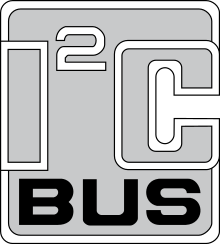
Notes i2c

Notes à propos du bus I2C



# Définition

<https://fr.wikipedia.org/wiki/I2C>

I2C est un bus [série](https://fr.wikipedia.org/wiki/Communication_s%C3%A9rie) [synchrone](https://fr.wikipedia.org/wiki/Synchrone) [bidirectionnel half-duplex](https://fr.wikipedia.org/wiki/Duplex_(canal_de_communication)), où plusieurs équipements, maîtres ou esclaves, peuvent être connectés au bus.

Les échanges ont toujours lieu entre un seul maître et un (ou tous les) esclave(s), toujours à l'initiative du maître (jamais de maître à maître ou d'esclave à esclave). Cependant, rien n'empêche un composant de passer du statut de maître à esclave et réciproquement.

## Topologie

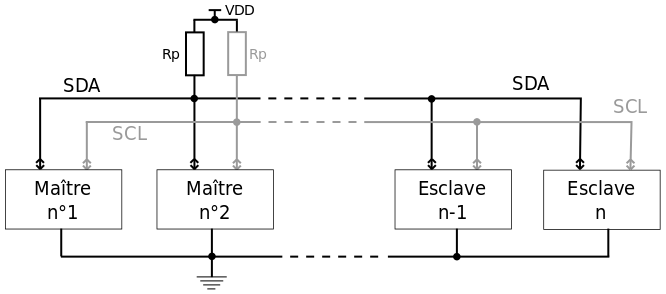
La connexion est réalisée par l'intermédiaire de deux lignes :

* SDA (Serial Data Line) : ligne de données bidirectionnelle,
* SCL (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

Il ne faut également pas oublier la masse qui doit être commune aux équipements.

Les 2 lignes sont tirées au niveau de tension VDD à travers des résistances de [pull-up](https://fr.wikipedia.org/wiki/R%C3%A9sistance_de_rappel) (RP).

Le nombre maximal d'équipements est limité par le nombre d'adresses disponibles, 7 bits d'adressage et un bit R/W (lecture ou écriture), soit 128 périphériques, mais il dépend également de la [capacité](https://fr.wikipedia.org/wiki/Capacit%C3%A9_%C3%A9lectrique) (CB) du bus (dont dépend la vitesse maximale du bus). Il faut savoir que des adresses sont réservées pour diffuser des messages en [broadcast](https://fr.wikipedia.org/wiki/Broadcast_(informatique)) et que de nombreuses adresses sont déjà attribuées par les fabricants ce qui limite grandement le nombre d'équipements (une variante d'adressage sur 10 bits existe également).

[](https://commons.wikimedia.org/wiki/File:I2C_Architecture_2.svg?uselang=fr)

En mode "Ultra-fast mode" (UFm), le bus est unidirectionnel, il ne peut donc y avoir qu'un seul maître. Les 2 lignes sont renommées USCL (ligne d'horloge) et USDA (ligne de données), et côté maître, elles sont toujours en sortie et de type [push-pull](https://fr.wikipedia.org/wiki/Sortie_totem-pole). Ce mode a un usage limité : seules les écritures sont possibles car dans ce mode le fil de donnée (SDA) n'est pas bidirectionnel.

## Calcul des résistances RP

Les [temps](https://fr.wikipedia.org/wiki/I2C#Temps_et_vitesses) et les [niveaux de tension](https://fr.wikipedia.org/wiki/I2C#Niveaux_électriques) dépendent de la capacité du bus (CB) et de la valeur des résistances de pull-up (RP).

Il est difficile de modifier la valeur de la capacité du bus, mais on peut choisir la valeur des résistances pull-up.

#### RPmin

La valeur minimale des résistances de pull-up est limitée par le courant des sorties SDA et SCL (IOL) lorsqu'elles sont à l'état LOW (VOL) :

|  |  |  |  |
| --- | --- | --- | --- |
| Mode | VOLmax | IOL | RPmin pour VDD=5V |
| Standard | 0,4 V | 3 mA | 1 534 Ω |
| Fast | 0,6 V | 6 mA | 733 Ω |
| Fast plus | 0,4 V | 20 mA | 230 Ω |

Pour les deux vitesses supérieures, les valeurs de IOL ne sont pas fournies.

#### RPmax

La valeur maximale de RP est limitée par les temps de montée et de descente.

La variation du signal est donnée par la formule :

|  |  |  |  |
| --- | --- | --- | --- |
| Mode | tr | CB | RPmax |
| Standard | 1 μs | 400 pF | 2 950 Ω |
| Fast | 300 ns | 400 pF | 885 Ω |
| Fast plus | 120 ns | 550 pF | 257 Ω |

# Utilisation du bus I2C sur RaspberryPi

Dans la suite de l’article je présente l’activation puis l’utilisation du bus I2C sur la carte RaspberryPI (ou tout autre carte qui supporte un bus I2C : Olimex A13 ou iMX233, Beagle/PandaBoard de TI, TQ6410, etc…). L’interrogation des esclaves connectés au bus I2C peut se faire sans aucune programmation ! Cela permet de valider rapidement l’écriture ou la lecture des registres d’un esclave I2C. On peut alors passer à la programmation en C/C++ avec des valeurs validées.

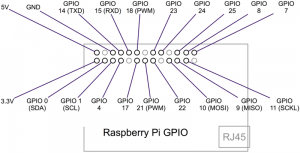
# Ressources internet

Une fois n’est pas coutume, je commence par les ressources Internet :

* L’article Wikipédia sur l’I2C : [article wikipédia](http://fr.wikipedia.org/wiki/I2C)
* Le site de LadyADA : [Adafruit Learning System](http://learn.adafruit.com/)
* Le site eLinux (partie sur l’I2C) : [eLinux](http://elinux.org/Interfacing_with_I2C_Devices)
* les pages de manuel de l’I2C : i2cdetect, i2cset, i2cget

# Brochage du connecteur

Vous trouverez sur Internet de nombreux exemples du brochage du connecteur du RaspberryPI. J’en ai choisi deux parmi d’autres:

[](http://innovelectronique.fr/wp-content/uploads/2013/02/gpios.png)[](http://innovelectronique.fr/wp-content/uploads/2013/02/gpio-srm.png)

Les broches qui nous intéressent ici (pour la V2 revB) :

* GPIO0 (SDA) : broche 3 du connecteur,
* GPIO1 (SCL) : broche 5 du connecteur,
* la masse (GROUND) : broche 6 du connecteur,
* l’alimentation (on prendra des esclaves I2C qui fonctionnent en 3,3V) : broche 1 du connecteur

# Accès aux périphériques du Raspberry Pi - Le bus i2c

Marc Silanus version 1.0, 04-11-2017

Les ordinateurs embarqués sous le système d’exploitation Linux sont massivement présents dans les technologies modernes (transports, multimédia, téléphonie mobile, appareils photos …).  
L’ordinateur Raspberry PI constitue un support d’apprentissage performant, très bon marché et disposant d’une forte communauté sur le net. Il possède des entrées/sorties puissantes permettant une connexion avec le monde physique par l’intermédiaire de capteurs et d’actionneurs.  
L’objectif de ce TP est de réaliser une rapide prise en main d’un système embarqué au travers d’un ordinateur Raspberry PI qui sera largement utilisée en projet et d’effectuer un tour d’horizon des pratiques de mise en œuvre et de développement.

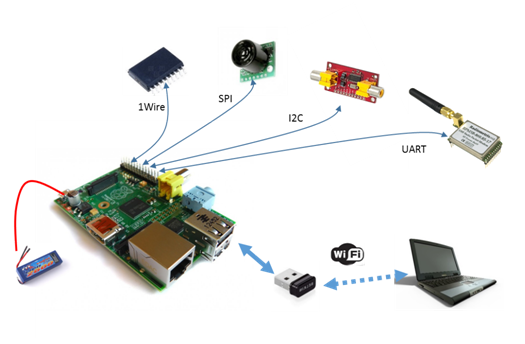


Figure 1. Ecosystème Raspberry Pi

# Présentation de l’activité

## Objectifs

**Se remémorer :**

* les bases de la programmation objet en C++ :
  + Déclaration/Implémentation d’une classe
  + Instanciation d’une classe (→ constructeur)
  + Associations entre classes
* les bases du formalisme UML
* les principes de la cross-compilation

**Apprendre :**

* les principes de la cross-compilation
* la manipulation du bus i2c du Raspberry pi

**Mettre en oeuvre :**

* un afficheur LCD RGB i2c
* ? un capteur de température i2c ?

**Durée**

* Entre 8h et 10h

## Ressources

**Matériel(s) :**

* 1 PC avec système d’exploitation Linux
* 1 Raspberry Pi avec OS Débian
* 1 afficheur LCD RGB i2c JHD1313M1 de Grove
* ? 1 capteur de température i2c ?

**Logiciel(s) :**

* cross-compilateur arm-linux-gnueabihf-g++

**Documentation :**

* Cours de Mr Antoine sur la programmation objet de 1ière année (disponible sur Chamilo).
* Sites de référence sur C/C++ :
  + cplusplus.com
  + cppreference.com
  + …
* [datasheet afficheur LCD JHD\*](https://seeeddoc.github.io/Grove-LCD_RGB_Backlight/res/JHD1214Y_YG_1.0.pdf#page=14)
* [datasheet contrôleur RGB PCA9633](https://seeeddoc.github.io/Grove-LCD_RGB_Backlight/res/PCA9633.pdf#G3231034)
* [Documentation du module i2c de la libraire BCM2835](http://www.airspayce.com/mikem/bcm2835/group__i2c.html)

**Pré-requis**

* Notions sur le formalisme UML (représentation d’une classe et des relations)
* Maîtrise du langage C (séquences, tests, boucles, variables, fonctions, paramètres, compilation multi-fichiers)
* Maîtrise de la gestion des fichiers sur un système déporté (ssh, scp, Dolphin, …​)
* Maîtrise de la cross-compilation pour processeurs à architecture ARM

**Compte rendu**

Il sera constitué :

* des réponses aux questions en veillant à soigner la rédaction
* des programmes source commentés

## 1. Présentation

Le Raspberry Pi dispose de deux interfaces i2c. Le bus numéro 0 qui était accessible à travers le connecteur P5 sur le modèle B a disparu depuis le modèle B+. De plus, ce connecteur étant présent sous forme de simples trous cuivrés, il était donc nécessaire de venir y souder des broches pour pouvoir l’utiliser. Les signaux du bus 0 sont toujours présents – mais pas très accessibles – sur les connecteurs J3 (camera) et J4 (display).

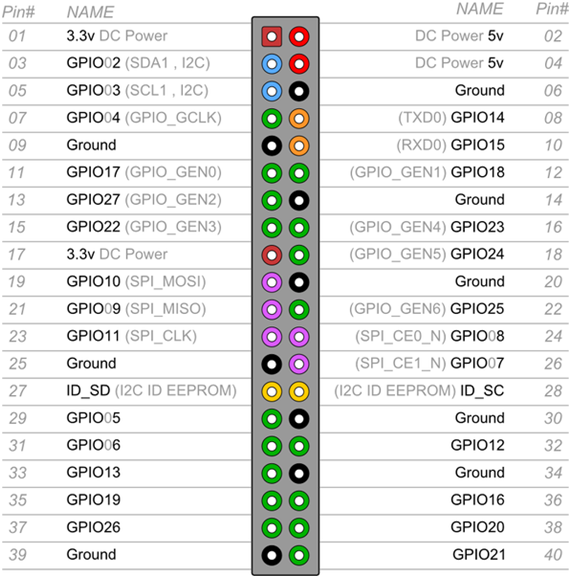


Figure 2. Affectation des broches

## 2. Le bus I2C sur le Raspberry pi

### 2.1 L’accès au bus I2C

Nous allons nous intéresser au second bus, accessible via le port d’extension P1, sur deux broches (que l’on peut également employer pour des entrées/sorties GPIO) identiques quel que soit le modèle de Raspberry. Il s’agit de la broche 3 (signal SDA) et de la broche 5 (signal SCL).

#### ****Exercice 1****

Rappelez à quoi servent ses deux signaux.

### 2.1 Activation du bus I2C sur le Raspberry

Le protocole i2c est supporté par le noyau Linux depuis sa version 2.4. De nombreux périphériques sont reconnus par le kernel, notamment dans le sous-système Hwmon (Hardware Monitor).

L’accès depuis l’espace utilisateur est facilité par le module i2c-dev qui rend les bus i2c visibles dans le répertoire /dev sous forme de fichiers spéciaux représentant des périphériques en mode caractère.

#### ****Exercice 2****

* Activez la prise en charge du bus I2C sur le Raspberry :

$ sudo raspi-config

→ Advanced options → I2C → Enabled (tout accepter).

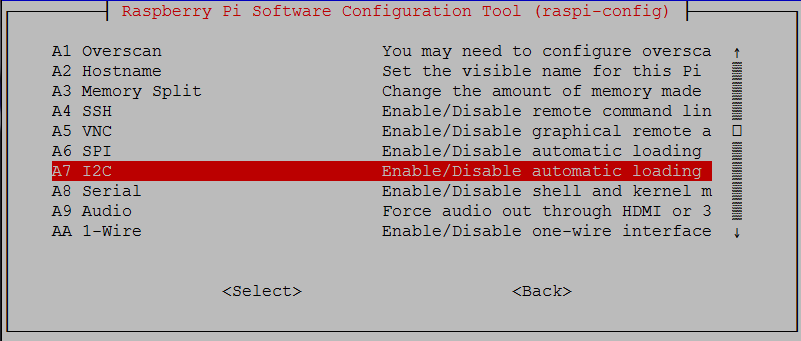


Figure 3. Raspi-config

OU

* Supprimez du fichier /etc/modprobe.d/raspi-blacklist.conf les référence à l’I2C
* Ajoutez i2c aux modules à charger

$ echo i2c-dev >> /etc/modules

|  |  |
| --- | --- |
|  | Normalement, les modules sont déja installés mais blacklistés. Cependant, dans certains cas, il peut être nécessaire de les installer manuellement : i2c\_bcm2708 et i2c-dev |

* Vérifiez la prise charge du bus I2C, identifiez la vitesse de transmission sur le bus :

$ dmesg | grep i2c

* Identifiez le(s) bus i2c-c disponible(nt) :

$ ls /dev/i2c\*

### 2.1 Installation des outils de gestion du bus i2c

#### ****Exercice 3****

* Installez les outils de gestion du bus i2c

$ sudo apt-get install i2c-tools

* Détectez des esclaves sur le bus :

$ i2cdetect –y 1

|  |  |
| --- | --- |
|  | -y pour répondre automatiquement yes et 1 pour le numéro de périphérique i2c |

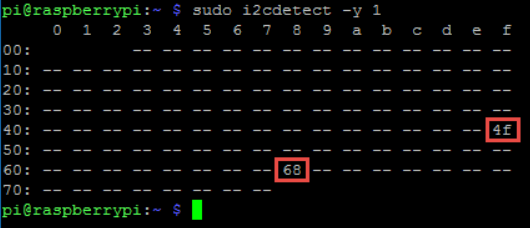


Figure 4. Affichage des périphériques i2c

|  |  |
| --- | --- |
|  | Les commandes i2cset et i2cget permettent respectivement d’écrire et de lire sur le bus i2c. |

## 3. Gestion d’un afficheur LCD RGB i2c

L’afficheur à gérer est un JHD1313M1 de chez Grove. Il est en fait composé d’un afficheur LCD basé sur le HD44780 d’Hitachi et d’un contrôleur du rétro-éclairage (RGB) PCA9633 de NXP Semiconductors. Il possède donc 2 adresses i2c :

* LCD : 0x3e
* RGB : 0x62

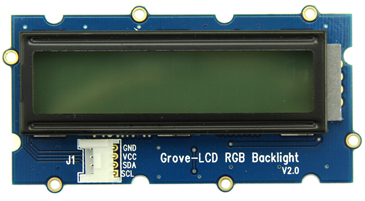


Figure 5. Afficheur LCD RGB i2c

### 3.1 Par l’invite de commandes

#### ****Exercice 4****

* Connectez l’afficheur au connecteur de la carte Raspberry pi.

|  |  |
| --- | --- |
|  | l’afficheur nécessite une tension de 5V pour fonctionner correctement. |

* Vérifiez la présence de l’afficheur sur le bus i2c.

|  |  |
| --- | --- |
|  | La commande i2cset s’utilise de la manière suivante :  i2cset [-f] [-y] [-m mask] [-r] i2cbus chip-address data-address [value] ... [mode] |

* Procédez à la séquence d’initialisation de l’afficheur :

i2cset -y 1 0x3E 0x80 0x3C

i2cset -y 1 0x3E 0x80 0x0C

i2cset -y 1 0x3E 0x80 0x01

i2cset -y 1 0x3E 0x80 0x06

i2cset -y 1 0x62 0x00 0x00

i2cset -y 1 0x62 0x08 0xFF

i2cset -y 1 0x62 0x01 0x20

i2cset -y 1 0x62 0x04 0x00

i2cset -y 1 0x62 0x03 0x00

i2cset -y 1 0x62 0x02 0xFF

i2cset -y 1 0x3E 0x40 0x31

i2cset -y 1 0x3E 0x40 0x32

i2cset -y 1 0x3E 0x40 0x33

* Identifiez les instructions passées à l’afficheur LCD (adresse i2c 0x3E) et au contrôleur de rétro-éclairage (adresse i2c 0x62):
  + [datasheet afficheur LCD JHD\*](https://seeeddoc.github.io/Grove-LCD_RGB_Backlight/res/JHD1214Y_YG_1.0.pdf#page=14)
  + [datasheet contrôleur RGB PCA9633](https://seeeddoc.github.io/Grove-LCD_RGB_Backlight/res/PCA9633.pdf#G3231034)
* Identifiez les registres de gestions des couleurs du contrôleur de rétro-éclairage.
* Configurez le rétro-éclairage pour qu’il clignote à une fréquence de 1Hz avec un rapport cyclique de 50%.

### 3.2 Visualisation d’une trame i2c

|  |  |
| --- | --- |
|  | Le bus i2c (Inter Integrated Circuit) fait partie des bus série : 3 fils pour faire tout passer. Il a été développé au début des années 1980, par Philips pour minimiser les liaisons entre les circuits intégrés numériques de ses produits (Téléviseurs, éléments HiFi, magnétoscopes, …​). |

Le bus i2c

Le bus i2c permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils : un signal de données (SDA), un signal d’horloge (SCL), et un signal de référence électrique (masse).

Il s’agit d’une liaison en mode série, ce qui signifie que la vitesse de transfert sera plus faible qu’avec un bus de type parallèle. Le bus i2c permet cependant des échanges à la vitesse de 100 kbits par seconde. Certes, la vitesse de transfert du bus i2c n’est pas fulgurante, mais dans bien des cas, la vitesse n’est pas l’élément prédominant.

L’utilisation d’un bus i2c permet de réduire la complexité des circuits imprimés à réaliser. Par exemple, pour connecter une EEPROM ou une RAM à un microcontrôleur classique, il faut relier entre eux les bits de données et les bits d’adresses des différents composants, et, en plus, il faut bien souvent ajouter une logique de sélection. Avec des composants prévus pour se connecter au bus i2c, il suffit de les relier par deux pistes seulement. Si, par la suite, on souhaite ajouter des composants sur le circuit, le nombre de pistes à ajouter sera vraiment plus réduit (essayez d’ajouter une EEPROM sur un circuit existant pour voir).

De nombreux fabricants ayant adopté le système la variété des systèmes disponibles disposant d’un port i2c est énorme :

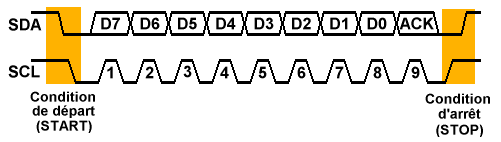
* microcontrôleurs
* expandeurs de bus (entrée/sortie 8 bits)
* convertisseurs A/N et N/A
* mémoires (RAM, EPROM, EEPROM, etc.)
* récepteurs infra-rouge (télécommande RC5)
* capteurs de température
* circuits audio (égaliseur, contrôle de volume, etc.)
* drivers d’affichage LCD ou à LEDs
* décodeurs télétexte
* chargeurs de batterie
* PLL pour tuner HF
* etc.

Le protocole i2c

Le protocole du bus I²C définit la succession des états possibles sur les lignes SDA et SCL, et comment doivent réagir les circuits en cas de conflit.

La prise de contrôle du bus

Pour transmettre des données sur le bus I²C, il faut surveiller deux conditions particulières : la condition de départ et la condition d’arrêt.

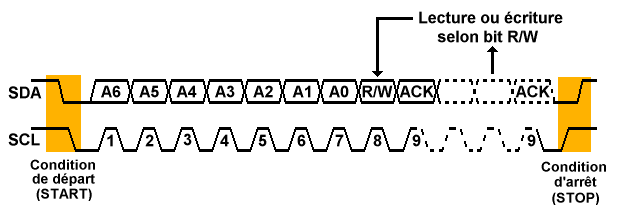


Avant de tenter de prendre le contrôle du bus, un circuit doit vérifier que les lignes SDA et SCL sont au repos, c’est-à-dire à l’état haut. Si c’est le cas, le circuit indique qu’il prend le contrôle du bus en mettant la ligne SDA à 0. A partir de ce moment là, les autres circuits savent que le bus est occupé et ils ne devraient pas tenter d’en prendre contrôle. Le circuit qui vient de prendre le contrôle du bus en devient le maître (en anglais « master »). C’est lui qui génère le signal d’horloge, quel que soit le sens du transfert.

La transmission d’une addresse

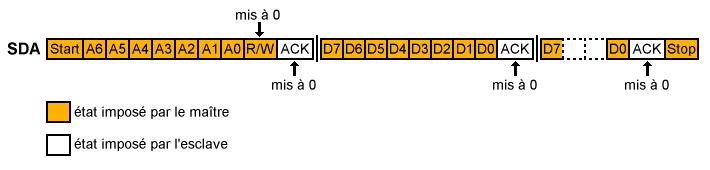
Le nombre de composants qu’il est possible de connecter sur un bus I²C étant largement supérieur à deux, le maître doit pouvoir choisir quel esclave est censé recevoir les données. Dans ce but, le premier octet que transmet le maître n’est pas une donnée mais une adresse. Le format de l’octet d’adresse est un peu particulier puisque le bit D0 est réservé pour indiquer si le maître demande une lecture â l’esclave ou bien au contraire si le maître impose une écriture à l’esclave. Adressage Chaque circuit connecté au bus I²C possède une adresse, qui doit être unique.

Une fois l’adresse envoyée sur le bus, l’esclave concerné doit répondre en plaçant le bit ACK à 0. Si le bit ACK vaut 1, le maître comprend qu’il y a une erreur de sélection et il génère la condition arrêt. En revanche, si le bit ACK vaut 0, le maître peut continuer les opérations.



Écriture d’une donnée

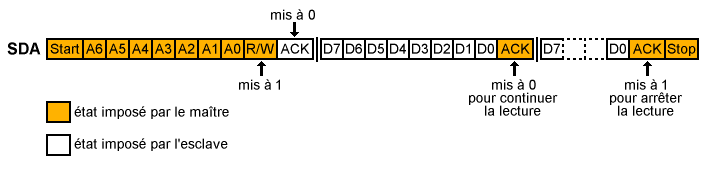
Si le bit R/W précédemment envoyé était à 0, cela signifie que le maître doit transmettre un ou plusieurs octets de données. Après chaque bit ACK valide, le maître peut continuer d’envoyer des octets à l’esclave ou bien il peut décider de terminer le dialogue par une condition d’arrêt.



Lecture d’une donnée

Si le bit R/W transmis en même temps que l’adresse est à 1, cela signifie que le maître veut lire des données issues de l’esclave. C’est toujours le maître qui va générer le signal d’horloge SCL. En revanche, après le bit ACK de l’adresse, c’est l’esclave qui va garder le contrôle de la ligne SDA. Pour cela, le maître va placer sa propre sortie SDA au niveau haut pour permettre à l’esclave de prendre le contrôle de la ligne SDA. L’esclave doit alors scruter la ligne SCL et attendre le niveau bas pour changer l’état de la ligne SDA, faute de quoi le maître détectera une condition arrêt et abandonnera le transfert (l’électronique intégrée dans l’esclave se doit de détecter aussi qu’il y a eu une condition arrêt, bien entendu).

Après que l’esclave a transmis les 8 bits de données, c’est le maître, cette fois-ci, qui va générer un bit d’acquittement. Si le maître désire lire des octets supplémentaires, il placera le bit d’acquittement à 0. En revanche, si le maître décide que la lecture est terminée, il placera le bit ACK au niveau 1. L’esclave comprendra alors que le transfert est terminé. Cette fois-ci, bien que le bit ACK soit au niveau 1, cela ne correspond pas à une condition d’erreur mais à une fin de transfert.



Référence web :

— https://www.aurel32.net/elec/i2c.php

#### ****Exercice 4****

* A tour de rôle, vous vous rendrez sur le poste dédié à l’utilisation de l’analyseur de protocole Saelae
* Exécutez le logiciel Logic qui permet de piloter l’analyseur Saelae
* Paramétrez l’analyseur logique, connectez 2 de ses sondes aux broches SDA et SCL du Raspberry Pi et lancez une acquisition avec analyse du protocole i2c.
* Envoyez la commande i2cset -y 1 0x3E 0x40 0x31
* Analysez les trames obtenues et vérifiez qu’elles sont conformes au standard i2c
  + Vitesse de transfert (fréquence de SCL)
  + Condition de départ (START) et condition d’arrêt (STOP)
  + Adresse de l’esclave
  + Type d’opération (lecture/écriture)
  + Adresse du registre visé
  + Valeur de la donnée
  + bits ACK

### 3.3 Programmation C++

Vous allez écrire un programme permettant d’afficher un message sur votre afficheur en vous appuyant sur la librairie rgb\_lcd de gestion de cet afficheur.

Mais il nous faut avant tout pouvoir utiliser le bus i2c. La librairie `BCM2835`que nous avons utilisé dans le précédent TP fournie des fonctions de gestions du bus :

#### ****Exercice 5****

* Codez le programme de test suivant :

testRGB.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Test du rétro-éclairage

\*

\* i2cset -y 1 0x62 0x00 0x00

\* i2cset -y 1 0x62 0x08 0xFF

\* i2cset -y 1 0x62 0x01 0x20

\* i2cset -y 1 0x62 0x04 0x00

\* i2cset -y 1 0x62 0x03 0x00

\* i2cset -y 1 0x62 0x02 0xFF

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <bcm2835.h>

#include <iostream>

using namespace std;

int main()

{

char buf[1];

if (!bcm2835\_init())return 1;

bcm2835\_i2c\_begin(); //Start I2C operations.

cout << "Start I2C operations : ok" << endl;

bcm2835\_i2c\_setSlaveAddress(0x62); //I2C address

cout << "I2C address : ok" << endl;

bcm2835\_i2c\_set\_baudrate(10000); //10k baudrate

cout << "Baudrate : ok" << endl;

buf[0] = 0x00;

buf[1] = 0x00;

bcm2835\_i2c\_write(buf,2);

buf[0] = 0x08;

buf[1] = 0xFF;

bcm2835\_i2c\_write(buf,2);

buf[0] = 0x01;

buf[1] = 0x20;

bcm2835\_i2c\_write(buf,2);

buf[0] = 0x02;

buf[1] = 0xFF;

bcm2835\_i2c\_write(buf,2);

buf[0] = 0x03;

buf[1] = 0xFF;

bcm2835\_i2c\_write(buf,2);

buf[0] = 0x04;

buf[1] = 0;

bcm2835\_i2c\_write(buf,2);

bcm2835\_i2c\_end();

bcm2835\_close();

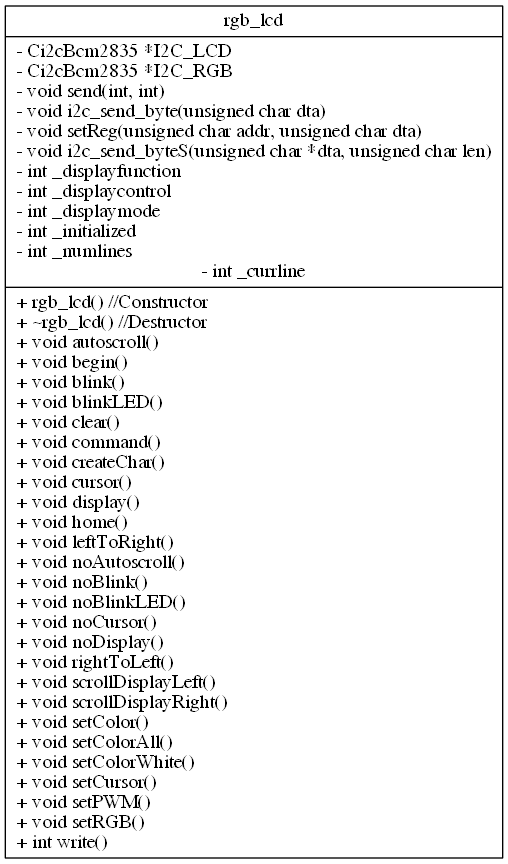
return 0;

}

* Donnez les définitions des fonctions suivantes :
  + bcm2835\_init()
  + bcm2835\_i2c\_begin()
  + bcm2835\_i2c\_setSlaveAddress(0x62)
  + bcm2835\_i2c\_set\_baudrate(10000)
  + bcm2835\_i2c\_write(buf,2)
  + bcm2835\_i2c\_end()
  + bcm2835\_close()

|  |  |
| --- | --- |
|  | Reportez vous à la documentation de la [librairie BCM2835](http://www.airspayce.com/mikem/bcm2835/group__i2c.html) |

La classe [rgb\_lcd](http://silanus.fr/bts/activites/raspberry/i2cBcm2835.zip) permet de gérer l’afficheur et le rétro-éclairage.



#### ****Exercice 6****

* Identifiez la méthode qui réalise l’initialisation de l’afficheur et montrez qu’elle est conforme à la séquence d’initialisation décrite par le fabricant.
* Identifiez la méthode qui permet de gérer la couleur du rétro-éclairage.
* Identifiez la méthode qui permet d’afficher des caractères sur l’afficheur.
* Complétez les définitions du fichier d’entête rgb\_lcd.h en remplaçant les "?" par les bonnes adresses :

// Device I2C Adrress

#define LCD\_ADDRESS ????

#define RGB\_ADDRESS ????

// color define

...

// RGB Registers

#define REG\_RED ????

#define REG\_GREEN ????

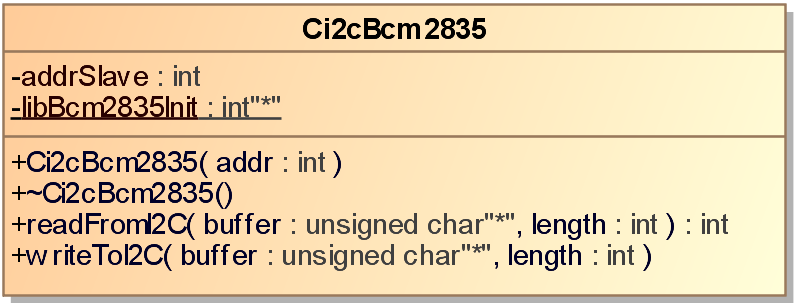
#define REG\_BLUE ????

#define REG\_MODE1 ????

#define REG\_MODE2 ????

#define REG\_OUTPUT ????

* Quels objets de la classe Ci2cBcm2835 permettent l’accès au bus i2c ?
* Codez la classe Ci2cBcm2835 qui définie par le diagramme de classe ci-dessous :



#### ****Exercice 7****

* Codez le programme principal mainLcd.cpp qui répond à l’algorithme suivant :

Début

Instancier un objet de la classe `rgb\_lcd`

Initialiser l'afficheur (16 caratères, 2 lignes)

Configurer le rétro-éclairage en bleu

Placer le curseur au caractère 5 de la première ligne

Ecrire \_"Bonjour"\_

Placer le curseur au caractère 1 de la seconde ligne

Ecrire \_"Afficheur i2c"\_

Attendre 5 secondes

Effacer l'écran

Configurer le rétro-éclairage en blanc

Placer le curseur au caractère 0 de la première ligne

Ecrire \_"OH"\_

Placer le curseur au caractère 0 de la seconde ligne

Ecrire \_"YES !!!"\_

Tant que 1

Décaler l'affichage vers la droite

Attendre 250 ms

Fin tant que

Fin

* proposez une séquence personnelle d’affichage en jouant avec les couleurs et le clignotement du rétro-éclairage, des décalages vers la droite ou la gauche du texte à l’écran.

## 4. Conclusion

Le bus de données i2c permet d’échanger des données numériques entre un capteur (ou un actionneur) et un mini-PC tel que le Raspberry Pi. Le bus i2c est très répandu également sur les micro-contrôleurs (Arduino, ESP8266, ESP32). Par défaut, le bus i2c n’est pas activé sur la distribution Raspbian.

|  |  |
| --- | --- |
|  | [Pensez à activer le bus i2c avant de l’utiliser !](http://silanus.fr/bts/activites/raspberry/i2cRaspberryPi-SN2V1.html#_2_1_activation_du_bus_i2c_sur_le_raspberry) |

Les outils de diagnostic et de gestion en ligne de commandes sont très utiles.

|  |  |
| --- | --- |
|  | [Pensez à installer i2ctools !](http://silanus.fr/bts/activites/raspberry/i2cRaspberryPi-SN2V1.html#_2_1_installation_des_outils_de_gestion_du_bus_i2c) |

La librairie BCM2835 fournie les fonctions nécessaires à la gestion du bus dans un programme C/C++

|  |  |
| --- | --- |
|  | [Pensez à consulter la documentation de la librairie !](http://www.airspayce.com/mikem/bcm2835/group__i2c.html) |

# Utilisation de l’I2C sur le RaspberryPI

<http://innovelectronique.fr/2013/03/02/utilisation-du-bus-i2c-sur-raspberrypi/>

## Activation de l’I2C

Il faut tout d’abord être « root » pour réaliser les commandes suivantes : sudo su. Il faut éditer le fichier « /etc/modprobe.d/raspi-blacklist.conf » avec l’éditeur nano puis commenter (mettre le caractère #) la ligne « blacklist i2c-bcm2708 ».  
Puis rajouter la ligne « i2c-dev » dans le fichier « /etc/modules ». Redémarrez le RaspberryPI avec la commande « reboot ».  
Pour vérifier que le pilote de l’I2C est bien chargé, tapez ces deux commandes:

* dmesg | grep i2c : permet de visualiser les messages du noyau se rapportant au bus i2c,
* ls /dev/i2c\* : cette commande permet de savoir si le bus i2c est bien accessible depuis « l’espace utilisateur » donc depuis vos programmes.

Voici le résultat de ces deux commandes :



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | root@raspberrypi:/home/pi# dmesg|grep i2c  [   32.803506] bcm2708\_i2c bcm2708\_i2c.0: BSC0 Controller at 0x20205000 (irq 79) (baudrate 100k)  [   32.832122] bcm2708\_i2c bcm2708\_i2c.1: BSC1 Controller at 0x20804000 (irq 79) (baudrate 100k)  [   38.616811] i2c /dev entries driver  root@raspberrypi:/home/pi# ls /dev/i2c-  i2c-0  i2c-1 |

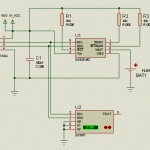
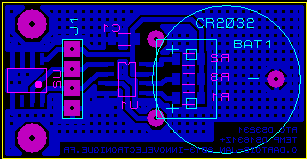
On remarque qu’il y a deux bus i2c reconnus sur la carte. La vitesse de ces bus est ici de 100kHz (mode basse vitesse du bus i2c). Ces deux bus sont accessibles par i2c-0 et i2c-1. En fait un seul bus est disponible sur le connecteur.

## Installation des outils I2C

Pour tester si le bus I2C est fonctionnel on connecte une platine avec deux circuits :

* un capteur de température DS1631,
* une horloge temps réel DS3231.

Schéma électrique de la platine  et typon (échelle 2) :

[](http://innovelectronique.fr/wp-content/uploads/2013/03/schema-electrique-carte-ds1631-ds3231-sur-bus-i2c.png)[](http://innovelectronique.fr/wp-content/uploads/2013/03/typon-carte-ds1631-ds3231-sur-bus-i2c.png)

Il faut ensuite installer les outils nécessaires pour dialoguer sur le bus i2c sans programmer. En tant qu’utilisateur root, tapez la commande suivante:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | root@raspberrypi:/home/pi# aptitude install i2c-tools  Les NOUVEAUX paquets suivants vont être installés :    i2c-tools  0 paquets mis à jour, 1 nouvellement installés, 0 à enlever et 0 non mis à jour.  Il est nécessaire de télécharger 59,5 ko d'archives. Après dépaquetage, 223 ko seront utilisés.  Prendre :  1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main i2c-tools armhf 3.1.0-2 [59,5 kB]  59,5 ko téléchargés en 1s (56,8 ko/s)  Sélection du paquet i2c-tools précédemment désélectionné.  (Lecture de la base de données... 60077 fichiers et répertoires déjà installés.)  Dépaquetage de i2c-tools (à partir de .../i2c-tools\_3.1.0-2\_armhf.deb) ...  Traitement des actions différées (« triggers ») pour « man-db »...  Paramétrage de i2c-tools (3.1.0-2) ... |

L’utilitaire i2cdetect permet de parcourir le bus i2c pour détecter des esclaves reliés à ce bus. Les commandes suivantes permettent de parcourir le bus i2c-0 : raspberry V2revA ou le bus i2c-1 : raspberry V2revB.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | root@raspberrypi:/home/pi# i2cdetect -y 0       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  00:          -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  70: -- -- -- -- -- -- -- --  root@raspberrypi:/home/pi# i2cdetect -y 1       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  00:          -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --  70: -- -- -- -- -- -- -- -- |

On remarque que sur le bus i2c-1 il y a deux esclaves i2c :

* adresse 0x48 : capteur DS1631 dont les lignes d’adresses A2, A1 et A0 sont reliées à la masse. L’adresse de ce composant est donné dans la documentation : 0b1001A2A1A0 donc 0b1001000 donc en hexadécimal 0x48.
* adresse 0x68 : horloge RTC. Dans la documentation on trouve l’adresse suivante : 0b1101000 soit 0x68.

Le résultat du parcours du bus est donc concluant.

## Utilitaires d’interrogation du bus i2c

Nous allons commencer à interroger le capteur de température de l’horloge temps réel. Le résultat de la conversion de température est accessible à l’adresse 0x11 sur deux octets (voir extrait de la documentation ci-dessous ) :

[registre-temp-ds3231](http://innovelectronique.fr/wp-content/uploads/2013/02/registre-temp-ds3231.png)

Vous disposez de deux utilitaires pour cela : i2cset et i2cget. Ci-dessous vous trouverez ce que renvoie ces commandes lorsqu’on les lance sans paramètres :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | root@raspberrypi:/home/pi# i2cset  Usage: i2cset [-f] [-y] [-m MASK] I2CBUS CHIP-ADDRESS DATA-ADDRESS [VALUE] ... [MODE]    I2CBUS is an integer or an I2C bus name    ADDRESS is an integer (0x03 - 0x77)    MODE is one of:      c (byte, no value)      b (byte data, default)      w (word data)      i (I2C block data)      s (SMBus block data)      Append p for SMBus PEC  root@raspberrypi:/home/pi# i2cget  Usage: i2cget [-f] [-y] I2CBUS CHIP-ADDRESS [DATA-ADDRESS [MODE]]    I2CBUS is an integer or an I2C bus name    ADDRESS is an integer (0x03 - 0x77)    MODE is one of:      b (read byte data, default)      w (read word data)      c (write byte/read byte)      Append p for SMBus PEC |

En gros on remarque qu’il faut spécifier :

* le numéro du bus i2c,
* l’adresse de l’esclave,
* la ou les registres à écrire/lire.

Dans l’exemple qui suit, on lit la température fourni par l’horloge temps réél : adresse 0x68, registres 0x11 et lecture d’un mot (2 octets) :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | root@raspberrypi:/home/pi# i2cget 1 0x68 0x11 w  WARNING! This program can confuse your I2C bus, cause data loss and worse!  I will read from device file /dev/i2c-1, chip address 0x68, data address  0x11, using read word data.  Continue? [Y/n] y  0x4013  root@raspberrypi:/home/pi# i2cget -y 1 0x68 0x11 w  0x4013 |

On obtient la valeur hexadécimal 0x4013. La valeur 0x13 correspond au contenu du registre 0x11 (partie entière de la température). La valeur 0x40 correspond à la valeur du registre 0x12 (partie fractionnaire de la température) dont seul les bits B7 B6 nous intéresse. On obtient donc la température suivante : 0x13 = 19 et 0x40 => B7=0 et B6=1. B7 représente 2^-1=0,5°C et B6 représente 2^-2=0,25°C. La température finale est donc de 19,25°C.

Pour initialiser correctement la RTC, il faut écrire la valeur 0x04 dans le registre 0x0E (registre de « control ») et 0x08 dans le registre 0x0F (registre de « status »). Pour la signification de ces valeurs reportez-vous à la documentation officielle. Cela se traduit par les commandes suivantes :



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | root@raspberrypi:/home/pi# i2cset -y 1 0x68 0x0E 0x04 b  root@raspberrypi:/home/pi# i2cget -y 1 0x68 0x0F b  0x88  root@raspberrypi:/home/pi# i2cset -y 1 0x68 0x0F 0x08 b  root@raspberrypi:/home/pi# i2cget -y 1 0x68 0x0F b  0x08 |

Vous pouvez maintenant lire les secondes, minutes, heures, jour du mois, mois et année. Exemple avec la lecture des secondes et de l’année :



|  |  |
| --- | --- |
| 1  2  3  4 | root@raspberrypi:/home/pi# i2cget -y 1 0x68 0x00 b  0x41  root@raspberrypi:/home/pi# i2cget -y 1 0x68 0x06 b  0x00 |

On obtient donc pour les secondes : 0x41 => 4 x 10s + 1 x 1s = 41s. Pour les années 0 ! La RTC n’est donc pas réglé. Je vous invite à chercher par vous même les commandes pour mettre à l’heure votre horloge RTC.

# Utiliser un P82B715PN pour étendre le Bus I2C sous Arduino

<https://arduino103.blogspot.com/2013/03/utiliser-un-p82b715pn-pour-etendre-le.html>

<http://wiki.mchobby.be/index.php?title=P82B715PN:_I2C_Bus_Extender>

Dominique Meurisse (MCHobby) jeudi 14 mars 2013

Nous venons de terminer notre deuxième tutoriel sur I2C pour Arduino.  
Cette fois-ci, nous allons nous attarder sur le **P82B715PN**, un composant qui permet de **booster un Bus I2C jusqu'à 25m**.

|  |
| --- |
| <https://3.bp.blogspot.com/-EbrTFHxbeX4/UUInrn5EMiI/AAAAAAAADJI/_qry6jJWCKQ/s1600/tlogo-I2C-P82B715PN.jpg> |
| [Tutoriel sur l'utilisation du P82B715N avec Arduino](http://mchobby.be/wiki/index.php?title=P82B715PN:_I2C_Bus_Extender) |

## Pourquoi un P82B715PN?

Si vous avez déjà utilisé des composants I2C, vous savez à quel point cette technologie est efficace et simple à mettre en oeuvre. Un bus I2C c'est littéralement "magique" sur un Arduino ou Raspberry Pi.   
Le problème d'un Bus I2C c'est qu'il ne peut pas vraiment faire plus d'un mètre!!!   
Heureusement, il y a les P82B715PN﻿ qui permettent d'étendre le bus sur une distance astronomique de 25m (et plus sous condition). Son utilisation est vraiment simple :-)

|  |
| --- |
| <https://4.bp.blogspot.com/-Su1GW4QhKMo/UUIuBFXaPfI/AAAAAAAADJg/nJAAI97_N6A/s1600/P82B715-sample.jpg> |
| [Le P82B715PN disponible chez MCHobby](http://mchobby.be/PrestaShop/product.php?id_product=234) |

Comme vous le constatez, le P82B715PN est petit et utilisable sur un breadboard :-)

## Comment cela fonctionne t'il?

D'une façon générale, les broches SDA et SLC d'un bus I2C peuvent débiter un courant de l'ordre de 1mA. C'est ce qui limite la distance de transmission. Le Bus Extender permet d'amplifier ce courant jusqu'à 10mA... ce qui augmente la portée du signal.

## Quel câble utiliser?

Le mieux est d'utiliser une paire torsadée pour SLA et SLC. Si possible dans un câble blindé (pour éviter le parasitage). Ce type de câble bon marché est vendu dans de nombreux magasin. C'est le cas du câble de raccordement téléphonique (celui utilisé par les installateur) ou du simple câble réseau (Catégorie UTP5).

## Combien de P82B715PN﻿?

Au minimum 2: 1 pour votre microcontroleur et 1 pour votre périphérique USB. Un P82B715PN﻿ est nécessaire par point de raccordement sur le Bus I2C (il ne faut absolument un P82B715PN﻿ entre chaque périphérique I2C et votre "bus I2C boosté"). 

## Montage

Voici le plan de montage que nous avons utilisé pour tester le Bus Extender... et ça marche super bien. [Notre tutoriel reprend plus de détails](http://mchobby.be/wiki/index.php?title=P82B715PN:_I2C_Bus_Extender) (comme les valeurs des résistances par exemple)

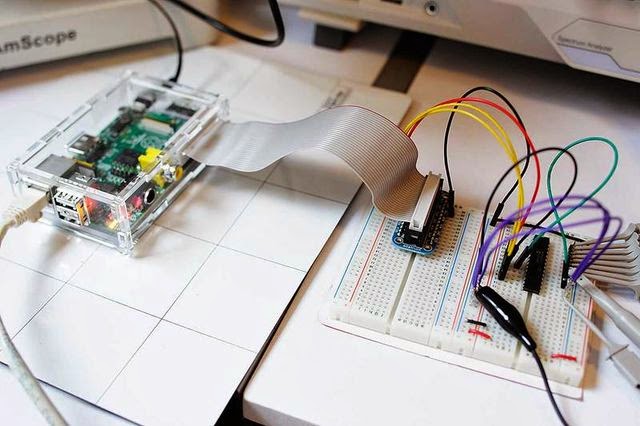
|  |
| --- |
| [https://3.bp.blogspot.com/-W93IlyZd7So/UUIr5ypjaTI/AAAAAAAADJY/HZmUlfGnDLE/s640/P82B715PN-Montage-2.jpg](https://3.bp.blogspot.com/-W93IlyZd7So/UUIr5ypjaTI/AAAAAAAADJY/HZmUlfGnDLE/s1600/P82B715PN-Montage-2.jpg) |

# Ajouter des GPIO à votre Raspberry avec le MCP23017

<https://arduino103.blogspot.com/2014/03/ajouter-des-gpio-votre-raspberry-avec.html>

Dominique Meurisse (MCHobby) lundi 31 mars 2014

Vous utilisez les broches GPIO de votre Pi mais vous en voudriez plus??? pas de problème, utilisez un ou plusieurs MCP23017 (ou MCP23008).

[](https://3.bp.blogspot.com/-D1F58GKA6GY/UzkZ5dn2u6I/AAAAAAAAEoU/Hb7vtebME0s/s1600/Rasp-Hack-MCP230XX-01.jpg)

En plus de d'augmenter le nombre d'entrée/sortie digitales, ce composant peut également servir de buffer car le MCP230xx est capable de fournir 25mA par broche là ou le GPIO du Pi est limité à 17mA par broche (avec un total de 50mA pour tout le GPIO)  
  
Si le sujet vous intéresse, je vous propose de vous [attarder dans le tutoriel](http://mchobby.be/wiki/index.php?title=Rasp-Hack-MCP230XX) qui se trouve être nettement plus complet  
  
**Comment brancher**

|  |
| --- |
| <https://3.bp.blogspot.com/-IY32l4hNg_A/UzkaYZwMp6I/AAAAAAAAEoc/XPgPMEKtfuk/s1600/Rasp-Hack-MCP230XX-Wire-01.jpg> |
| Plus de détail sur le raccordement dans notre tutoriel |

### Alimentation du MCP23017 en 3.3v

Dans le cadre de ce montage, le MCP est alimenté en 3.3v. Ses entrées/sorties fonctionne donc aussi en 3.3v.

### Alimentation du MCP en 5V

Alimenter votre circuit intégré MCP en 5V permet d'avoir les entrées/sorties du MCP23017 en 5V :-). Il est également possible d'alimenter le MCP en 3.3V mais l'alimentation du Pi sait fournir plus de courant sur la ligne 5V (ce qui peut être plus intéressant).   
Etant donné que ce composant utilise I2C pour communiquer, vous pouvez l'alimenter 5V du pi pour alimenter votre MCP même si vous connectez les lignes de données (SDA, SCL) 3.3V de votre Raspberry Pi. C'est possible parce que l'interface I2C du Pi utilise deux résistances Pull-up qui ramènent le potentiel à 3.3V pour les broches SDA et SCL.   
  
**Mais attention:** vous ne pouvez brancher qu'**un seul** composant I2C 5V directement sur le bus I2C 3.3V de votre Pi. Si vous branchiez plusieurs composants I2V 5V directement sur le Bus I2C 3.3V, les résistances PullUp du Pi n'arriverons pas à maintenir la tension à 3.3V max côte Pi. Résultat: vous risquez d'endommager votre Bus I2C... voire votre Pi.   
  
**Si vous avez plusieurs composant I2C 5V**: Vous pouvez très facilement convertir le Bus I2C de 3.3v en 5V en utilisant un "[Level Shifter](http://shop.mchobby.be/product.php?id_product=131)" compatible I2C. Derrière le level shifter, vous pouvez brancher autant de composant I2C 5v que vous voulez.  
  
**Le code**  
Le code de test est relativement simple... mais nécessite l'installation de la bibliothèque [depuis le dépôt GitHub d'AdaFruit](https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code)

#!/usr/bin/python

# -\*- encoding: utf8 -\*-

from Adafruit\_MCP230xx import \*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Utiliser num\_gpios=8 pour un MCP23008

# ou num\_gpios=16 pour un MCP23017!

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# mcp = Adafruit\_MCP230XX(address = 0x20, num\_gpios = 8) # MCP23008

mcp = Adafruit\_MCP230XX(address = 0x20, num\_gpios = 16) # MCP23017

# Déclarer les broches 0, 1 et 2 comme sortie (vous pouvez faire de même pour les broches de 0 à 15)

mcp.config(0, mcp.OUTPUT)

mcp.config(1, mcp.OUTPUT)

mcp.config(2, mcp.OUTPUT)

# Déclarer la broche 3 comme entrée (input) et activer la résistance pullup

mcp.config(3, mcp.INPUT)

mcp.pullup(3, 1)

# Lire l'état de la broche 3 et afficher le résultat

print "Pin 3 = %d" % (mcp.input(3) >> 3)

# Changer l'état de la broche 0

print "Clignoter la broche 0 (CTRL+C pour quitter)"

while (True):

mcp.output(0, 1) # Mettre la broche 0 à l'état HAUT/High

time.sleep(1)

mcp.output(0, 0) # Mettre la broche 0 à l'état BAS/Low

time.sleep(1)

**Error accessing 0x20**  
Je suis certains que nombre d'entre-vous on déjà eu l'occasion de rencontrer l'énigmatique série de messages

Error accessing 0x20:check your I2C address

Error accessing 0x20:check your I2C address

Error accessing 0x20:check your I2C address

Error accessing 0x20:check your I2C address

Error accessing 0x20:check your I2C address

Error accessing 0x20:check your I2C address

Ce dernier se produit lorsque la bibliothèque n'est pas capable de détecter correctement la révision de votre Pi... mais c'est un problème très facile à contourner.   
Lorsque la bibliothèque ne sait pas détecter la révision du pi, elle utilise le Bus I2C sur la broche numéro 0 (correspondant au Raspberry-Pi 256Mb). Le hic, c'est que la majorité des Raspberry actuels sont des modèles 51Mb Modèle B Révision 2 utilisant le Bus I2C sur la broche numéro 1.   
Par conséquent, essayer d'adresser le périphérique I2C à l'adresse 0X20 sur la mauvaise broche du bus I2C ne peut que produire une erreur.

### Fixer le bus à utiliser

Pour résoudre le problème, il suffit simplement de fixer la broche du bus I2C. Editez le fichier Adafruit\_I2C.py avec la commande

nano Adafruit\_I2C

et modifiez le paramètre **busnum** de la ligne suivante

 def \_\_init\_\_(self, address, busnum=-1, debug=False):

pour la fixer de la valeur -1 (auto-détection) à **la valeur 1** (valeur correspondant au Raspberry 512Mb modèle B rev 2)

 def \_\_init\_\_(self, address, busnum=1, debug=False):

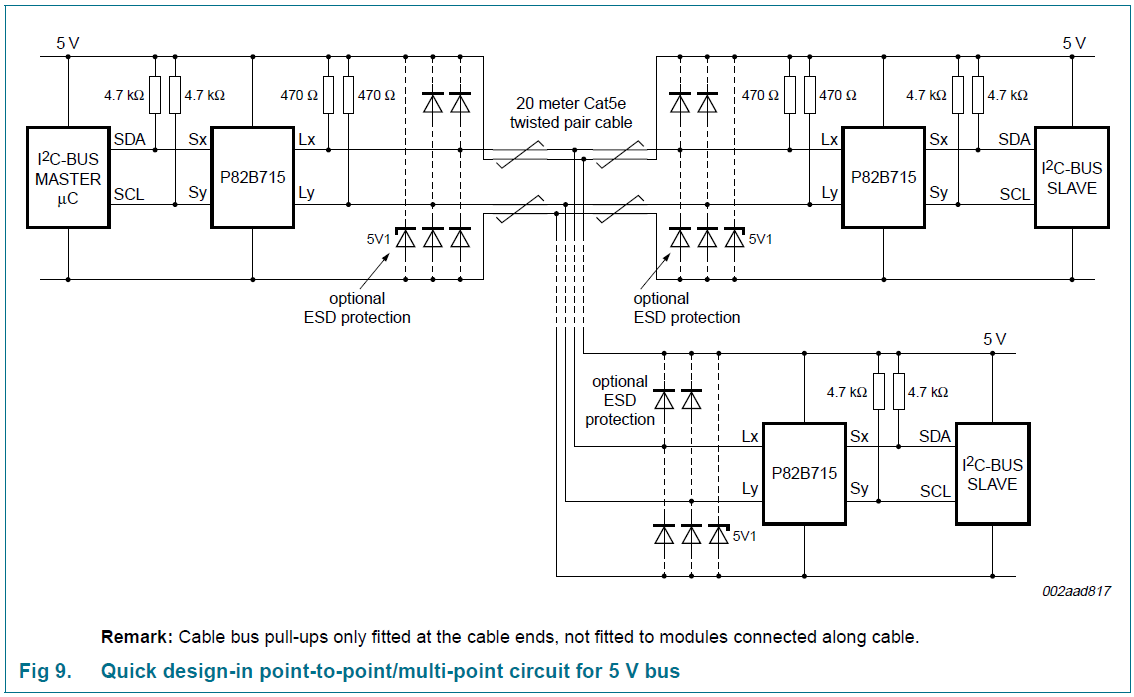
**Tutoriel**  
Vous trouverez de nombreuses autres informations dans notre tutoriel

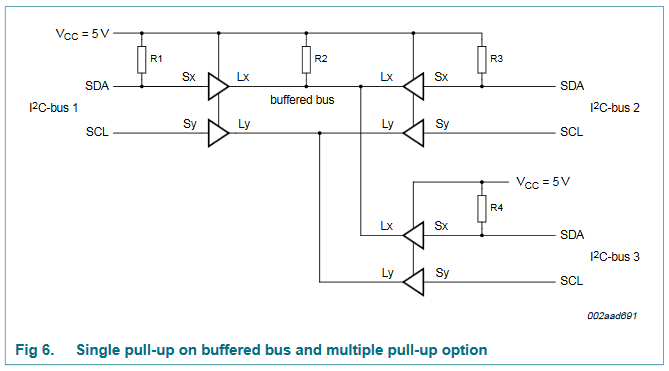
Boost Bus I2C P82B715 et MCP23017

<https://forums.framboise314.fr/viewtopic.php?f=22&t=4843>

29 août 2018 07:57

Je rencontre un souci avec l'extander de bus I2C, et la reconnaissance de mes MCP23017.  
Il y a très peu de docs au sujet du P82B715. Tout le monde dit que c'est très simple (et ça semble l'être), mais ça ne fonctionne pas pour moi.  
J'ai placé le premier P82B715 derrière le Rpi (SDA et SCL). Ensuite viennent les longs câble (que j'ai simulé pour l'instant avec des fils de 15cm sur breadbord) sur lesquelles il faut apparemment placer des resistances de Pull-up, ce que j'ai fait (470 Ohm).  
à l'autre bout l'autre P82B715, puis derrière mon MCP23017.  
  
Lorsque je fais un i2cdetect -y 1, c'est vide. Lorsque je mets directement les pin SDA et SCL du Rpi sur le MCP, il est reconnu.  
  
Est-ce qu'avec le P82B715, cela empeche de voir les périphérique de cette façon ?  
Comment faire pour le voir, l'adresser, interagir avec lui.  
  
quel schéma suivre, ce n'est pas du tout la même chose, dans un j'ai valeurs de résistance différentes, dans l'autre R2 est remplacé par une résistance dans chaque sous-système, avec des diodes de protection.  
  
Une idée ? Merci beaucoup.





Alors, en fait il n'y avait pas vraiment de problème à proprement parler... :x  
Ce qui m'a aidé : voltmètre et fiche technique du P82B715PN.  
J'ai constaté qu sur le premier semi-conducteur j'avais une tension aux bornes 4 et 8 (VCC et GND) < 2V.  
Or dans la fiche technique, la tension minimale de fonctionnement est de 4.5V.  
Le problème provenait du breadboard. J'ai déplacé le P82B715 et remis la conectique à l'identique, tension de 4,8V, OK lors du i2cdetect -y 1 l'adressage apparait.  
  
J'en ai profité pour tester de mettre deux MCP23017 en parallèle sur le bus local de sortie. Bein ça fonctionne, les deux adresse apparaisse, j'ai fait des tests cela fonctionne pour les deux. Donc je ne sais pas ce qu'il faut comprendre dans la doc quand il est question de mettre obligatoirement 1 P82B715 avant chaque périphérique.  
  
J'ai testé avec un long câble c'est nickel. Donc voilà super content, même si personne n'a pu m'aider, si ça peut servir à quelqu'un...