\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RASPBERRY PI ET LE PROTOCOLE I2C

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Partie 1 : Présentation de I²C

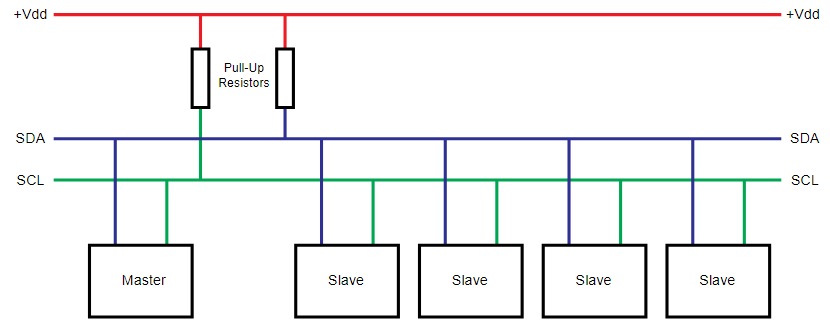
Protocole de communication série développé par Philips Semiconductor (NXP).

L’I2C est utilisé sur une large gamme d’appareils, des microcontrôleurs, CAN, DAC aux EEPROM, périphérique E/S et horloges en temps réel.

I2C conçu pour être un bus semi-duplex multi-esclaves, un maître peut communiquer avec plusieurs esclaves connectés au même bus mais la communication ne peut avoir lieu que dans un sens à la fois.

Alors que l’I2C est conçu pour autoriser plusieurs appareils maîtres, sur le Raspberry Pi, le Pi agit en tant que maître et tous les appareils connectés au bus sont des esclaves.

L’interfaces I²C



Le bus I²C utilise deux fils appelés SDA et SCL.

SDA (données série) transporte les paquets de données envoyés par les appareils maître et esclave.

SCL (horloge série) transporte un signal d’horloge généré par le maitre et sert à synchroniser les paquets de données sur le fils SDA entre le maître et les esclaves.

SDA et SCL sont connectés au fil d’alimentation +Vdd via une paire de résistance de rappel. Donc l’état pas défaut du bus I2C est élevé à la tension +Vdd. Les tensions +Vdd sont 3,3V et 5V bien que d’autres tensions puissent être utilisée.

Les données sont envoyées sur le bus I2C en tirant le fil SDA à 0V. Les signaux d’horloge sont générés par le maître et les dispositifs esclaves envoient un paquet de données en réponse à une demande du maître.

Comme le bus utilise la tension +Vdd du maître tous les appareils esclave fonctionnent également généralement à la même tension.

Si un appareil esclave doit fonctionner à une tension différente de celle du maître, un décaleur de niveau I2C peut être utilisé pour servir d’interface entre les différentes tensions du bus.

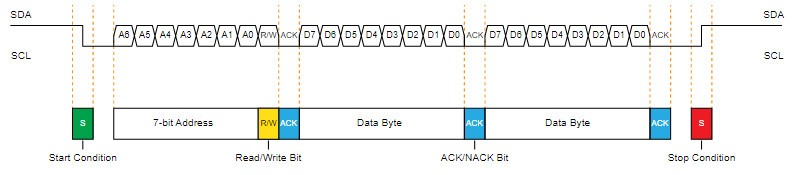
Adresse I2C

Chaque esclave à une adresse unique. Le protocole permet l’utilisation d’un système d’adressage à 7 ou 10 bits. (10 bits rarement utilisé). L’adressage 7 bits donne une plage d’adresses autorisée de 0x03 à 0x77. L’appareil maître enverra l’adresse au début de toute communication et seul l’appareil esclave à cette adresse répondra.

Si un circuit nécessite l’utilisation de plusieurs dispositifs esclaves partageant la même adresse I2C, un commutateur I2C peut être utilisé pour diviser un bus I2C en plusieurs bus séparés, chaque esclave étant connecté à son propre sous-bus. Avant de communiquer avec un esclave, le commutateur peut être réglé sur canal avec l’esclave requis et seul cet esclave sera alors visible pour le maître.

Certains appareils I2C incluent une adresse All-Call qui peut être utilisée pour envoyer des données à plusieurs appareils esclaves simultanément. Alors que ces appareils recevront les données sur l’adresse All-Call, chaque appareil ne renverra des données au maître que sur sa propre adresse unique.

Protocole I2C



Paquet de données I2C avec une adresse de 7 bits.

Chaque octet d’un paquet I2C a une longueur de 8 bits, suivi d’un 9e bit d’accusé de réception généré par le périphérique esclave récepteur en tirant le fil SDA vers le bas pendant un cycle d’horloge. Un bit d’accusé de réception est affiché dans le logiciel I2C sous la forme ACK. Si l’octet n’est pas acquitté par un esclave, le bit d’acquittement est affiché comme NACK.

En début et fin de paquet se trouvent les conditions de démarrage et d’arrêt et sont toujours générées par le maître.

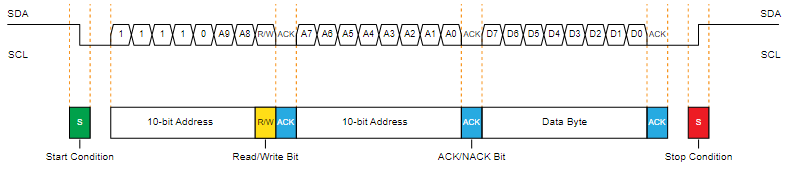
. Début : transition de haut en bas sur SDA alors que SCL est élevé.

. Arrêt : transition de bas en haut sur SDA alors que SCL est haut.

Après la condition de démarrage, un octet d’adresse est renvoyé par le maître.

L'octet d'adresse se compose de 7 bits d'adresse suivis d'un bit de lecture/écriture. Le bit de lecture/écriture est défini sur 1 lors de la lecture des données de l'esclave et sur 0 lors de l'écriture des données sur un esclave.

Adresses 10 bits :



Dans un paquet d'adresses de 10 bits, l'adresse est contenue dans les deux premiers octets. Le premier octet d'adresse se compose de quatre 1, un 0, deux bits d'adresse et le bit de lecture/écriture. Le deuxième octet d'adresse se compose des huit bits restants.

Une fois l'adresse envoyée par le maître, les octets de données seront envoyés ou reçus. Les cycles d'horloge sur le fil SCL sont générés par le maître, donc dans une condition de lecture, l'esclave attendra chaque cycle d'horloge avant d'envoyer ses données en tirant le fil SDA vers le haut ou vers le bas.

Vitesses d’horloge

L’I2C supporte une vitesse de 100 kbit/s, mais plus tard elle a été augmentée pour prendre en charge de nouveaux appareils et nous avons 100 (standard) 400 kbit/s (pleine vitesse) 1Mbit/s (mode rapide) 3,2 Mbit/s (haut débit) mais vérifier la fiche technique de l’appareil pour la prise en charge des vitesses d’horloge.

Etirement de l’horloge

Si un périphérique a besoin de plus de temps pour exécuter une autre fonction avant d’envoyer une réponse, il peut tirer le fil SCL vers le bas, forçant le maître à attendre qu’il soit prêt. Plus d’infos dans le manuel I2C de phillips/NXP.

<https://www.nxp.com/docs/en/application-note/AN10216.pdf>

1. Partie 2 : Activation de l’I²C sur le Raspberry Pi

Le port I2C sont les broches GPIO2 (SDA) et GPIO3 (SCL)



Activer dans les préférences l’I2C redémarrer ou raspi-config et reboot,

.sudo i2cdetect -y 1 #

Installation des outils I2C et bibliothèques Python.

Si le SE est le lite installer le package i2ctools et bibliothèques Python.

Si le SE et Buster ou antérieur

.sudo aptèget upadate

.sudo apt-get install python-smbus python3-smbus python-dev python3-dev i2c-tools

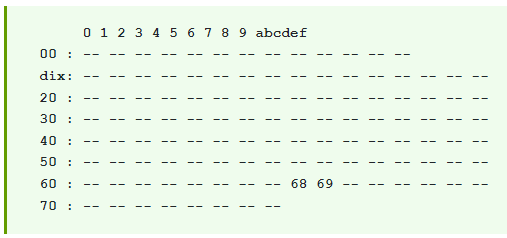
Si le SE Bulleye release et os python 2.7

.sudo aptèget update

.sudo apt-get install python3-smbus python3-dev i2c-tools

On peut tester sudo i2cdetect -y 1

I2cdetect affichera une grille de nombres avec les adresses de tous les appareils I2C affichés dans la grille. Dans l’exemple suivant, deux appareils I2C sont affichés aux adresse 0x68 et 0x69



Réglage de la vitesse du bus I2C

Le bus prend en charge plusieurs vitesses de bus, généralement 100KHz, 400KHz, 1MHz, 3,4MHz et 5,0MHz.

Définir la vitesse en éditant /boot/config.txt et ajoutez au bas du fichier ; Le nombre est la fréquence du bus I2C en hertz : dtparam=i2c\_baudrate=400000 ; et enregistrer les modifications sudo reboot

Notes

Pour le Pi3B 3B+ et Zero W, le contrôleur a son horloge liée au noyau VPU de sorte que la fréquence VPU change en fonction de la charge du processeur, de même que la fréquence d’horloge du bus I2C. Cela peut causer des problèmes avec tous les appareils qui s’attendent à une fréquence d’horloge constante pendant la communication.

Pour résoudre ce problème, la fréquence principale du VPU doit être définie sur une fréquence fixe en modifiant /boot/config.txt

.sudo nano /boot/config.txt

Ajoutez le texte suivant au bas du fichier : core\_freq=250 et redémarrer.

Choisir la bonne vitesse de bus I2C

Voir tous les périphériques la vitesse inférieurs au maximum et choisir la vitesse de l’appareil le plus lent.

Adresse des appareils I²C

Le maître doit connaître l’adresse attribuée à cet appareil. Généralement de 7 bits ça donne au total 128 adresses d’appareils possibles.

Certains ont leurs adresses prédéfinies par le fabriquant d’autres auront une petite plage d’adresses sélectionnables par l’utilisateur. Pour trouver la liste connecter au bus : sudo i2detect -y 1

Le Pi d’origine avec l’entête GPIO à 26 broches vous devez peut être changer en -y 0

1. Partie 3 : Outils I²C sous Linux

Le bus Raspberry Pi I2C prend en paramètre pour l’ID du bus cible ID1 d’où sudo i2cdetect -y 1 d’autre ID 0.

Rasp A, B nous avons un GPIO à 26 broches avec en SDA broche 3 et SCL broche 5

Rasp A+, B+ , 2, 3 à 40 broches ID1 broche 3 SDA (données) et broche 5 SCL (horloge)

Et un bus secondaire pour les données de configuration HAT EEPRIM sur broche 27 SDA et broche 28 SCL.

.sudo apt install i2c-tools pour la commande i2cdetect -y 1

On peut aussi avec les commandes

. i2cdetect détecte les puces I2C connectées au bus

.i2cdump Examiner et lire les registres I2C sur un périphérique connnecté

.i2cget Lire à partir des registres de la puce I2C/SMBus sur un appareil connecté

.i2cset Définir les registres I2C sur un appareil connecté avec de nouvelles données ou valeurs.

.i2ctransfert, Envoyer des messages I2C définis par l’utilisateur en un seul transfert vers un appareil connecté.

.i2cdetect [-y] [-a] [-q|-r] i2cbus [premier dernier]

.i2cdetect -F i2cbus

.i2cdetect -V

.i2cdetect -l

.i2cdetect est un programme pour scanner un bus I2C les paramètre premier et dernier limitent la plage de balayage.

-y désactive le mode interactif et effectue l’opération directement utile dans les script ou lors de détection de périphériques.

-F répertorie les fonctionnalités de l’adaptateur

-V affiche la version et quitte

-l affiche une liste des bus i2C installés.

Exemples :

Liste des bus I2C disponible i2cdetect -l

Répertoriez tous les appareils détectés sur le bus I2C 1 i2cdetect -y 1

.i2cdump [-f] [-r premier-dernier] [-y] [-a] adresse i2cbus [mode [banque [bankreg]]]

.i2cdump -V

.i2cdump est un programme d’aide pou lire et examiner les registres sur les appareils connectés via le bus I2C.

Avertissement

i2cdump peut être dangereux s'il est mal utilisé. Plus particulièrement, le mode c commence par l'ÉCRITURE d'un octet sur la puce. N'utilisez pas i2cdump sur des adresses aléatoires.

Exemples :

Vide le contenu du périphérique I2C à l’adresse 7 bits 0x68 sur le bus 1 en utilisant le mode d’octet de lecture par défaut avec une confirmation de l’utilisateur : i2cdump 1 0x68

Même chose mais à l’aide de transactions de lecture de bloc I2C sans confirmation de l’utilisateur : i2cdump \_y 1 0x68 je

.i2cget [-f] [-y] [-a] i2cbus chip-address [data-address [mode]]

.i2cget -V

.i2cget aide pour lire les registres sur les appareils connectés via le bus I2C

Avertissement

i2cget peut être dangereux s'il est mal utilisé. Certaines puces peuvent traiter une commande de lecture I2C comme une commande d'écriture et écrire des données incorrectes dans un registre.

Exemples :

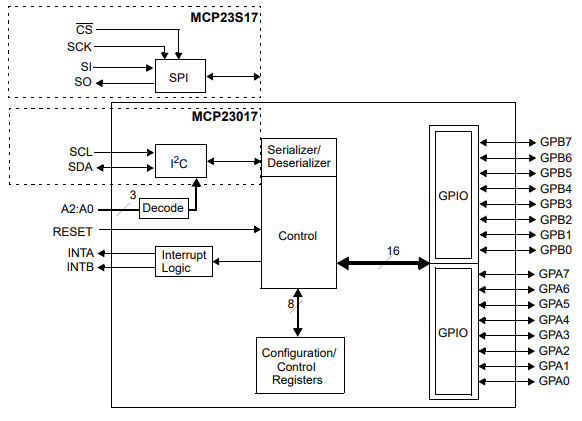
Lire le contenu d’un périphérique I2C à l’adrese 7bits 0x68 sur le bus 1 à partir d’un registre 8bits à 0x20 aprsè confirmation de l’utilisateur : i2cget 1 0x68 0x20 (sinon ajouter w en fin pour confirm user)

… <https://www.abelectronics.co.uk/kb/article/1094/i2c-part-4---programming-i-c-with-python>

1. Partie 4 : Programmation I2C avec Python

Nous avons un contrôleur d’E/S MCP23017 de Microchip avec 16 broches GPIO

<http://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf>

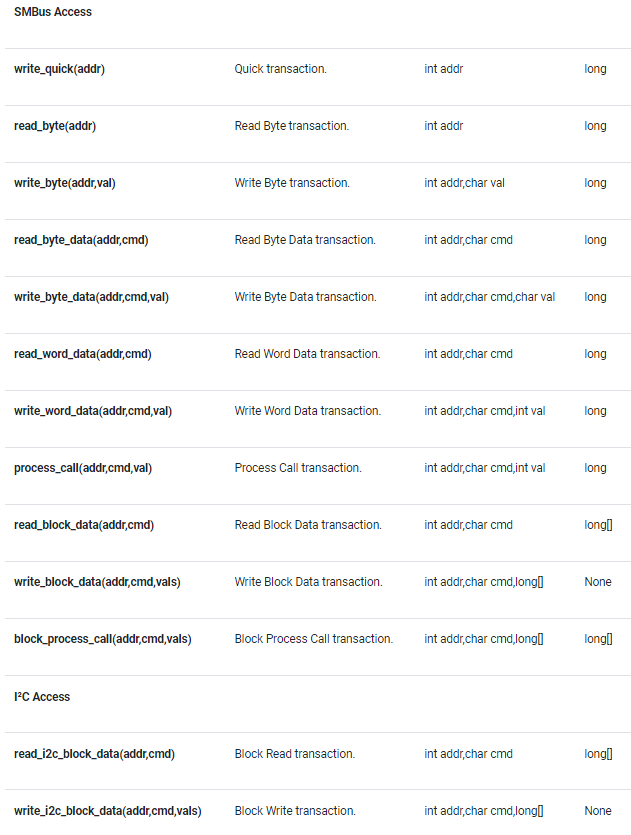


Le contrôle ce fait via ses registres de 8 bits accessible à partir d’une adresse spécifiée, il a 21 registres, 10 associés au port A, 10 au port B et un registre de contrôle qui est partagé entre les deux ports. (liste registre p.17) pour 8 bits : 1000 0111 = 135 = 0x87 (1 octet = 8bits)

Pour vous connecter au bus I2C en Python on a besoin d’une bibliothèque smbus inclus dans raspbian.

On utilise la fonction time.sleep pour mettre le programme en pause

Fonction description paramètres valeur de retour



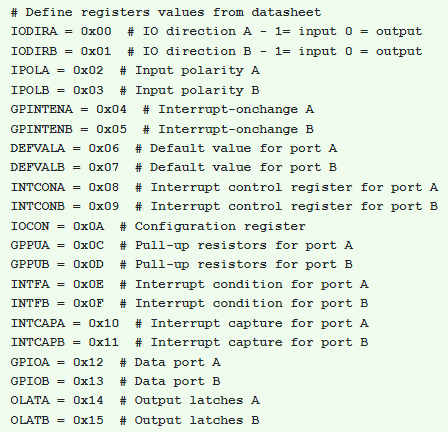
Pour le moment nous utiliserons write\_byte\_data(), write\_word\_data() et read\_byte\_data() pour lire et écrire des données à partir d’adresse de registre sur un périphérique I2C.

Définir les registres :

MCP23017 dispose de 21 registres numérotés de 0x00 à 0x15.

Pour définir la direction du port B le registre 0x01, pour lire à partir du port A le registre 0x12. L’ensemble du programme pourrait se faire à base de registre mais le code serait compliqué on donnera un nom à chaque registre en fonction de sa fonction. La doc technique comprend une liste de noms pour les registres.

#define registers values from datasheet



**Créer l’objet SMBus qui représente le bus I²C**

i2cbus = SMBus(1) # Create a new I2C bus le parameter est l’ID du bus I2C qu’on souhaite utiliser GPIO 40 pin.

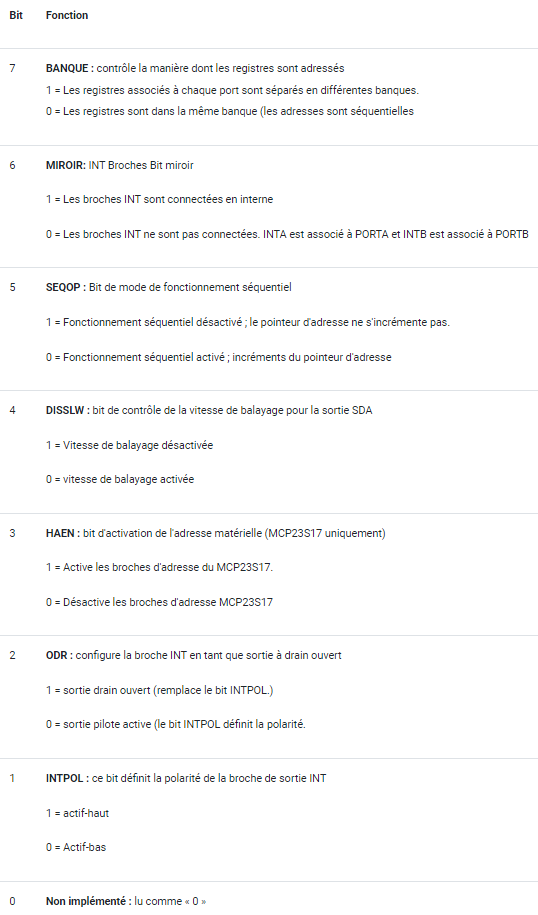
i2caddress = 0x20 # Address of MCP23017 device le MCP dispose de 3 broches d’adresse pour l’utilisateur ente 0x20 et 0x27 si toutes les broches sont faibles c’est 0x20

**Le registre de contrôle**

La puce du MCP a deux listes d’adresses de registre. Dans la première la première les registres associés à chaque port sont séparés en différent banques et la seconde les registres sont dans la même banque et les adresse sont séquentielles.

Pour indiquer que notre liste de variables de registre pointe vers les bonnes on indique au MCP quelle version de la liste nous voulons utiliser.

Nous réglons le registre de contrôle IOCON a une taille de 8 bits et chaque bit contrôle une partie différente des fonctions de la puce voici la liste.



La bank de registres sont contrôlées à l’aide du bit 7 et nous voulons les adresses des registres soient séquentielles, nous allons définir ce bit à 0.

En utilisant des adresses de registre séquentielles, cela nous donne la possibilité de mettre à jour les deux ports sur le bus en même temps en écrivant à deux adresses de registres de manière séquentielle avec deux octets. Pour activer cela, nous devons activer le mode de fonctionnement séquentiel en réglant le bit 5 sur 0.

Par défaut les deux broches d’interruption sont définies comme actives au niveau bas, cela signifie que les deux broches fourniront de l’énergie lorsque la fonctionnalité d’interruption n’est pas utilisée. Nous pouvons désactiver les broches par défaut en définissant le bit 1 sur 1.

Les bits 6, 4, 3, 2 et 0 peuvent être laissés dans leur état par défaut de 0, de sorte que le seul bit qui doit être mis à 1 est le bit 1. Cela donne une valeur de 0b00000010 ou 0x02.

On écrit dans le registre IOCON avec

write\_byte\_data(@ du périphérique, registre qu’il faut écrire, valeur 8 bits à écrire)

Nous voulons écrire à l’adresse I²C définie dans la variable i2address, le registre IOCON et une valeur 0x02 :

On mets à jour le registre de configuration :



**Réglage de la direction du port**

Chaque broche a une direction IODIRA, IODIRB, IODIRA contrôle le port A pin 1 à 8

IODIRB, le port B pin 9 à 16

Pour définie une pin entrée une valeur de 1 est utilisée pour une sortie une valeur 0.

Nous allons définir le port A en sorties et le port B en entrées.

Nous pouvons utiliser write\_byte\_data() deux fois pour mettre à jour IODIRA puis IODIRB, mais dans le registre de contrôle nous avons activé l’écriture séquentielle, ce qui signifie que nous pouvons mettre à jour les deux registres de direction en écrivant deux octets sur le périphérique.

Nous utiliserons write\_word\_data(adresse, registre, valeur) avec les mêmes valeur sauf valeur prend 16 bits.

Définir le port A comme sorties et le port B comme entrées



**Lire et écrire aux ports**

On peut commencer à lire et écrire sur les ports

Créez un while infini puis on peut :

Lire les entrées sur le port B la fonction renvoie un seul octet :



Affiche la valeur du port B



Ecrire une valeur sur le port A, nous avons 2 options écrire directement dans le registre GPIOA (mise à jour toutes les broches) ou dans les verrous de sortie à l’aide du registre OLATA (dans le registre OLAT met à jour que les broches définies comme sorties).

Comme le port A sont des sorties il n’y a aucune différence entre l’écriture nous utilisons le registre GPIO.

Pour voir un changement nous activons la pin 1, dormir pendant 500 millisecondes, désactiver la broche 1 et attendre 500 millisecondes supplémentaires. (Onde carré d’environ 1 seconde) ce ne sera pas exactement 1 seconde car nous lisons sur le port B dans chaque boucle ce qui prend 2 millisecondes pour se terminer lorsque la fréquence d’horloge I²C est de 100KHZ.

#activez la broche 1





#désactivez la broche 1





**Exécution du programme** dans le I2c\_Python\_raspi

