  
- les câbles pour relier la carte au composant

Plus précisément, nous allons d’abord utiliser un extenseur de port PCF8575 sur une carte de support simple "breakout" de Sparkfun. On utilisera également une mémoire EEPROM 24LC16B qui passait par là.

### Code Arduino

On branche SCL sur ANALOG 5 et SDA sur ANALOG 4.  
Puis le Vcc sur 5 ou 3.3 volts et Gnd sur... Gnd.

Pour commencer, on va "scanner" le bus I2C avec un petit programme pour Arduino. Téléchargez l’archive ci-dessous

[Zip - 799 octets](http://www.pobot.org/IMG/zip/Scanning_I2C_bus.pde.zip)

**Sketch Arduino "scan I2C"**

Envoyer le programme dans votre carte Arduino et ouvrez la console série. Le message suivant apparait :

Scanning...  
I2C device found at address 0x20 !  
done

Attention à l’adresse qui est en hexadécimal. Il faudra soit conserver l’écriture 0x dans votre code, soit faire la conversion en décimal. Ainsi 0x20 devient 32.

On peut maintenant écrire un code pour lire le composant I2C :

#include

<Wire.h>

// Test de l’I2C expander PCF8575 sur breakout Sparkfun

void setup()  
 *Wire.begin() ;  
Serial.begin(9600) ;*

void loop()  
 *Wire.requestFrom(32, 2) ;   
while(Wire.available())   
  
char c = Wire.read() ;   
Serial.print(c & 0xFF, BIN) ; // astuce pour n’afficher que le premier octet, et en binaire pour voir quelle patte est activée ou pas  
Serial.print(", ") ;   
  
Serial.println() ;  
delay(200) ;*

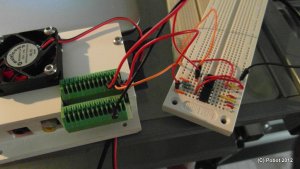
Et voici ce qu’on obtient en touchant les 16 pattes de l’I2C extender :

11111111, 11111101,   
11111111, 11111101,   
11111111, 11111011,   
11111111, 11111011,   
11111111, 11110111,   
11111111, 11110111,   
11111111, 11111111,   
11111111, 11111111,   
11111111, 11111111,   
11111111, 11111111,   
1111111, 11111111,   
10111111, 11111111,   
11011111, 11111111,   
11110111, 11111111,   
11111101, 11111111,   
11111111, 11111111,

### Commandes Linux / Raspberry Pi

Nous utilisons ici [la distribution Occidentalis](http://www.pobot.org/La-distribution-Occidentalis.html) avec le support d’I2C.

Le montage est réalisé sur une plaque d’essai (*breadboard*) avec une mémoire EEPROM I2C de référence 24LC16B comportant 8 blocs de 2 ko. On conserve également l’extenseur de port. La Raspberry Pi est dans son boîtier PVC et on utilise notre [connecteur rapide pour GPIO](http://www.pobot.org/spip.php?page=article&id_article=1025) pour relier le bus I2C à la plaque Labdec.

[](http://www.pobot.org/IMG/jpg/test_I2C_raspberrypi.jpg)

Il y a plusieurs bus I2C sur la RPi. Il faut utiliser le 0 pour accéder à celui qui est sur le connecteur GPIO, donc celui qui est accessible via des picots au pas standard de 2,54 mm. Note : il y a des Raspberry Pi (testée en décembre 2012) qui ont l’I2C du port GPIO sur le numéro 1.

pi@raspberrypi ~ $ sudo i2cdetect 0  
WARNING! This program can confuse your I2C bus, cause data loss and worse!  
I will probe file /dev/i2c-0.  
I will probe address range 0x03-0x77.  
Continue? [Y/n] Y  
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:          -- -- -- -- -- -- -- -- -- -- -- -- --   
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
50: 50 51 52 53 54 55 56 57 -- -- -- -- -- -- -- --   
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
70: -- -- -- -- -- -- -- --                           
pi@raspberrypi ~ $ sudo i2cget 0 0x20  
WARNING! This program can confuse your I2C bus, cause data loss and worse!  
I will read from device file /dev/i2c-0, chip address 0x20, current data  
address, using read byte.  
Continue? [Y/n] y  
0xef

Notez la syntaxe : d’abord le numéro du bus (0) puis l’adresse en hexadécimal avec 0x devant.

Le retour 0xef indique qu’il y a une patte de l’extenseur à 0.

L’écriture est aussi simple. N’hésitez pas à nous contacter par mail pour obtenir plus d’explications et pour que l’on mette en ligne d’autres tests.

### Astuce pour PC Linux

La gestion de l’I2C sous Linux a déjà été en partie abordée avec la section précédente sur Raspberry Pi.

Comme les distributions standard pour PC ne sont pas dédiées à l’électronique, il faut installer les programmes de gestion de l’I2C :

atelier@ubuntu:~$ sudo apt-get install i2c-tools  
atelier@ubuntu:~$ i2c<TAB>  
i2cdetect  i2cdump    i2cget     i2cset    
atelier@ubuntu:~$ i2cdetect -l  
atelier@ubuntu:~$

Rien n’est détecté ! Il faut insérer le module

atelier@ubuntu:~$ sudo modprobe i2c-dev  
atelier@ubuntu:~$ lsmod  
...  
atelier@ubuntu:~$ i2cdetect -l  
i2c-0        unknown           SMBus nForce2 adapter at 3040           N/A  
i2c-1        unknown           SMBus nForce2 adapter at 3000           N/A  
i2c-2        unknown           NVIDIA i2c adapter 0 at 2:00.0          N/A  
i2c-3        unknown           NVIDIA i2c adapter 2 at 2:00.0          N/A  
i2c-4        unknown           NVIDIA i2c adapter 3 at 2:00.0          N/A  
atelier@ubuntu:~$

Oh oh, 5 bus I2C sur le portable de l’association. Pour le "SMBus", pas de surprise, il s’agit du bus de gestion des informations entre périphériques (system management). Mais pour les trois suivants, c’est intéressant.

Voyons ce qui est connecté :

atelier@ubuntu:~$ sudo i2cdetect -y 1  
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:          -- -- -- -- -- 08 -- -- -- -- -- -- --   
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
30: 30 31 -- -- -- -- -- -- -- -- -- -- -- -- -- --   
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --   
50: 50 51 -- -- -- -- -- -- -- -- -- -- -- -- -- --   
60: -- 61 -- -- -- -- -- -- -- -- -- -- -- -- -- --   
70: 70 -- -- -- -- -- -- --                           
atelier@ubuntu:~$

Pas mal, il faudra trouver ce que ça peut être. Pour lire tout le contenu d’un device, plutôt que de faire un i2cget à chacun des registres, on peut utiliser i2cdump. Remarquez le paramètre -y pour éviter de devoir confirmer qu’on veut faire une lecture

atelier@ubuntu:~$ sudo i2cdump -y 1 0x50  
00: 80 08 08 0e 0a 61 40 00 05 30 45 00 82 08 00 00    ?????a@.?0E.??..  
10: 0c 08 38 01 04 00 07 3d 50 50 60 3c 1e 3c 2d 01    ??8??.?=PP`<?<-?  
20: 20 27 10 17 3c 1e 1e 00 06 3c 7f 80 18 22 00 00     '??<??.?<???"..  
30: 00 03 00 00 00 00 00 00 00 00 00 00 00 00 12 1c    .?............??  
40: ce 00 00 00 00 00 00 00 02 4d 34 20 37 30 54 35    ?.......?M4 70T5  
50: 36 36 33 52 5a 33 2d 43 45 36 20 33 52 08 36 82    663RZ3-CE6 3R?6?  
60: 76 24 ff 00 53 57 4c 31 33 30 33 00 00 00 00 00    v$..SWL1303.....  
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................  
80: 00 62 43 33 73 20 35 55 fe e0 58 50 00 19 11 00    .bC3s 5U??XP.??.  
90: 00 62 43 33 73 20 35 55 e0 20 00 00 01 40 06 00    .bC3s 5U? ..?@?.  
a0: 00 62 43 65 36 00 35 55 fe e0 58 50 00 19 11 00    .bCe6.5U??XP.??.  
b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................  
c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................  
d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................  
e0: 45 44 4d ff ff ff ff ff ff ff ff ff ff ff ff ff    EDM.............  
f0: 01 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff    ?...............

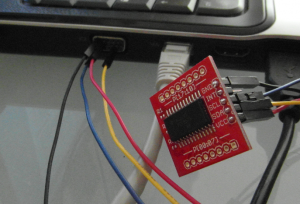
Testons un autre bus :

atelier@ubuntu:~$ sudo i2cdump -y 3 0x50  
No size specified (using byte-data access)  
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef  
00: 00 ff ff ff ff ff ff 00 30 e4 c2 01 00 00 00 00    ........0???....  
10: 00 12 01 03 80 22 13 78 0a e8 95 9b 56 48 96 26    .????"?x????VH?&  
20: 1a 50 54 00 00 00 01 01 01 01 01 01 01 01 01 01    ?PT...??????????  
30: 01 01 01 01 01 01 3e 1c 56 a0 50 00 16 30 30 20    ??????>?V?P.?00   
40: 35 00 58 c2 10 00 00 19 00 00 00 00 00 00 00 00    5.X??..?........  
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 fe 00 4c    .............?.L  
60: 47 20 44 69 73 70 6c 61 79 0a 20 20 00 00 00 fc    G Display?  ...?  
70: 00 4c 50 31 35 36 57 48 31 2d 54 4c 41 33 00 de    .LP156WH1-TLA3.?  
80: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................  
90: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................

Oh oh... "LG display", "LP156WH1-TLA3" : ce sont des indications sur l’écran du portable.

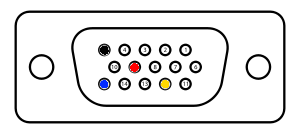
La recherche des bus i2c 1 et 3 et surtout de l’adresse 0x50 (ou 80 en numérique) n’est pas anodine. Elle nous a été soufflée par **Keuronde, notre ami de l’équipe Poivron** qui rédige [des articles très intéressants](http://poivron-robotique.fr/static-i2clinux).

Mais si l’I2C permet de détecter un écran , est-il vrai que **la sortie VGA du portable peut servir d’adaptateur I2C** à peu de frais ? C’était un challenge, et nous sommes fiers de l’avoir démontré.

[](http://www.pobot.org/IMG/png/i2c_VGA.png)

En regardant la prise femelle côté PC, les numéros se lisent de droite à gauche, de haut en bas. La patte 1 est donc en haut à droite et la patte 15 en bas à gauche.

Les signaux I2C sont sur la patte 12 (SDA) et la patte 15 (SCL).  
La masse est sur la patte 5 et la tension d’alimentation 5 volts est sur la patte 9.

[](http://www.pobot.org/IMG/png/VGA_DB15_I2C.png)

Et ça marche : voici la détection avant connection :

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

Et une fois le composant I2C branché par 4 simples fils dans la prise bleu VGA :

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

Allez, une lecture de l’I2C expander PCF8575 à la santé de Samuel :

atelier@ubuntu:~$ sudo i2cget -y 2 0x20 2  
0xfe