Bus et modules I²C pour Arduino

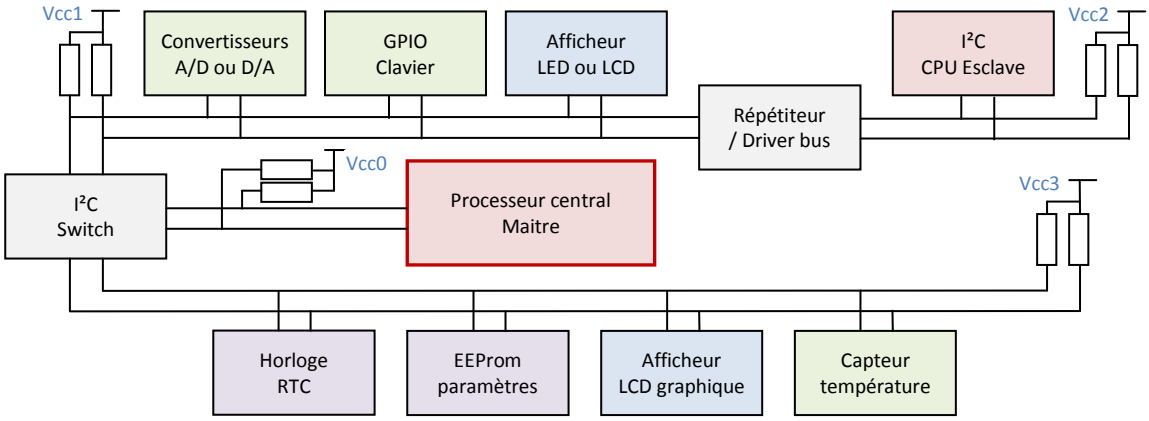
(Bus stop ou 7 jours de réflexion ?)

Table des matières

Généralités Bus I²C

A l'origine le bus I²C (Inter IC) ou parfois dénommé TWI (Two wire interface) est un système développé par Phillips - NXP pour simplifier les problèmes de câblage interne a une machine causés par l'inflation du nombre de systèmes d'entrée sorties (actuateurs, claviers, afficheurs) et pour standardiser les échanges entre composants complexes (mémoires, horloges RTC …..).

Ce bus constitué de deux signaux actif permet un dialogue bidirectionnel entre les différents périphériques directement câblés en parallèle, l'utilisation optionnelle de multiplexeurs et de drivers de bus permet d'en augmenter les possibilités et le nombre de périphériques connectés.



En raison de sa souplesse d'emploi le bus I²C est devenu un standard de facto, de nombreux constructeurs l'ayant adopté et distribuant une multitude de composants proposants des fonctionnalités variées dépassant les simples ports d'entrée sortie de base d'origine. On trouvera par exemple :

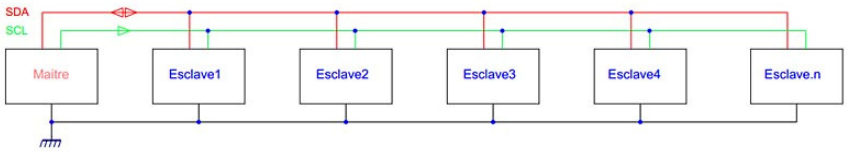
- Gpio (Port E/S) : PCF8574-8575, PCA950x, MCP23017
- Drivers et afficheurs à Led : HT16K33, TLC5947, MAX6955-58-59, PCA953x-955x
- Afficheurs LCD alphanumériques ou graphiques : PCF8577
- Gestionnaires de clavier : LM8330, MPR121
- Convertisseurs A/D- D/A : ADS1000, LTC2309, MCP4725, DAC7571, DAC8574
- Potentiomètres : AD5171, AD5243, MCP4661, TPL401, Max5417-19,
- Gyroscopes et capteurs 6 axes : FX0S8700,
- Sondes de pression ou temperature : LM75, SE95, SA56004, MPL3115,
- Mémoires EEprom : Atmel 24Cxx, 24M0x, PCF8594, PCF8570 (Ram)
- Ponts Série ou USB : Max3107, CY7C6521,
- Horloges RTC : DS1307, DS1337, DS3231, PCA85063, PCF8523
- Récepteurs GPS

N'étant conçu que dans un but d'interconnexion de systèmes d'une même machine ce bus n'est pas prévu pour des liaisons distantes, en raison des faibles capacités parasites permises par l'utilisation de sorties a collecteur ouvert la longueur maximale d'un bus ne pourra dépasser quelques mètres. Pour des distances supérieures plusieurs solutions préférables existent dont les liaisons série RS232 ou RS485.

Rappels fonctionnement bus I²C

Principe de fonctionnement simplifié protocole

Le bus I²C est basé sur un mode de transmission bidirectionnel sériel synchrone entre un maitre et un ou plusieurs périphériques esclaves. Ce bus dit 2 fils utilise outre la référence Gnd un signal d'horloge de synchronisation (SCL Serial Clock Line) et un signal de donnée (SDA Serial Data Line). Si chaque élément connecté en parallèle au bus reçoit les données émises par tous les autres, seul le maitre génère le signal d'horloge.

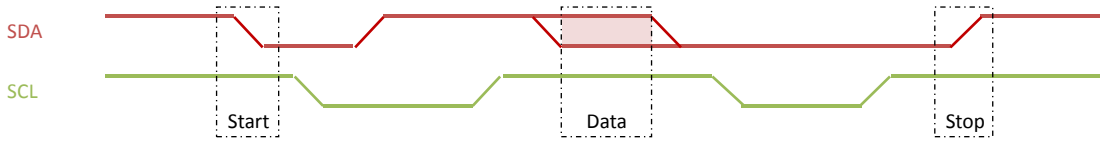


Chaque esclave est sélectionné par une adresse codée un octet : 7 bits d'adressage effectifs plus celui de poids faible R/W déterminant le sens du dialogue. L'adresse 00 étant réservée au broadcast (envoi simultané de données a l'ensemble des esclaves) il est alors possible dans les limites de l'aspect physique du bus de connecter 127 périphériques différents simultanément. Dans l'absolu ceci n'est pas tout à fait vrai, quelques autres adresses étant réservées et un mode étendu 10bits permettant 1024 adresses étant possible, dans le cadre courant de ce document seul le mode de base sera utilisé.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0 : Ecriture 1 : Lecture
Adresse périphérique esclave							R / W	
Adresse complète								

L'ensemble des utilisateurs du bus pouvant être alternativement maitre (initiateur d'un dialogue) ou esclave et utilisant les mêmes signaux simultanément plusieurs problèmes vont être rencontrés dont en particulier les phénomènes de collision. Pour éviter que plusieurs périphériques émettent au même moment un système de ticket va être utilisé avec la génération de deux bits spécifiques Start et Stop, la détection d'un bit Start sur le bus provoque le passage en mode esclave des autres périphériques pendant une durée évitant tout risque de collision

Les bits Start et Stop sont respectivement définis par une transition Haut-bas et Bas-Haut de SDA quand l'horloge SCL est au niveau haut, les données doivent elles être stables pendant cette période.



Les séquences de transfert de données du maitre vers le périphérique esclave utiliseront les structures suivantes :

Maitre		Ecriture vers le périphérique		Esclave
Envoi	>	Bit Start : Ouverture transaction	>	Ecoute générale
Envoi	>	AAAA AAA0 : Adresse esclave WRITE	>	Lecture et attente data
Lecture	<	Bit Acquittement esclave prêt + Pause	<	Envoi
Envoi	>	DDDD DDDD : Donnée	>	Lecture et memo data
Lecture	<	Bit Acquittement donnée bien reçue + Pause	<	Envoi
Envoi	>	DDDD DDDD : Donnée	>	Lecture et memo data
Lecture	<	Bit Acquittement donnée bien reçue + Pause	<	Envoi
		.....		
Envoi	>	Bit Stop : Fermeture transaction	>	Passage en écoute générale

Maitre		Lecture depuis le périphérique		Esclave
Envoi	>	Bit Start : Ouverture transaction	>	Ecoute générale
Envoi	>	AAAA AAA1 : Adresse esclave READ	>	Lecture et attente data
Lecture	<	DDDD DDDD : Donnée	<	Envoi data
Envoi	>	Bit Acquittement donnée bien reçue + Pause	>	Lecture
Lecture	<	DDDD DDDD : Donnée	<	Envoi data
Lecture	>	Bit Acquittement donnée bien reçue + Pause	>	Lecture
		.....		
Envoi	>	Bit Stop : Fermeture transaction	>	Passage en écoute générale

Chaque octet émis est suivi lors du neuvième top d'horloge par un bit a zéro ACK d'acquittement si l'opération s'est bien déroulé ou au contrait un bit a l'état un NACK dans le cas contraire.

Les échanges entre les périphériques I²C seront donc toujours effectués par trames constituées d'un octet d'adressage suivi de un ou plusieurs octets de donnés, le tout géré par le périphérique maitre, ce principe de fonctionnement se retrouvera dans l'architecture de la bibliothèque Arduino wire basée elle aussi sur l'ouverture et la fermeture de blocs de transaction.

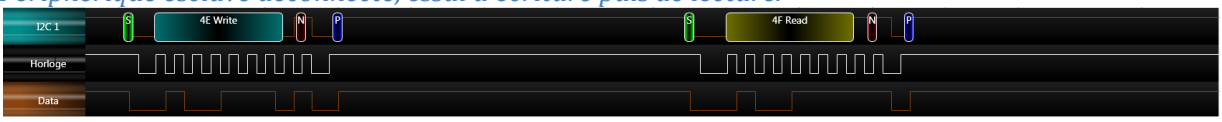
Un descriptif complet du fonctionnement du bus peut être trouvé dans le document UM10204 édité par Phillips / NXP :

http://www.nxp.com/documents/user_manual/UM10204.pdf

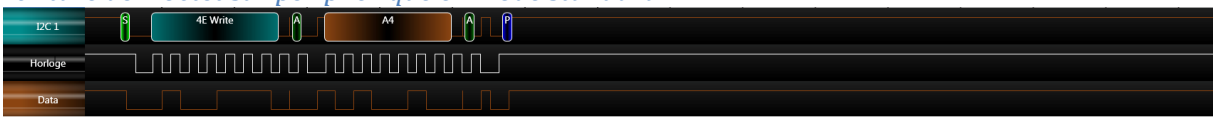
Exemples de trames I²C

Ces trames ont été relevées avec le programme de test exemple de la bibliothèque wire avec un périphérique d'adresse 0x27 => Valeur premier octet envoyé en écriture 27 << 1 + 0 = 0x4E, en lecture : 27 << 1 + 1 = 0x4F.

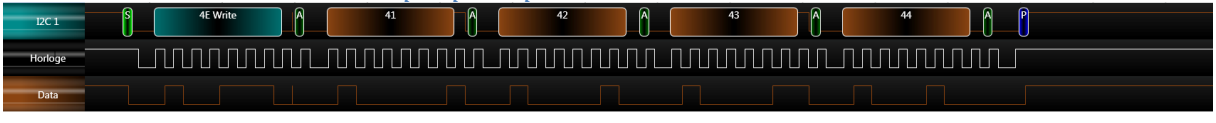
Périphérique esclave déconnecté, essai d'écriture puis de lecture.



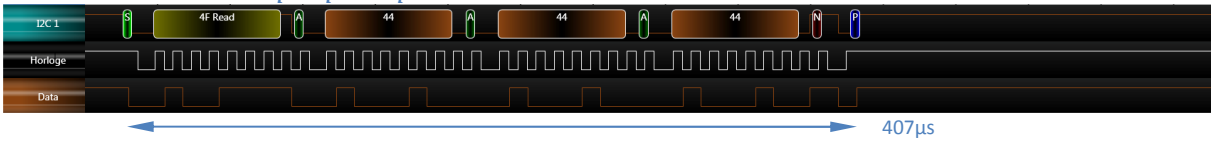
Ecriture de 1 octet sur périphérique en mode standard 100kHz



Ecriture de 4 caractères ascii sur périphérique en mode standard 100kHz



Lecture de 3 octets sur périphérique en mode standard 100kHz

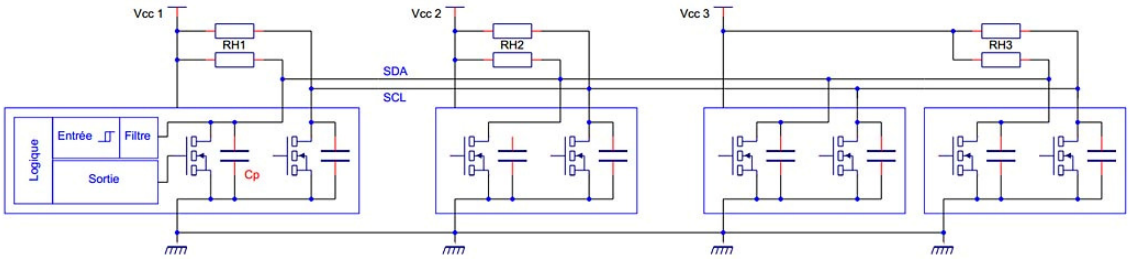


Récapitulatif composants courants et plan d'adressage

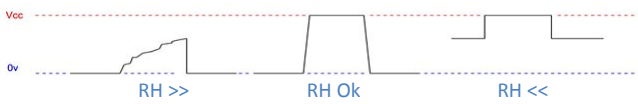
Composant	Type	Marque	Nb entrées d'adressage	Plage d'adresse (hex 7bits)	Vitesse bus		
					Std	Fast	F+
-	Broadcast			0x00			
-	Reset (option certains composants en écriture seule)			0x03			
LSM 303	Accéléromètre + boussole 3 axe	ST	0	0x19	●	●	
LSM 9DS0	Gyroscope, Accéléromètre 3axes	ST	1	0x 1E-1F + 0x6A-6B	●	●	
PCF8574	GPIO x 8	Générique	3	0x20 - 0x27	●		
PCF8575	GPIO x 16	Générique	3	0x20 - 0x27	●		
MCP23008	GPIO x 8	Microchip	3	0x20 - 0x27	●	●	●
MCP2317	GPIO x 16	Microchip	3	0x20 - 0x27	●	●	●
TCA9535	GPIO x 16	Texas instrument	3	0x20 - 0x27	●	●	
MAX5417 - 5419	Potentiomètre numérique	Maxim	1	0x28 - 0x29	●	●	
MCP4661	Potentiomètre digital	Microchip	3	0x28 - 0x2F	●	●	●
AD5171	Potentiomètre numérique	Analog Device	1	0x2C - 0x2D	●	●	
AD5248	Resistance variable double	Analog Device	2	0x2C - 0x2F	●	●	
TPL0401A - 401C	Resistance variable	Texas instrument	0	0x2E	●	●	
AD5243	Potentiomètre double	Analog Device	0	0x2F	●	●	
PCF8574A	GPIO x 8	Générique	3	0x38 - 0x3F	●		
SSD1306	Driver afficheur Led matriciel 128x64	Solomon systech	1	0x3C - 0x3D			
TPL0401B	Resistance variable	Texas instrument	0	0x3E	●	●	
PCA9685	Driver 16 leds PWM 12 bits	NXP	6	0x40 - 0x7F	●	●	●
LM8330	Contrôleur de clavier, GPIO	Texas instrument	0	0x44	●	●	
PN532	Interface NFC / RFID	NXP	0	0x48			
ADS1013 -14-15	Convertisseur A/D 12bits	Texas instrument	1 (4 niveaux)	0x48 - 0x4B	●	●	●
ADS1113 -14-15	Convertisseur A/D 16bits	Texas instrument	1 (4 niveaux)	0x48 - 0x4B	●	●	●
LM75	Thermostat -55/+125°	National SC	3	0x48 - 0x4F	●	●	
24Cxx	Mémoire EEprom	Générique	3	0x50 - 0x57	●	●	●
MPR121	Décodeur de clavier capacitif + GPIO	NXP	1 (4 niveaux)	0x5A - 0x5D	●	●	
MCP4725 -A0 a A3	Convertisseur D/A 12 bits 1 canal	Microchip	1 + référence	0x60 - 0x67	●	●	●
MAX6965	Driver multiplexeur Leds + clavier	Maxim	2 (4 niveaux)	0x60 - 0x6F	●	●	
PCA953x - 955x	Driver de leds	NXP	3	0x60 - 0x6F	●	●	
DS 1307 - DS3231	Horloge temps réel	Maxim	0	0x68	●		
HT16K33 (28pin)	Driver multiplexeur Leds + clavier	Holtek	3	0x70 - 0x77	●	●	
MS5607 - MS5611	Capteur de pression barométrique	Mes. Specialities	1	0x76 - 0x77	●	●	

Lignes SDA - SCL

Les lignes SDA et SCL sont pilotées par des transistors en drain ou collecteur ouvert ce qui permet d'adapter le bus a des périphériques disposant de tension d'alimentation différente sans trop de problèmes. Une résistance de tirage à Vcc RH est donc nécessaire pour obtenir le niveau haut de ces signaux.



L'influence de la valeur de cette résistance pourra être cruciale pour le bon fonctionnement du bus en raison des capacités parasites Cp du câblage et des transistors employés et surtout de leurs performances et du courant maximum qu'ils peuvent supporter. Ce dernier point pourra être gênant avec l'utilisation de plusieurs modules I²C du commerce possédant chacun une résistance RH, celles-ci se retrouvant en parallèle du point du vue du transistor actif sur le bus.



Lors de la transition du niveau bas au niveau haut le transistor de commande est désactivé et la capacité parasite Cp se charge à travers la résistance RH. Si sa valeur est insuffisante le temps de montée du signal sera trop important limitant la fréquence d'horloge maximale du signal, une bus d'impédance élevé sera aussi plus sensibles aux perturbations et parasites externes. A contrario une résistance faible provoquera un courant élevé dans le transistor de sortie qui pourrait ne pas permette d'obtenir un niveau bas correct.

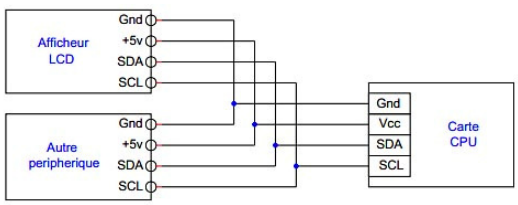
Le Bus I2c ayant évolué au fil du temps et propose plusieurs débits et fréquences d'horloges, ces différents modes de fonctionnement seront à prendre en compte lors de l'estimation des valeurs de ces résistances. La conception du câblage à employer ne sera pas non plus identique entre un bus utilisé en mode standard et un autre en mode Ultra fast.

	Standard	Fast	Fast +	High Speed
Fréquence d'horloge SCL	100kHz	400kHz	1Mhz	4Mhz
V_{IL} Tension max niveau bas	0.3 x Vcc			0.3 x Vcc
V_{IH} Tension min niveau haut	0.7 x Vcc			0.7 x Vcc
I_{OL} : Courant max niveau bas	3mA	3mA	20mA	3mA
t_R : Temps descente signal max	1µs	300ns	120ns	
t_F : Temps montée max signal	300ns	300ns	120ns	
Cp max : Capacité bus maximum	400pF	400pF	550pF	
RH max (Cp = 20pF)	58Ko	17Ko	7Ko	
RH max (Cp = 100pF)	12Ko	3K5	1K5	
RH min (Vcc = 3v3)	960 Ho	960 Ho	150 Ho (20mA)	
RH min (Vcc = 5v)	1K2	1K2	230 Ho (20mA)	

La datasheet UM10204 du constructeur NPX donne pour le calcul de RH les formules approchées suivantes :

$RH\ max = t_R / (0.85 \times Cp)$ $RH\ min = (Vcc - V_{OL}\ max) / I_{OL}$

Connexion Arduino

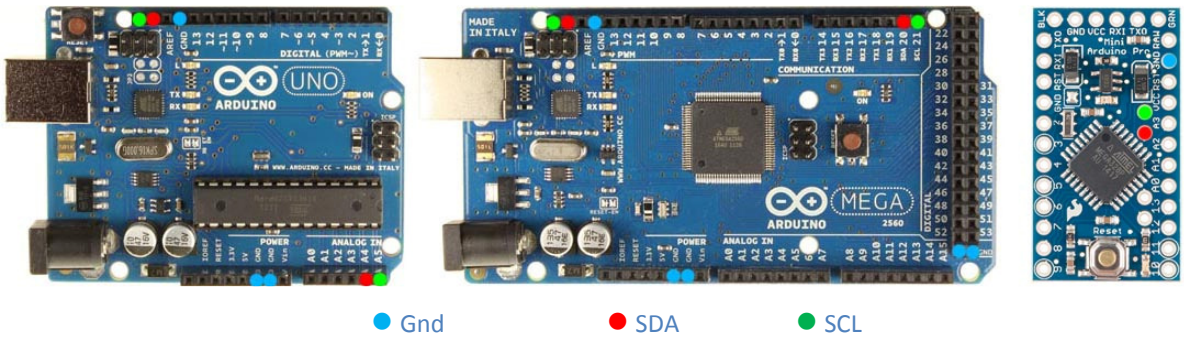


La connexion à une carte Arduino ou toute autre solution basée sur un microcontrôleur ne pose pas de soucis majeurs trois fils seulement étant en jeu, quatre si le périphérique est alimenté par la carte CPU principale.

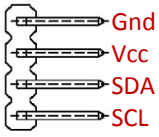
L'emplacement des signaux SDA et SCL change en fonction du type de carte et de processeur utilisé, certaines versions de carte Uno voient ces sorties doublées sur une extension du second connecteur digital, le tableau suivant récapitule les ports utilisés et les fonctions annexes assurées par le processeur outre celle du bus I²C. Certains constructeurs de cartes clones intègrent parfois un emplacement pour les résistances de tirage RH sans que celles-ci soient implantées.

	Proc	Vcc	Connecteur	Sérigraphie		Ports microcontrôleur	
				SDA	SCL	SDA	SCL
Lilipad	168	5v	Périphérie	A4	A5	PC4/Adc4/PCInt12	PC5/Adc5/PCInt13
Lilpad Usb	32U4	3v3	Périphérie	2	3	PD1/Int1	PD0/Int0/OC0B
Uno, Uno Eth.	328	5v	Analog + Extension	Analog4	Analog5	PC4/Adc4/PCInt12	PC5/Adc5/PCInt13
Mega, Due	1280/2560	5v	Digital	Digital20	Digital21	PD1/Int1	PD0/Int0
Pro Mini, Nano	328	5v	Central A4-A5	A4	A5	PC4/Adc4/PCInt12	PC5/Adc5/PCInt13
Pro micro	32U4	3v3/5v	Lateral	2	3	PD1/Int1	PD0/Int0/OC0B
Carte Leonardo	32U4	5v	Digital + Extension	Digital2	Digital3	PD1/Int1	PD0/Int0/OC0B
Wemos D1 (ESP12F)	ESP8266	3v3	Digital + Extension	D4	D3	GPIO 4	GPIO 5
Fishino UNO + ESP8266	328	5v	Analog	Analog4	Analog5	PC4/Adc4/PCInt12	PC5/Adc4/PCInt13
Sparkfun	ATtiny85	3v3/5v	Boitier Dil	Pin5	Pin7	PB0/Mosi/Aref …	PB2/Sck/Adc1…
Sparkfun Pro	ATtiny167	3v3/5v	Lateral droit	0	2	PB0/DI/Int8	PB2/USCK/Int10

Seules les cartes Uno et Mega post révision 3 disposent d'un doublement des bornes de sortie du bus I²C sur une extension du connecteur digital située au voisinage de la prise USB.

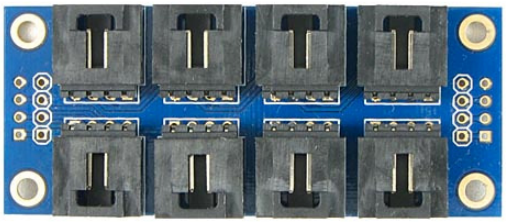


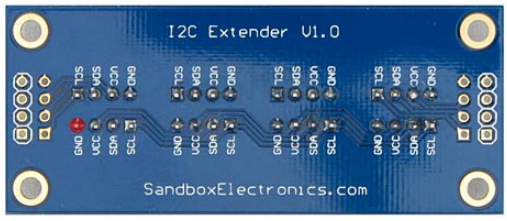
Connecteur et adaptateurs



Sans que cela soit une norme ou une règle de nombreux modules utilisent le modèle de câblage ci-contre pour leur connecteur de liaison …. ordre ne respectant pas les préconisations constructeur de câblage des bus. Les fils de liaison devront être torsadés de préférence deux par deux en associant un fil d'alimentation et un fil de bus, jamais les deux fils SDA/SCL ensembles. En fonction de la longueur et la capacité du câblage la valeurs des résistances de tirage a Vcc du bus pourront être adaptées.

La plupart des shields Arduino d'extension des ports passifs proposant plusieurs connecteur I²C mis en parallèle utilisent cet ordre de câblage qui ne pourra être modifié les broches étant directement connectées aux ports de la carte CPU. Des répartiteurs passifs comme celui représenté ci-dessous offrira quelques opportunités de modification hormis pour Gnd relié au plan de masse du circuit imprimé.



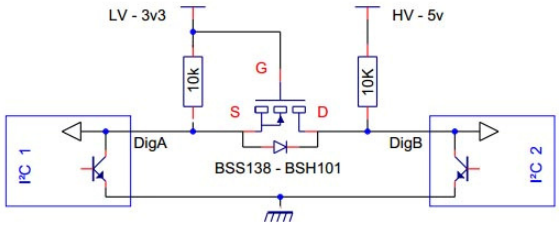


En cas de besoin d'une distance de liaison supérieure a plusieurs dizaines de mètres il sera toujours possible d'utiliser un driver de puissance ou un répétiteur de bus pour éviter les erreurs de transmission. Le document AN10658 toujours édité par Phillips propose plusieurs solutions de ce type avec l'utilisation de câbles standard type cat5e utilisé en informatique.

http://www.nxp.com/documents/application_note/AN10658.pdf

Changement de tension

Si l'emploi de cartes CPU fonctionnant en 3v3 ne pose pas de problèmes les IC périphériques acceptant généralement cette tension d'alimentation, des cartes CPU 5v associées a des périphériques fonctionnant exclusivement sous une tension plus basse nécessiteront une adaptation des signaux SDA et SCL pour un fonctionnement optimal.

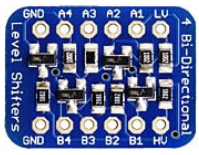


La solution pont diviseur parfois utilisée avec des circuits comme les NRF24 ne peut fonctionner dans ce cas la liaison étant bidirectionnelle, le petit schéma (open hardware) ci-contre couramment employé et utilisé sur de nombreux convertisseurs du marché donne de bons résultats, que ce soit dans ce contexte ou pour tout autre signal ou bus.

Le fonctionnement de ce montage en fonction des états logique des périphériques I²C 1 et 2 est le suivant :

- DigA=3v3, DigB=xx : Le transistor est bloqué sa tension Vgs étant égale a zéro.
- DigA=0v : La tension Vgs du transistor est égale a 3v3, il devient conducteur, DigB=DigA=0v.
- DigB=0v : La diode de protection du transistor est passante, DigA=0.6v environ (niveau L).

Le fonctionnement n'est assuré que pour des transistors possédant un faible Vgs_{th}. Le montage d'origine utilisant des BSS138 (Vgs_{th} < 1v5) ou les platines chinoises souvent des BSH101 (Vgs_{th} < 1v) permettent leur utilisation avec des montages utilisant des tensions d'alimentation LV de 1v8 au minimum.



On trouve généralement ces adaptateurs tout faits sous la forme d'un groupement de 4 canaux, pour un bus I²c seulement la moitié d'entre eux sera donc utilisé.

Bibliothèque wire Arduino

La gestion du bus I²C et des fonctions hardware TWI (Two Wire Interface) des processeurs Atmel est intégré nativement dans l'IDE Arduino et ne nécessite donc pas d'installation de bibliothèques externe.

Par défaut la bibliothèque utilise pour ses variables et buffers interne jusqu'a 220 octets de mémoire Ram et 2.3Ko de mémoire flash programme (bootloader Arduino compris). Certains microcontrôleurs comme les ATtiny peu généreux de ce point de vue limiteront la complexité du logiciel ou les possibilités d'utilisation d'autres fonctions.

Que ce soit pour économiser la quantité de Ram utilisée ou pour obtenir des trames I²C d'une longueur supérieure a 32 octets, il sera possible de modifier la taille des buffers d'émission et de réception utilisée par la bibliothèque en modifiant la valeur de la constante `BUFFER_LENGHT` declarée dans le fichier Wire.h situé dans le répertoire \hardware\arduino\avr\libraries\Wire. Par défaut cette valeur étant égale a 32 les buffers utilisent 32rx + 32tx = 64 octets de Ram, avec la majorité des circuits I²C il sera possible de diminuer cette valeur a 4 ou 8 en adaptant le logiciel.

Fonctions de base bibliothèque

Les variables utilisées par la bibliothèque seront déclarées au format uint8_t ou byte, il sera préférable de passer par une constante pour déterminer l'adresse de chaque périphérique.

`Const uint8_t Add` : Adresse du périphérique distant sur 7bits la bibliothèque gérant le bit de poids faible R/W du dialogue I²C, le bit de poids fort de cette valeur est automatiquement mis à 0.

Déclarations et initialisation

- `#include <Wire.h>` : Déclaration de compilation classique à insérer en tète de programme.
- `Wire.begin();` : Initialisation de la bibliothèque en mode maitre. (A utiliser dans la section `setup()`.)
- `Wire.begin(add);` : Initialisation de la bibliothèque en mode esclave d'adresse add.

Accès bus mode maitre

Dans ce mode la carte Arduino initie les transactions et accède a en lecture ou en écriture a des périphériques esclaves. Ce sera le cas avec la plupart des composants I²C passifs comme les ports IO, les mémoires, afficheurs, ou autres capteurs.

Ecriture : L'envoi de données à l'esclave est réalisée par passage des valeurs à transmettre a un buffer interne de 32 octets avant émission réelle sur le bus I²C de son contenu. Une opération de transmission devra donc toujours être constituée d'une série d'instruction begin/send/end consécutives.

- `Wire.beginTransmission(add);` : Ouvre une session de transmission, et initialise les variables internes pour envoyer des données a l'esclave d'adresse `add`.

- `Wire.write(variable, lg);` : Envoi de `lg` octets contenu dans la variable a l'esclave, avec `lg` inferieur a 32. La variable étant passée à un pointeur, en fonction son type la syntaxe sera :
  - `uint8_t` : Octet seul, lg peut être omis.
  - `string, lg` : Envoi de lg octets de la chaine de caractère.
  - `array, lg` : Envoi de lg octets du tableau de valeurs.

- `result=Wire.endTransmission();` : Ferme la session et renvoie sur un octet une valeur rendant compte du résultat de la transaction.
  - `Result=0` : Transaction ok.
  - `Result=1` : Dépassement du buffer d'émission, toutes les valeurs n'ont pu être émises.
  - `Result=2` : Pas de réponse de l'esclave lors de l'émission de son adresse.
  - `Result=3` : Refus des données par l'esclave.
  - `Result=4` : erreur inattendue.

Lecture : La aussi la réception de données en provenance de l'esclave se fera de manière indirecte par peuplement d'un buffer de réception.

- `Wire.requestFrom(add, NbOct);` : Efface et initialise le buffer de réception, lit `NbOct` en provenance de l'esclave d'adresse `add`. L'opération `NbOct=NbOct & 0x1F` est effectuée en interne pour éviter tout dépassement du buffer de 32 octets. Une réponse nack en provenance de l'esclave arrêtera la fonction avant lecture des `NbOct`.

- `NbRec=Wire.available();` : Retourne le nombre d'octets restants dans le buffer de réception, 0=Buffer vide.

- `Data=Wire.read();` : Retourne sous forme d'un uint8_t l'octet suivant en provenance du buffer de réception.

Le temps d'exécution d'une requête en lecture ou en écriture dépend bien sur du nombre d'octets échangés avec le périphérique. Un dialogue utilisant au minimum deux octets (Adresse + Data) le temps d'execution sera alors d'environ 210µs pour un bus fonctionnant a la vitesse standard.

Accès bus mode esclave

Contrairement au mode précédent la carte Arduino prendra la place de l'esclave, l'ensemble des transactions étant à l'initiative d'une carte processeur distance.

```
Wire.onReceive();


Wire.onRequest();
```

……

Vitesse bus I²c

La fréquence d'horloge de SCL est en mode standard a 100kHz par défaut, il est possible de modifier sa vitesse et passer en mode Fast pour les périphériques esclave le nécessitant.

La fonction `Wire.setClock(freq.)` permet de modifier cette fréquence pour le sketch en cours, elle doit être appelée après la fonction `Wire.Begin()` avec comme paramètre `freq.` au format uint16_t une valeur comprise entre 31000L et 400000L.

```
Void  setup () {
  Wire.begin;
  Wire.setClock(100000L);                    //Mode standard 100kHz
  //Wire.setClock(400000L);                  //Mode fast 400kHz
```

Cette modification peut être réalisée de façon permanente en éditant le fichier twi.h situé dans le répertoire \hardware\arduino\avr\libraries\Wire\src\utility de l'IDE.

```
#ifndef TWI_FREQ
#define TWI_FREQ 100000L                     //Mode standard 100kHz
//#define TWI_FREQ 400000L                   //Mode fast 400kHz
```

Exemple d'utilisation en mode maitre

```
#include <Wire.h>                            //Déclaration bibliothèque
const uint8_t ETest1=7;                      //Déclaration entrées de test, leur mise
const uint8_t ETest2=6;                      //au niveau bas provoque l'exécution du
const uint8_t ETest3=5;                      //test et génère du trafic sur le bus I2c
const uint8_t ETest4=4;
const uint8_t Add_I2C=0x27;                  //Déclaration adresse périphérique

uint8_t CptLoop=0;
uint8_t TrameTest[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}; //Datas a envoyer

void setup() {  // ------------------------------------------------- Initialisation
  Wire.begin();
  Serial.begin(115200);
  Serial.println (" _______ Start");
  pinMode(ETest1, INPUT);
  digitalWrite(ETest1, HIGH);
  pinMode(ETest2, INPUT);
  digitalWrite(ETest2, HIGH);
  pinMode(ETest3, INPUT);
  digitalWrite(ETest3, HIGH);
  pinMode(ETest4, INPUT);
  digitalWrite(ETest4, HIGH);
}

void loop() {  // ---------------------------------------------- Boucle principale

  if (digitalRead(ETest1)==LOW) { // ----------- Envoi valeur CptLoop au périphérique
    Serial.print("Test write 1 octet = ");
    Wire.beginTransmission(Add_I2C);
    Wire.write(CptLoop);
    Serial.println(Wire.endTransmission());
    delay(5);
  }

  if (!digitalRead(ETest2)) { // ----------------- Envoi 4 valeurs Ascii d'une chaine
    Serial.print("Test write n ascii = ");      //Trame I2C=0x4E-0x41-0x42-0x43-0x44
    Wire.beginTransmission(Add_I2C);
    Wire.write("ABCDEFGH",4);
    Serial.println(Wire.endTransmission());
    delay(5);
  }

  if (!digitalRead(ETest3)) { // ----------------------- Envoi 16 octets d'un tableau
    Serial.print("Test write n octets = ");       //Trame I2C=0x4E-0x00-0x01...-0x0F
    Wire.beginTransmission(Add_I2C);
    Wire.write(TrameTest,16);
    Serial.println(Wire.endTransmission());
```

```
    delay(5);
  }

  if (!digitalRead(ETest4)) { // ------------------------------ Lecture de 3 octets
    Serial.print("Test lecture 3 octets = ");        //Trame I2C=0x4F-0x44-0x44-0x44
    if (Wire.requestFrom(Add_I2C, 3) !=3) {Serial.print ("Fail");}
    else {
      while (Wire.available()) {
      Serial.print (Wire.read(),HEX);
      Serial.print(" - ");
      }
    }
    Serial.println();
    delay(100);
  }

  CptLoop ++;
}
```

Périphériques I²C

Adaptateurs digitaux - GPIO

Buffers 8 ports E/S : PCF8574A

Descriptif circuit intégré

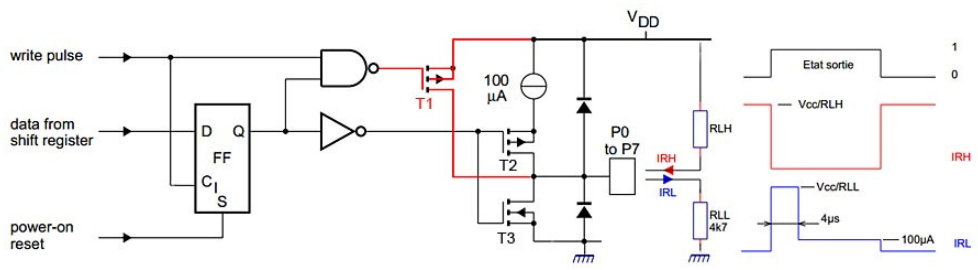
Le PCF8574 est devenu un produit générique a bas cout fabriqué par de nombreux constructeurs et utilisé couramment dans les modules et shields Arduino en raison de sa facilité et souplesse d'emploi. Ce circuit dispose de 8 ports d'entrées sorties et de 3 lignes d'adressage permettant l'association de plusieurs circuits en parallèle sans composants supplémentaires.

Ce composant existe en deux versions 8574 et 8574A ne différant que par leur adresse de base, il sera important de bien déterminer l'indice du circuit intégré utilisé sur des modules pré-montés.

Ports d'entrée sortie

Le PCF8574 dispose de ports E/S dits pseudo bidirectionnels d'un fonctionnement un peu particulier. En se reportant au schéma de principe ci-dessous, trois états peuvent être distingués en fonction du niveau logique du port.

- Etat bas : T3 est actif, le courant de sortie maximal d'un charge connectée a Vcc ne doit pas dépasser 25mA (10mA typ), l'utilisation du port en entrée au niveau Haut n'est pas possible sans résistance de limitation.
- Etat haut : T2 est actif, le courant de sortie maximal est géré par le générateur de courant constant et ne peut dépasser 100µA environ. L'utilisation du port en entrée est possible sans limitations.
- Transition bas/haut : Lors de la validation du registre de sortie, T1 est actif pendant 4 a 5µs, le courant de sortie n'est limité que par la résistance de charge connectée a la masse et peut monter au dessus de 1mA.

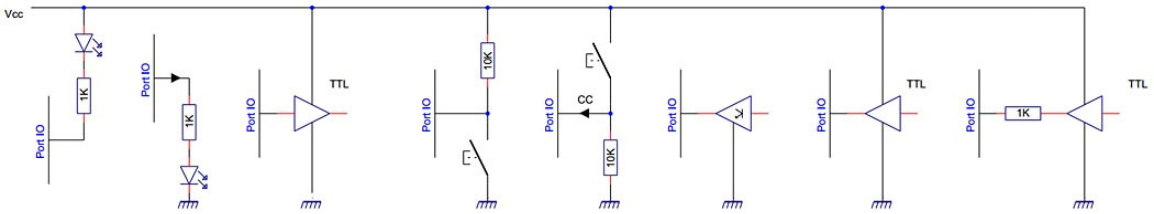


Les circuits connectés aux ports d'entrée sortie de PCF8574 devront donc respecter ces quelques règles et recommandations :

- Utilisation en sortie : Une charge consommatrice de courant comme une Led devra être utilisée avec le commun à Vcc, dans le cas contraire le courant maximum la traversant sera de 0.1mA, insuffisant pour une illumination correcte.
- Utilisation en entrée : Si le port du PCF est placé au niveau bas (T3 actif) et le circuit de commande associé au niveau haut le courant de sortie dépassera les limites permises. Le même cas de figure se retrouvera de manière fugitive pendant 4µs si ce circuit est au niveau bas pendant la transition bas-haut du port (T1 actif).

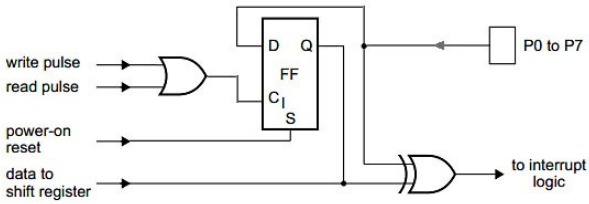
  Si le port est au niveau haut le générateur de courant de T2 limite le courant de sortie a la valeur de 100µA ce qui enlève toute restriction sur ce circuit de commande.

  Malgré que par défaut les huit ports sont mis à l'état haut a la mise sous tension, il sera préférable de toujours insérer une résistance de limitation avec un circuit délivrant un signal pour pallier à un changement de cet état, intentionnel ou non.

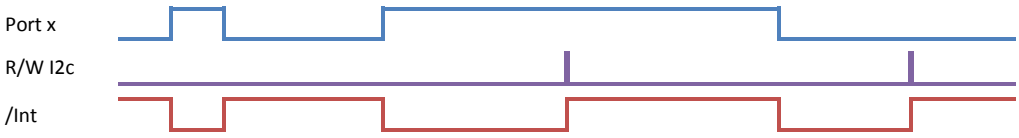


Sortie Int

La sortie /Int est prévue pour être connectée a une entrée d'interruption du microcontrôleur maitre et lui informer d'un changement d'état d'un des ports E/S sans sollicitation du bus I²C.



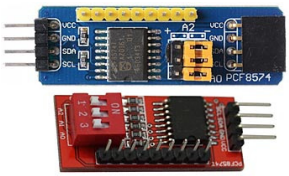
Pour ce, une comparaison entre la valeur du port lue et une valeur mémorisée lors d'un accès en lecture ou en écriture précédente via le bus I²C est effectuée, toute différence provoque alors la mise a l'état bas de la sortie /Int. Cette sortie peut éviter un polling régulier du ou des circuits 8574 par le processeur maitre et assurer un temps de réaction a un événement minimal.



Caractéristiques électriques principales

		min	typ	max
PCF8574	Tension d'alimentation Vcc	2.5v		6v
Bus I²C	Fréquence bus I2c		Standard	100kHz
	Capacité entrée Cp / Courant d'entrée bas I_{ol}	7pF / 3mA		
Ports IO	Courant sortie niveau bas I_{oL}	10mA	25mA	
	Courant sortie niveau haut I_{oh}	30µA		300µA
	Courant sortie niv. haut durant validation I_{oht}		1mA	

Modules commerciaux chinois



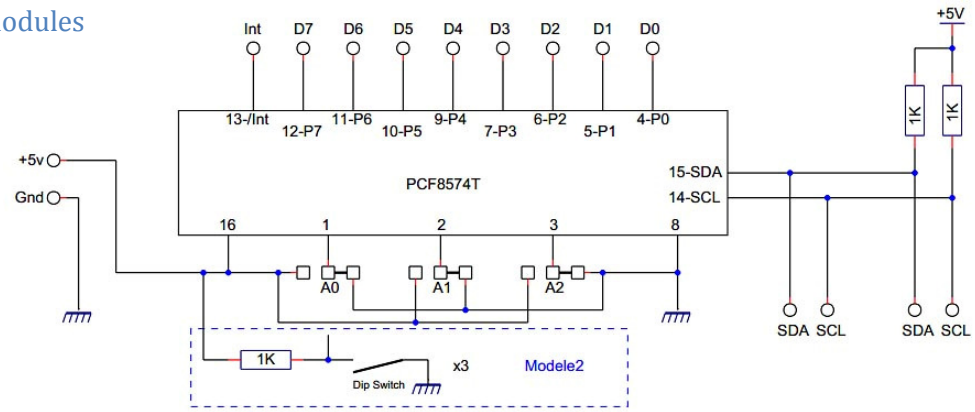
Ces petit modules existent en deux formes couramment distribuées par les fournisseurs chinois se différentiant uniquement par leur système de codage d'adressage (l'un par interrupteurs dip switch, l'autre par cavaliers) et par la présence d'un second connecteur permettant un empilement des modules sur le modèle bleu (Il pourra être alors nécessaire de supprimer les résistances de tirage de 1k ohms sur une partie des modules). L'ordre d'attribution des ports sur les connecteurs de liaison reste identique.

Le tableau ci contre donnera les adresses à utiliser pour accéder aux modules en fonction de la position des interrupteurs ou cavaliers de codage et du type de circuit employé. Généralement ce sera la version 8574 qui sera employée sur ces modules.

Inters	Ponts	A2-A1-A0	PCF8574	PF8574A
▬	L L L	000	0x20 - 32	0x38 - 56
▄▬	L L H	001	0x21 - 33	0x39 - 57
▬▄	L H L	010	0x22 - 34	0x3A - 58
▄▄	L H H	011	0x23 - 35	0x3B - 59

Inters	Ponts	A2-A1-A0	PCF8574	PF8574A
▀▬	H L L	100	0x24 - 36	0x3C - 60
▀▄	H L H	101	0x25 - 37	0x3D - 61
▀▀	H H L	110	0x26 - 38	0x3E - 62
▬	H H H	111	0x27 - 39	0x3F - 63

Schéma modules



Exemples d'utilisation Arduino

Le registre des ports E/S étant d'accès direct et la seule valeur configurable des PCF857x les opérations de lecture ou d'écriture des entrées sorties se résument a leur simple expression et ne posent pas de problème majeur. Si des bibliothèques spécialisées dans la commande de ce circuit intégré existent leur intérêt reste limité du fait de cette grande facilité d'emploi.

Les petits montages suivant offrent des exemples d'emploi basique de ce circuit intégré et de ce qu'il est possible de réaliser avec.

Affichage Led 7 segments 2 digits

Plus pour l'exemple et pour découvrir ce circuit que pour une utilisation pratique des circuits spécialisés assurant cette fonction de façon moins gourmande en ressources processeur et plus efficace. Les 8 E/S du PCF8574 ne permettent au choix que soit la commande d'un digit + le point décimal d'un afficheur a Led 7 segments, soit de deux digits multiplexés sans l'utilisation du dp.

Le programme et le montage suivant proposent une solution à cette seconde proposition avec le principe de fonctionnement suivant.

Les 7 segments de l'afficheur double sont commandées par la mise a l'état bas (obligatoire dans le cas du PCF) des ports 0 a 6 impliquant alors l'utilisation d'un afficheur a anode commune. Le tableau de valeurs constantes Segment [] donne les valeurs à utiliser par ces ports pour afficher les valeurs hexadécimales 0 a F.

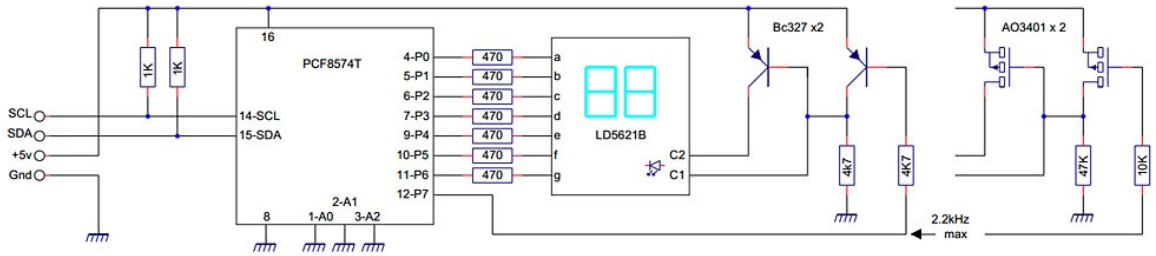
La sélection du digit allumé est réalisée par le port 7 commutant un des deux transistors Pnp (ou de préférence des mosfets canal P). Ce port au niveau bas allumera les segments du digit des unités, un niveau haut celui des dizaines.

Plutôt que d'utiliser des fonctions `delay()` bloquant le processus, un pseudo système de temps partagé utilisant l'horloge interne de l'Arduino permet de modifier la valeur affichée toutes les 500ms, et de gérer le multiplexage des deux digits l'afficheur. Pour ce, les deux variables `CptValeur` et `CptDigit` sont comparées à la valeur de l'horloge système `ValTemsp`, une différence dépassant un seuil prédéfini déclenchant l'événement associé.

Pour éviter les phénomènes de flicker le basculement entre les deux digits devra être réalisé au minimum toutes les 7 à 10ms (début apparition du phénomène). En l'absence de temporisation la durée du bloc de commandes I²c d'écriture en direction des ports est de 230µs ce qui permet une fréquence de multiplexage de 2.2khz max. Le temps utilisable pour le reste du programme dans la boucle principale sera donc relativement faible pour un fonctionnement sans scintillement de l'affichage avec des restrictions importantes au niveau de l'utilisation des temporisations.

La sélection des données envoyées a l'afficheur est gérée par la variable `Digit` avec l'utilisation dans la table Segment [] des 4 bits inferieurs de la valeur à afficher avec le port7=0, et des 4 bits supérieurs et le port7=1 (valeur par défaut dans la table de transcodage).

Schéma de câblage



Listing Arduino

```
#include <Wire.h>

const uint8_t Segment[16]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,
   0x83,0xC6,0xA1,0x86,0x8E};                    // Table transcodage BCD-7segment
uint8_t Digit;

uint32_t ValTemps;                               // Variables gestion temps
uint32_t CptValeur;
uint32_t CptDigit;
```

```
uint8_t Valeur;                                   // Valeur a afficher
uint8_t DataT;                                    // Temporaire de calcul GPIO

void setup() {   // ------------------------------------------------------------Init
  Wire.begin();
}

void loop() {  // ------------------------------------------------ Boucle principale

  ValTemps=millis();

  if (ValTemps-CptValeur > 500){   // --------------------Incrément valeur a afficher
    CptValeur=ValTemps;
    Valeur++;

  if (ValTemps-CptDigit > 5){  // -------------------------------Multiplexage digits
    CptDigit=ValTemps;

    if (Digit==0) {                               // Digit .0
      DataT=Segment[Valeur & 0x0F]& 0x7F;
    }
    else {
      DataT=Segment[(Valeur & 0xF0) >> 4] ;       // Digit 1.
    }
    Digit=~Digit;

    Wire.beginTransmission(0x27);                 // Ouverture transaction I²C
    Wire.write(DataT);                            // Envoi data
    Wire.endTransmission();
  }

  // Code utilisateur à insérer

}
```

Clavier 16 touches multiplexé X-Y

Une classique matrice de touches 4x4 est utilisée, les ports 0 a 3 du PCF sont utilisées en entrées et reçoivent si une touche est actionné le scan code de niveau bas généré par un des ports 4 a 7. Les résistances optionnelles de 1K ohms ne sont normalement pas nécessaires, un seul des ports étant utilisé en sortie à la fois.

Dans cette exemple la routine d'analyse du clavier est executée 10 fois par secondes, dans un autre contexte elle pourrait egalement fair l'objet d'une fonctio et etre appellée uniquement a la demande. L'utilisation de la sortie /Int du PCF8574 ne peut pas fonctionner le scan des colonnes n'etant actif qu'avec cette routine.

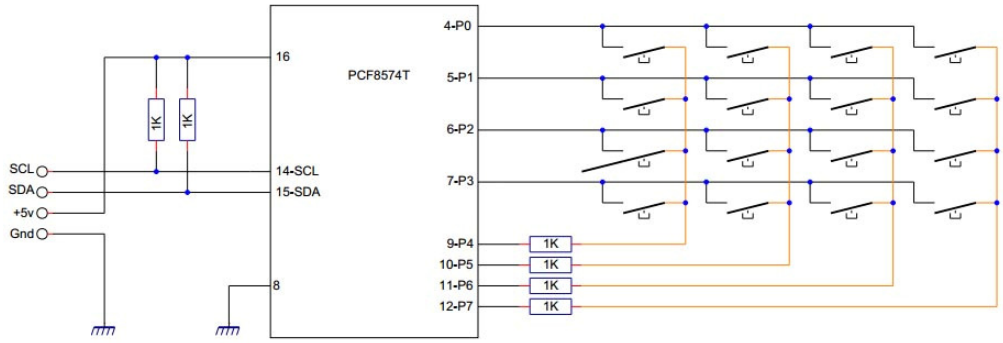
Les ports P4 a P7 sont successivement mis a l'etat bas grace au decalage circulaire de la variable CptScan, les lignes du clavier sont alors lues via les ports P0 a P3, le resultat obtenu mis en mémoire dans les 4 bits bas de la variable ValClavier laquelle est decalée de 4 bits pour recevoir la ligne suivante.

La variable 16bits ValClavier contient le résultat de l'analyse du clavier, chaque bit au niveau 1 correspond a la touche correspondante active, une ou plusieurs d'entre elle pouvant bien sur l'être simultanément sans problème.

Masque ValClavier	0x0001	0x0002	0x0004	0x0008	0x0010	0x0020	0x0040	0x0080
Touche associée	Lg1-Col1	Lg2-Col1	Lg3-Col1	Lg4-Col1	Lg1-Col2	Lg2-Col2	Lg3-Col2	Lg4-Col2
Masque ValClavier	0x0100	0x0200	0x0400	0x0800	0x1000	0x2000	0x4000	0x8000
Touche associée	Lg1-Col	Lg2-Col3	Lg3-Col3	Lg4-Col3	Lg1-Col4	Lg2-Col4	Lg3-Col4	Lg4-Col4

A titre indicatif la durée de la routine ScanClavier est d'environ 1200µs dans le mode de transfert I²C standard du PCF8574.

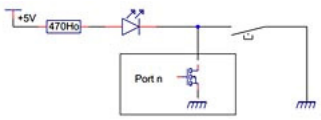
Schéma de câblage



Listing Arduino

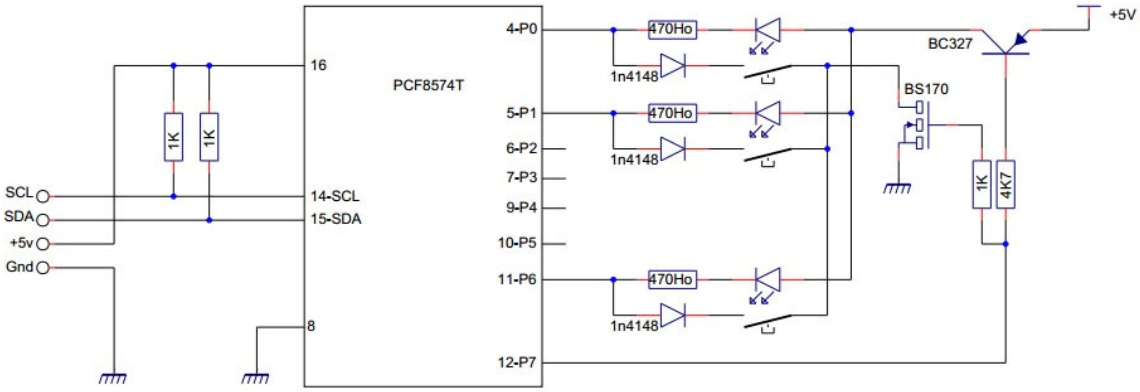
```
#include <Wire.h>
const uint8_t AddPCF1 =0x27;                          // Declaration Adresse PCF8574

uint32_t ValTemps;

uint32_t CptClavier;                                  // Variables Scan clavier
uint16_t ValClavier=0;


void setup() {  // ------------------------------------------------- Initialisation
   Wire.begin();
   Serial.begin(9600);
}

void loop() {  // ---------------------------------------------------- Boucle principale
  ValTemps=millis();

  if (ValTemps-CptClavier > 100){                     // Exécuté toutes les 100ms
    CptClavier=ValTemps;

    uint8_t CptScan=0x80;    // ------------------------------------ Scan clavier
    ValClavier=0;

    while(CptScan !=0x08) {
      Wire.beginTransmission(AddPCF1);                // Mise a l'état bas colonne
      Wire.write(~CptScan);
      Wire.endTransmission();

      Wire.requestFrom(AddPCF1,1);                    // Lecture ligne
      while (Wire.available()) {
        ValClavier=ValClavier << 4;                   // Décalage et enregistrement
        ValClavier |=~Wire.read() & 0x0F;             // lecture clavier
      }

      CptScan = CptScan >> 1;
    }

    Serial.println(ValClavier,HEX);                   // Affichage résultat
  }

}
```

La encore des circuits spécialisés réaliseront cette opération beaucoup plus efficacement. Si il est tout a fait possible de repartir les 8 ports en entrées reliées a des détecteurs OU en sorties commandant des Leds un système de multiplexage relativement simple similaire a celui utilisé précédemment permet d'obtenir 12 entrées ET sorties soit plus que ce que permet un PCF8575.

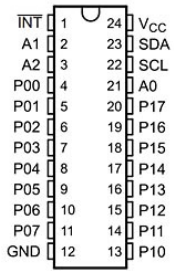


La conception des ports du PCF8574 font qu'une Led en mode sortie ou une touche en mode entrée sont toutes les deux actives port au niveau bas impliquant un montage similaire au schéma ci-contre. Pour éviter que l'action sur une touche puisse allumer la Led une solution peut être d'activer les Leds ou le clavier alternativement en fonction des besoins. Toujours en comptant sur la persistance rétinienne il est alors possible d'inhiber l'affichage le temps de lecture des touches du clavier sans que cela soit visible. Dans le cas des PCF857x une opération d'écriture puis de lecture des ports durant environ 450µs, une répétition de cette séquence pour un rafraichissement du clavier 10 fois par seconde n'est pratiquement pas discernable sur les Leds allumées.

Schéma de câblage

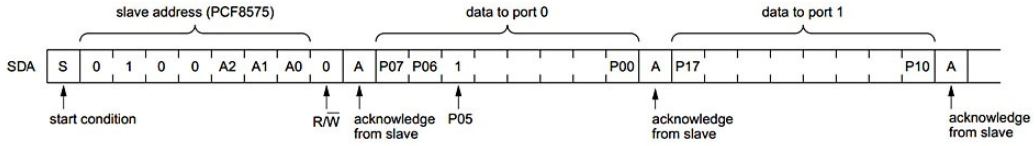


Buffers 16 ports E/S : PCF8575



Le PCF8575 est une version doublée du 8574, intégrant deux blocs de 8 ports séparés. La structure, les caractéristiques des ports d'entres sorties tout comme le système d'adressage sont strictement identiques au modèle simple décrit dans le chapitre précédent.

L'accès aux registres de ports E/S est réalisé par l'emploi d'un couple de données suivant l'adresse du périphérique I²C. Ces deux octets représenteront respectivement dans l'ordre les valeurs des ports 0 à 7 et 10 à 17 (Dénomination des ports effectué par le constructeur sous la forme NoBloc-NoPort).

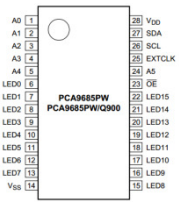


La commande de ce circuit par un système Arduino pourrait donc être réalisée sous cette forme. Si dans l'absolu toute opération uniquement sur le bloc pourrait être effectuée en utilisant qu'un seul octet un bit de START remettant les compteurs interne du PCF a zéro cette manière de procéder ne serait pas trop orthodoxe et a éviter.

```
//Version une variable 8bits par bloc de ports

uint8_t ValB0=0xFF;                                 // Valeurs internes ports
uint8_t ValB1=0xFF;
uint8_t Add8575=0x20;                               // Adresse PCF8575


void Send8575() {                                   // Ecriture PCF8575
  Wire.beginTransmission(Add8575);
  Wire.write(ValB0);                                // Ports 0 a 7 (bits 0 a 7)
  Wire.write(ValB1);                                // Ports 10 a 17 (bits 8 a 15)
  Wire.endTransmission();
}

void Read8575() {                                   // Lecture PCF8575
  if (Wire.requestfrom(Add8575,2)==2) {
    ValB0=Wire.read();                              // Validation résultats si
    ValB1=Wire.read();                              // opération ok.
  }
}
```

```
//Version une variable 16bits

Uint16_t ValPorts=0xFFFF;                           // Valeur interne ports
uint8_t Add8575=0x20;                               // Adresse PCF8575

void Send8575() {                                   // Ecriture PCF8575
  Wire.beginTransmission(Add8575);
  Wire.write(ValPort &00FF);                        // Ports 0 a 7 (bits 0 a 7)
  Wire.write((ValPort & 0FF00) >> 8 );              // Ports 10 a 17 (bits 8 a 15)
  Wire.endTransmission();
}

void Read8575() {                                   // Lecture PCF8575
  if (Wire.requestfrom(Add8575,2)==2) {
    ValPort=Wire.read();                            // Validation résultats si
    ValPort |=Wire.read() << 8;                     // opération ok.
  }
}
```

Driver de Leds PWM 16 canaux : PCA9635 - PCA9685



Le PCA9685 remplaçant du 9635 est un prévu a l'origine pour commander en mode modulation de largeur d'impulsion (PWM) des retro-éclairages et des panneaux de Leds RGBA avec 4 canaux par couleurs soit 16 canaux individuels. Ce circuit peut bien sur être utilisé avec n'importe quel autre type d'application utilisant des commandes PWM comme des servo moteurs par exemple.

	Réglages individuels	Résolution	Fréquence	Rapport cyclique	tOn - tOff séparés	Led On -Off	Synchro externe	Clignotement Leds	Etat Led après reset
9635	Oui	8 bits	97khz	0-99%	Non	Oui	Non	40ms-4s	1
9685	Oui	12 bits	40Hz-1kHz	0-100%	Oui	Oui	Oui		0

Chaque sortie Led est constituée de deux transistors montés en totem pole, ayant les limites suivantes :

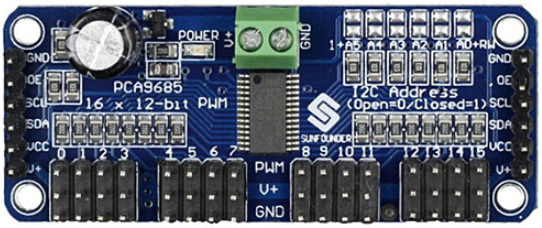
- Le transistor à la masse en drain ouvert permet au maximum un courant de 25mA sur une Charge connectée alimentée par une tension pouvant être différente de celle du circuit (plage d'alimentation 3v3-5v) mais ne pouvant dépasser 5.5v.
- Le transistor situé à Vcc ne peut fournir qu'un courant de 10ma dans une charge connectée a la masse.
- Le courant global circulant dans le circuit (communs Vcc ou Vss) global ne doit pas dépasser 400mA.

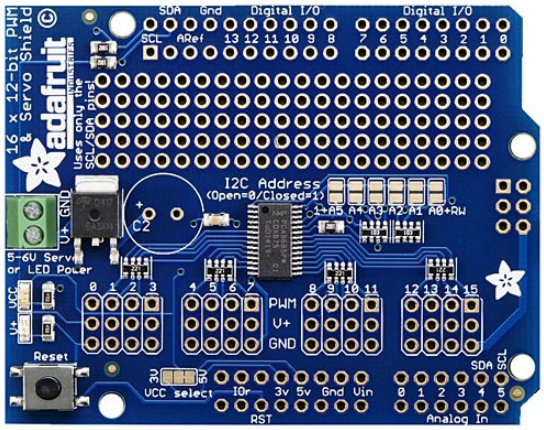
Une entrée /Oe permet mise a l'état haut de désactiver l'ensemble des sorties, celles-ci passant au niveau sélectionné dans le registre de configuration Mode2.

L'entrée d'horloge externe permet de synchroniser plusieurs circuits entre eux pour eviter les phénomènes de papillonnement, cette entrée remplace l'horloge interne 25Mhz et permet d'injecter un signal 50Mhz au maximum.

Modules commerciaux

Les modules a basés sur de PCF9685 se trouvent généralement sous forme de cartes individuelles, Adafruit propose un shield au format standard s'enfichant directement sur la carte CPU.

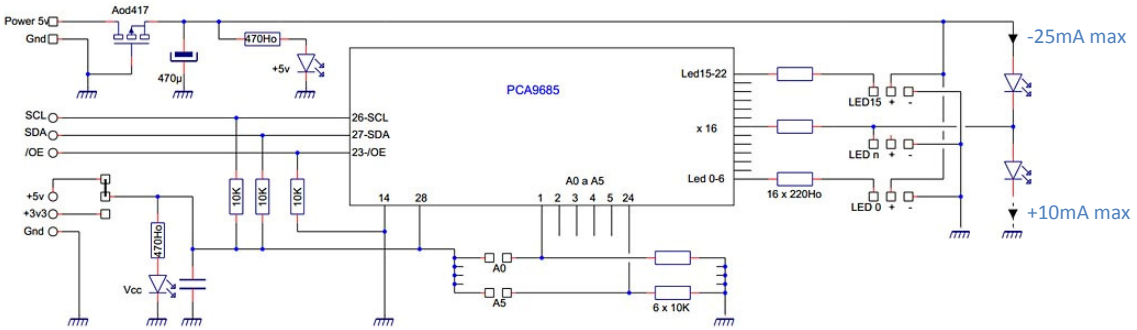




Ces deux ensembles sont globalement similaires, les 16 sorties PWM son disponibles sur des connecteurs 3 points avec le +5v et la masse, la résistance de limitation de 220 ohms en serie avec les sorties sera suffisante pour alimenter une Led connectée au +5v mais sera un peu juste sortie utilisé en source avec la Led connectée a la masse. En raison du courant maximum de 400mA que peut absorber le driver une alimentation est proposés sur un connecteur a vis, un mosfet canal P est mis en protection en cas d'inversion des polarités des fils de cette alimentation externe.

Les niveaux des six entrées d'adresse sont modifiables par soudure de ponts, par défaut elles sont tirées au niveau 0, soit une adresse I²C égale à 0x40.

Schéma shield



Fonctionnement interface I²C

Adressage et dialogue I²C

Le PCA9685 dispose de 6 entrées de sélection d'adresse de dialogue I²C permettant de choisir celle-ci sur la plage 0x40 a 0x7F, a cela s'ajoute plusieurs adresses spécifiques permettant un accès en écriture simultané a un groupe ou l'ensemble des circuits connectés.

Reset	0x03	Permet de réinitialiser l'ensemble des circuits du bus
Led All Call Add	0x70	Valeur par défaut configurée dans le registre 0x05 AllCallAdr
Led Sub Call Add 1	0x71	" 0x02 SubAdr1
"	0x72	" 0x03 SubAdr2
Led Sub Call Add 3	0x74	" 0x04 SubAdr3

Les paramètres de fonctionnement du circuit sont enregistrés dans environ 70 registres dont l'accès est classiquement assuré par un pointeur, auto-incrémenté si l'option AI est activée. La lecture ou la modification d'un des registres de travail devra donc toujours être précédé par la mise à jour de ce pointeur. Le tableau suivant donne quelques exemples de trames I²C résultante.

Op. Lecture registre Mode1 adresse 0x00	Add PCA (0x40) write = 0x80	Pointeur = 00		
	Add PCA (0x40) read = 0x81	Valeur Reg		
Op. Lecture paramètres Off Led1 (Add 0x0C-0D)	Add PCA (0x40) write = 0x80	Pointeur = 0C		
	Add PCA (0x40) read = 0x81	Val. Reg 0C	Val. Reg 0D	
Op. Ecriture registre Mode2 adresse 0x01	Add PCA (0x40) write = 0x80	Pointeur = 01	Valeur Reg	
Op. Ecriture paramètres On Led3 (Add 0x12-13)	Add PCA (0x40) write = 0x80	Pointeur=0x12	Val Reg12	Val Reg13

Registres interne

Les registres du PCF peuvent être répertoriés en deux catégories, les registres de configuration globale, et les registres de paramètre PWM des Leds groupés par 4 octets. Le tableau suivant donne leu valeur d'adresse à utiliser avec le pointeur.

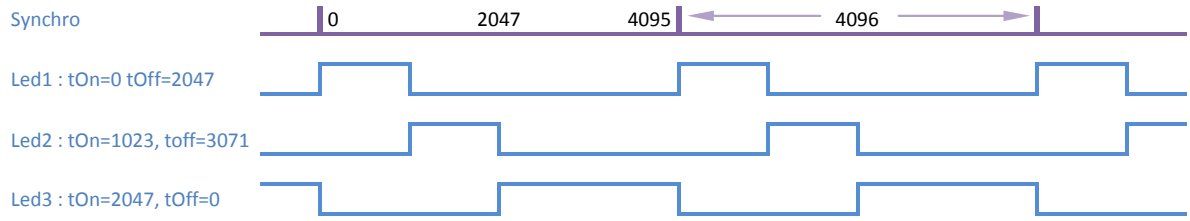
Add Reg. Dec.	Hex	Nom	Descriptif	Valeur par Défaut
00	00	Mode1	Registre de configuration 1	0x11 - b0001 0001
01	01	Mode2	Registre de configuration 2	0x04 - b0000 0100
02	02	SubAdr1	Adresse I2C sur 8bits du groupe N°1 de PCA9685	0xE2 - b1110 001**x**
03	03	SubAdr2	Idem groupe N°2	0xE4 - b1110 010**x**
04	04	SubAdr3	Idem groupe N°3	0xE8 - b1110 100**x**
05	05	AllCallAdr	Idem ensemble des circuits du bus	0xE0 - b1110 000x
06	06	Led 0_ON_L	Valeurs d'enclenchement et de déclenchement sur 2 octets de la sortie Led0	0x00 - b0000 0000
07	07	Led 0_ON_H		0x00 - b0000 0000
08	08	Led 0_OFF_L		0x00 - b0000 0000

09	09	Led 0_OFF_H		0x10 - b0001 0000
10	0A	Led 1_ON_L	Valeurs d'enclenchement et de déclenchement sur 2 octets de la sortie Led 1	
11	0B	Led 1_ON_H		
12	0C	Led 1_OFF_L		
13	0D	Led 1_OFF_H		
14	0E	Led 2_ON_L	……	
…	…	…..	A suivre ensemble des Leds	
65	41	Led 15_OFF_H	……	
			Non utilisés	
250	FA	All Led_ON_L	Valeurs d'enclenchement et de déclenchement sur 2 octets de l'ensemble des LEDS 0 à 15.	0x00 - b0000 0000
251	FB	All Led_ON_H		0x10 - b0001 0000
252	FC	All Led_OFF_L		0x00 - b0000 0000
253	FD	All Led_OFF_H		0x10 - b0001 0000
254	FE	Prescale	Fréquence de travail PWM	0x1E - b0001 1110
255	FF	TestMode	Test constructeur I/O	

Fonctionnement et paramètres PWM

Le mode de fonctionnement du PWM est extrêmement simple et permet toutes les combinaisons y compris des déphasages complexes entre plusieurs sorties. Un top synchro commun a l'ensemble du système est émis a la fréquence définie par le registre Prescale, la période de ce signal est divisé par 4096 par un compteur, les valeurs des registres Ledx ON et Ledx OFF provoquent le basculement de la sortie considérée quand leur valeur est égale a celle du compteur.

Dans l'exemple ci-dessous, les Led1 et 2 auront le même éclairement à ¼ de leur puissance mais déphasé de 90°, la Led3 éclairera elle a mi puissance.



Les valeurs utilisées étant sur 12 bits deux registres d'un octet sont utilisés en mode little-endian, octet de poids faible en premier. Le bit 4 du second registre mis à 1 permet d'obtenir l'état haut ou bas permanent de la sortie, les bits supérieurs 7 à 5 de ce registre ne sont pas utilisés.

	Registre H								Registre L							
	7	6	6	4	3	2	1	0	7	6	5	4	3	2	1	0
Registres 1-0 Ledx ON	0	0	0	0/1	Valeur compteur mise a 1 sortie : 0 - 4095											
				Permanent ON = 4096												
Registres 3-2 Ledx OFF	0	0	0	0/1	Valeur compteur mise a 0 sortie : 0 - 4095											
				Permanent OFF = 4096												

La fréquence du top synchro est régie par la valeur contenue dans le registre Prescale situé a l'adresse 0xFE, cette valeur pouvant varier de 2 a 255. La formule de calcul de cette valeur fournie par le constructeur est :

$$Val\ Prescale = \left(\frac{f\ Ocillateur}{4096 * f\ PWM}\right) - 1$$

Avec la fréquence d'oscillateur interne de 25Mhz utilisée par défaut il est possible d'en tirer les équations simplifiées suivantes :

$$Val\ Prescale = \left(\frac{6103.5}{f\ PWM}\right) - 1 \qquad f\ PWM = \left(\frac{6103.5}{Val\ Prescale + 1}\right)$$

Toujours avec l'oscillateur interne la valeur 0x1E utilisée par défaut à l'initialisation du circuit génère une fréquence PWM de 200Hz, et la plage de réglage est de 24 à 1500 Hz.

Registres de configuration

Le premier registre de configuration situé à l'adresse 0 gère les paramètres de fonctionnement généraux du circuit, chaque bit ayant les fonctions suivantes. Dans la majorité des cas la première opération sera de basculer le bit 5 pour valider l'auto incrémentation du pointeur d'adresse de registre, la prise en compte est immédiate, l'ecriture de ce bit sur le registre Mode1 entrainera l'accès au registre suivant dans la même trame I²C.

bit	Fonctions Mode 1	0	1
7	Restart mode	Non	Oui
6	ExtClk : Horloge interne 25Mhz ou horloge externe pin 25	**Interne**	Externe
5	AI : Incrémentation auto du pointeur d'adresse registre	**Non**	Oui
4	Sleep : Mode veille, oscillateur et sorties off, modification de certains registres inactif	Non	**Oui**
3	Sub1 : Réponse a une adresse I²C égale à celle définie dans le registre Sub1Adr	**Non**	Oui
2	Sub2 : " " Sub2Adr	**Non**	Oui
1	Sub2 : " " Sub3Adr	**Non**	Oui
0	AllCall : " " AllCallAdr	Non	**Oui**

Le second registre Mode2 s'occupe plus généralement de la configuration des drivers des sorties Led0 a Led15.

bit	Fonctions Mode 2	0	1
7-5	Réservés	0	
4	INVRT : Inversion logique des sorties vis-à-vis du signal PWM (uniquement avec /OE inactif)	**Non**	Oui
3	OCH : Les changements sont appliqués a la réception des signaux I²C Ack (chgt immédiat) ou Stop a la fin de réception de la trame complète (chgt global)	**Stop**	Ack
2	OUTDRV : Type de sortie, totem pole ou drain ouvert (transistor Vcc inactif)	Drain Ouvert	**Totem P.**
1	OUTNE1 : Si l'entrée /OE est active les sorties LED sont a l'état haute impédance	**OUTNE0**	HI
0	OUTNE0 : " " indiqué par ce bit	**Bas**	Haut

Les valeurs en gras sont celles par défaut.

Librairie Arduino

N'ayant pas utilisé ce produit, je ne donnerais que des informations succinctes sur la partie logicielle, malgré que le circuit ne soit pas difficile a utiliser il est toujours possible d'utiliser la librairie développée par la société AdaFruit pour ses interfaces.

La documentation matérielle peut se trouver à ces adresses :

https://www.adafruit.com/product/815
https://www.adafruit.com/product/1411

Et la librairie sous licence BSD est librement téléchargeable ici

https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

Les principale fonctions et méthodes de cette librairie sont :

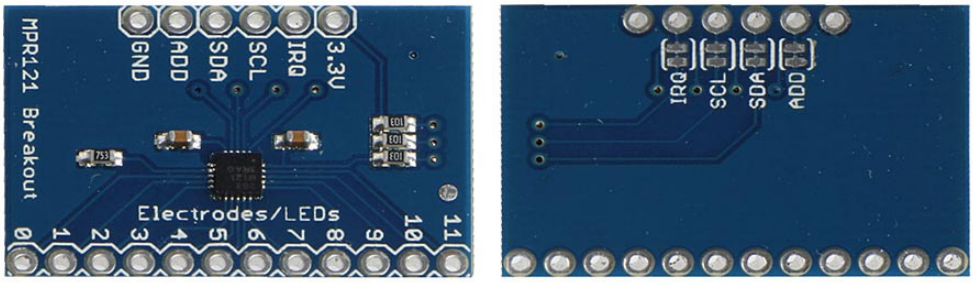
- `#include <PWMServoDriver.h>` : Déclaration de compilation classique à insérer après celle de la bibliothèque Wire en tête de programme.
- `PWMServoDriver nomPCA(Add)` : Déclaration de l'instance du circuit avec le nom utilisateur nomPCA situé a l'adresse Add, si celle est omise la valeur 0x40 est employée par défaut.

- `nomPCA.begin();` : Initialisation globale = `Wire.begin() + nomPCA.Reset()`
- `nomPCA.reset();` : Registre Mode1=0

- `nomPCA.setPWMFreq(f0);` : Réglage fréquence dans le registre Prescale avec la valeur fournie `f0` en Hertz.

- `nomPCA.set(Canal,tOn,tOff)` : Modification des valeurs de consigne d'enclenchement et de declenchement de la sortie du canal spécifié avec `Canal` uint8_t de 0 a 15, et `tOn-tOff` uint16_t de 0 a 4095.
- `nomPCA`.setPin(Canal,ValOn,Invert) : Version simplifiée de `.set`, `ValOn` uint16_t de 0 a 4096 représente le pourcentage du temps ou la sortie est active (0=sortie 100% off, 4096=sortie 100% on), `Invert` est un boolean indiquant qu'il faut inverser le signal.

Contrôleur de clavier XY capacitif MPR121



http://www.nxp.com/assets/documents/data/en/application-notes/AN3894.pdf

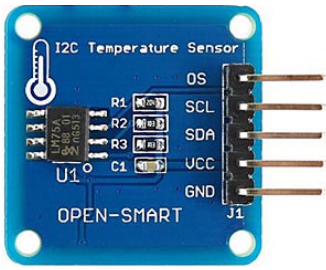
http://www.nxp.com/assets/documents/data/en/application-notes/AN4600.pdf

Sondes et divers analogique

Convertisseur analogique numérique ADS1114/1115

Potentiomètres numériques

Sonde de température LM75



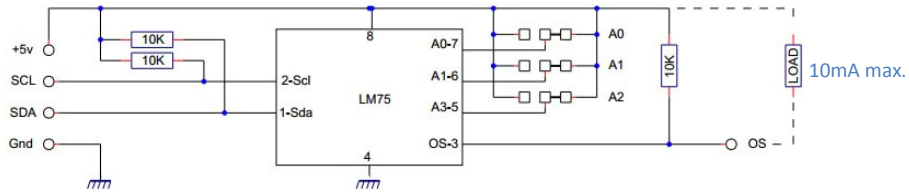
Le LM75 est un capteur de température fonctionnant sans sonde externe sur la plage -55/+125° avec une précision inferieure au degré sur la plage de température ambiante et une erreur garantie inferieure a 2°. La température mesurée est donnée sous la forme d'une valeur 9 bits signés, soit une résolution de 0.5°.

Un thermostat programmable est disponible avec la sortie en collecteur ouvert OS, celle est au activée si la temperature mesurée dépasse le seuil tOs, et désactivée si elle repasse sous le seuil tHy (hysteresis = tOs - tHy). Le fonctionnement de ce thermostat est indépendant de toute connexion ou dialogue sur le bus I²C.

Pour rappel les valeurs de la temperature mesurée, de tOs ou THy étant sous la forme d'une valeur 9bits signés, le bit 9 a l'état 1 signifie des températures négatives, la conversion peut être effectuée avec le calcul tneg = val int9_t - 512 en décimal.

t ° mesurée	bin	Dec. signé	Hex. Signé	Dec brut	Hex brut
+125	0 1111 1010	250	0xFA	250	0x0FA
0	0 0000 0000	0	0x00	0	0
-0.5	1 1111 1111	-1	- 0x01	511	0x1FF
-55	1 1001 0010	-110	- 0x6E	402	0x192

Schéma module



Fonctionnement interface I²C

Les LM75 disposant de 3 entrées de sélection, l'adresse I²C à utiliser pour y accéder dépendra des niveaux qui leur seront appliqués, le tableau suivant indique les valeurs hexadécimales correspondantes.

A2 - A1 - A0	000	001	010	011	100	101	110	111
Add hex	48	49	4A	4B	4C	4D	4E	4F

L'acces aux valeurs du LM75 est classiquement assurée par 4 registres sélectionnés par un pointeur qui devra être positionné sur la bonne valeur au moins une fois avant tout accès. Ce pointeur n' a que 2bits d'actifs les autres devant être mis impérativement laissés a zéro.

RegSel	Registres LM75	Acces	Longueur
0x00	Temperature mesurée	Lecture	2 octets
0x01	Configuration	RW	1 octet
0x02	tHy - Temperature coupure thermostat sortie OS (**75°C**)	RW	2 octets
0x03	tOs - Temperature enclenchement thermostat (**80°C**)	RW	2 octets

Si les valeurs de températures sont transferées sur deux octets elles utilisent les 9 bits hauts de ceux-ci, cette particularité permet de séparer les unités sur l'octet supérieur et les demi-degrés sur l'inferieur.

	Premier octet	Second octet	val t°	
	b8-b7- ….. -b2-b1	b0-0-0-0 - 0-0-0-0	T,0	0,t
Exemple 20.5° (=0x29)	0001 0100	1000 0000	20	0.5

Le registre de configuration gère les modes de fonctionnement du LM75 et du thermostat avec les options suivantes :

RegConf	Fonction bits
bits 7 à 5	Non utilisés (tests constructeurs) = 0
bits 4-3	Compteur d'erreurs du convertisseur AD de température : 0=normal
bit 2	Polarité sortie OS si thermostat actif (t > tOs) **: 0=état bas** (transistor on), 1=état haut (si résistance pull up)
bit 1	Mode action Os a l'enclenchement thermostat : **0=continu** tout ou rien, 1=Impulsion seule pour interruption.
bit 0	Veille LM75 : **0=LM75 actif**, 1= En veille, conversion température a l'arrêt.

Par défaut a la mise sous tension du circuit les valeurs en gras sont utilisées.

Dialogue I²C

L'accces aux differents registres etant conditionnés a la valeur du pointeur RegSel celui-ci devra obligatoirement etre ecrit en premier lors d'une operation d'ecriture sur le registre de configuration ou des valeurs de thermostat. En revanche lors d'une operation de lecture de valeurs l'ecriture prealable de RegSel pourra etre omise si il est déjà bien positionné.

Op. Lecture registre de configuration	Add PCF (0x48) write = 0x90	RegSel = 01		
	Add PCF (0x48) read = 0x91	RegConf		
Op. Lecture température mesurée	Add PCF (0x48) write = 0x90	RegSel = 00		
	Add PCF (0x48) read = 0x91	t° unité	t° decimal	
Op. Relecture température mesurée	Add PCF (0x48) read = 0x91	t° unité	t° decimal	
Op. Ecriture registre de configuration	Add PCF (0x48) write = 0x90	RegSel = 01	RegConf	
Op. Ecriture température thermostat tOs	Add PCF (0x48) write = 0x90	RegSel = 03	tOs unité	tOs decimal

Utilisation Arduino

Si l'utilisation de ce circuit ne pose pas de problèmes particuliers de nombreuses bibliotheques facilitent le travail dont celle de TheFekete couramment citée et disponible a cette adresse.

https://github.com/thefekete/LM75

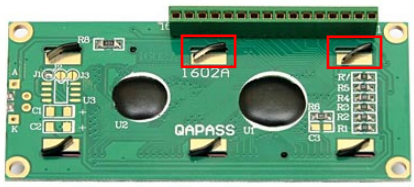
Toutes les valeurs de température fournies ou utilisées par la bibliothèque sont en degrés C au format flottant sur 4 octets. Aucun contrôle de dépassement des limites de mesure du LM75 ou de cohérence tOs > thyst n'est effectué par la bibliothèque.

- `#include <LM75.h>` : Déclaration de compilation classique à insérer après celle de la bibliothèque Wire en tête de programme.
- `LM75 NomCapteur;` : Initialisation d'une l'instance de la bibliothèque "NomCapteur" avec l'adresse LM75 par défaut 0x48 (constante LM78_ADRESS).
- `LM75 Nom(Add);` : Idem précédent avec adresse du capteur autre que celle par defaut.
- `RetFloat=NomCap.temp()` : Lecture température mesurée, retour sous la forme de 4 octets au format flottant.
- `RetFloat=NomCap.tos();` : Idem précédent pour la consigne d'enclenchement du thermostat.
- `RetFloat=NomCap.thyst()` : " " de déclenchement " "
- `NomCap.tos(ValFloat);` : Ecriture de la consigne d'enclenchement du thermostat.
- `NomCap.thyst(ValFloat);` : Idem précédent pour la consigne de déclenchement.
- `NomCap.shutdown(Bool);` : Mise en ou hors veille du capteur (true/false).
- `NomCap`.conf(ValByte); : Ecrit le registre de configuration du LM75.

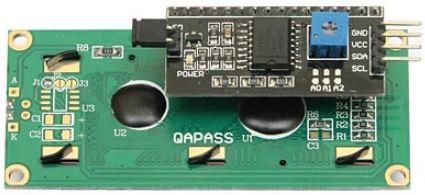
Driver afficheur alphanumérique LCD - Hitachi 44780

Petit module permettant de commander tout affichage LCD disposant d'un connecteur au standard 44780 à partir d'une liaison I2c. Cette solution sur des cartes disposant de peu d'entrées sorties permet d'économiser cinq ports digitaux par rapport à un shield LCD classique.

Le brochage du connecteur de sortie est prévu pour s'adapter a tout afficheur doté d'un retro éclairage, la liaison avec le LCD pourra être effectués soit a l'aide d'un connecteur femelle, soit en soudant directement les broches males au pas de 2.54 mm directement. Dans ce dernier cas attention aux pattes métallique du cadre de maintient du LCD pouvant entrer en contact avec l'adaptateur.



LCD nu avec connecteur fem.



LCD I2c

Basé sur un PCF8574 le module peut être alimenté en 3v3 ou 5v, un cavalier en extrémité de module permet de désactiver le retro-éclairage du LCD. L'autre extrémité reçoit un connecteur male 4 points avec l'alimentation et le bus I2c disposé de façon standard.

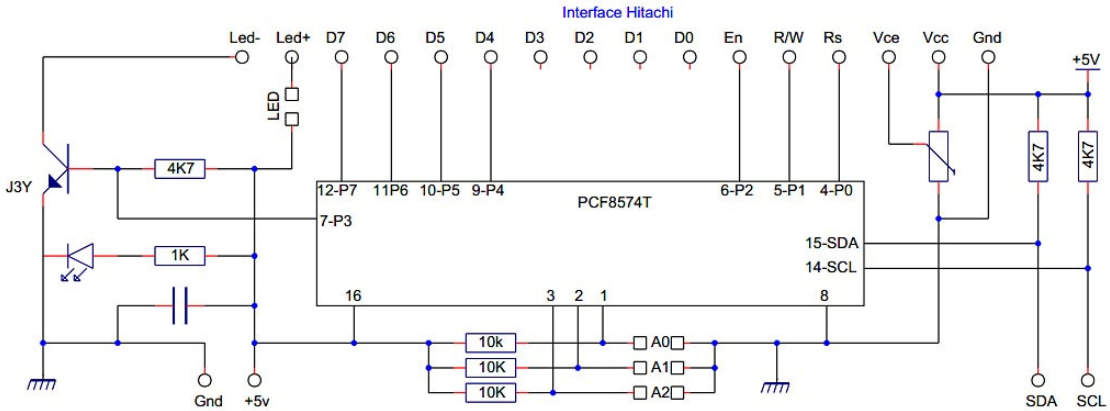
La sélection de l'adressage I2c est effectuée par ponts à souder, la bibliothèque LCD-I2c utilisant uniquement l'adresse globale sans tenir compte du bit0 de selection R/W les valeurs à déclarer en fonction des ponts soudés seront celles du tableau ci-contre, l'adresse 0x27 est celle d'origine par défaut.

Ponts	A2 - A1 - A0	Adresse hex
	111	27
	110	26
	101	25
	100	24

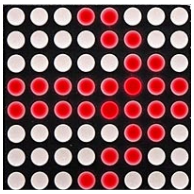
Ponts	A2 - A1 - A0	Adresse hex
	011	23
	010	22
	001	21
	000	20

L'afficheur sera commandé par la bibliothèque *LiquidCrystal-I2c* d'une manière similaire à la bibliothèque LCD standard (Voir le document dédié aux afficheurs Arduino).

Schéma adaptateur



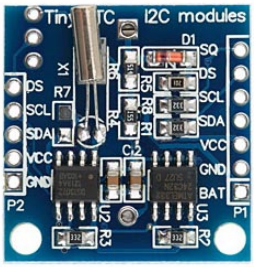
HK16K33 Adafruit 87x : Matrice 8x8 led - HT16K33



Afficheurs Oled matriciels

Horloge temps réel RTC DS1307

Hardware



Le circuit intégré DS1307 est une horloge temps réel basse consommation offrant outre le service d'horodatage 56 octets de mémoire utilisateur de type novram (ram sauvegardée par batterie) et une sortie horloge programmable a collecteur ouvert.

Son oscillateur fonctionnant avec un quartz horlogerie standard et le diviseur associé propose les fréquences de 32.76, 8.2, 4.09 et 1Hz pour piloter cette sortie. L'alimentation interne intègre les fonctions de détection d'absence tension et de gestion de la pile de sauvegarde avec les caractéristiques suivantes :

	min	Typ	max
Tension d'alimentation Vcc - pin8 (v)	4.5	5	5.5
Coant d'alimentation principal		<500nA	<500nA
Tension entrée batterie de sauvegarde Vbat - pin3	2v	3v	3.5v
Consommation circuit batterie oscillateur horloge en fonction			<500nA

Ce circuit se trouve couramment utilisé sur les petits modules se trouvant pour moins de 2€ dont la photo se trouve ci-dessus comportent en plus de cette horloge une mémoire EEprom de 32ko et l'emplacement pour un capteur de température de type DS18B20 utilisant une interface 1 wire (borne DS).

Si la qualité de fabrication de ces modules n'est généralement pas mauvaise celle de la conception et souvent de la documentation fournie l'est moins. Ils sont prévus pour fonctionner avec une batterie (rechargeable) de type LIR2032, en aucun cas une pile de type CR ou BR2032 ne doit être utilisée telle quelle, le mot battery en anglais ne faisant pas la différence entre les deux est aussi source de confusion. La conséquence de ce fonctionnement est une perte d'autonomie en cas d'absence d'alimentation, le pont diviseur protégeant Vbat en cas d'absence de l'accu à une consommation divisant l'autonomie permise par l'usage d'une pile seule par 4 ou 5. Si une pile CR2032 doit être utilisée ils sera nécessaire de procéder a une petite modification par suppression de l'ensemble résistance diode de charge et du pont diviseur comme sur ce petit schéma.

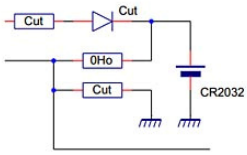
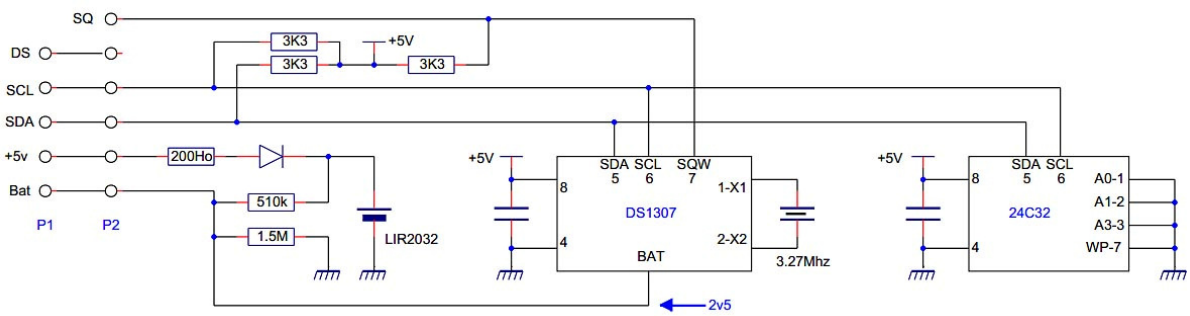


Schéma module



Adresse I²C = 0x68

Adresse I²C= 0x50

Fonctionnement Interface I²C

L'accès aux 64 octets des registres et de la mémoire ram du DS1307 est réalisé par l'intermédiaire de l'adresse unique 0x68 accédant a la valeur pointée par un compteur circulaire interne, celui-ci est automatiquement incrémenté a chaque opération d'écriture ou de lecture. La différence entre ces deux fonctions réside dans la première valeur envoyée qui correspond à celle de ce compteur en mode écriture. Le tableau suivant montre un exemple de ce mode de fonctionnement.

Op. Ecriture	Add PCF (0x68) write = 0xD0	Val cpt =0	Registre [0]	Registre [1]	Registre [2]	Registre [3]
Op. Lecture1	Add PCF (0x68) read = 0xD1	Registre [4]				
Op. Lecture2	Add PCF (0x68) read = 0xD1	Registre [5]	Registre [6]	….		

Cartographie mémoire

Les 64 octets de la mémoire ram du DS1307 sont repartis de la façon suite :

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Fonction	Plage
00	CH	Dizaines			Unités				Horl. secondes	00-59
01	0	Dizaines			Unités				Horl. minutes	00-59
02	0	0=24H	Dizaines		Unités				Horl. heures	00-23
		1=12H	PM / AM	Diz.						1-12
03	0	0	0	0	0	Unités			Horl. jours	1-7
04	0	0	Dizaines		Unités				Horl. date jour	1-31
05	0	0	0	Diz.	Unités				Horl. Date mois	1-12
06	Dizaines				Unités				Horl. Année	00-99
07	Out	0	0	SQW En.	0	0	RS1	RS2	Configuration	-
08	-	-	-	-	-	-	-	-	Ram utilisateur 58 x8	00-FF
3F	-	-	-	-	-	-	-	-		

Registres horloge

Les valeurs de l'horloge RTC sont codées en BCD, le nibble supérieur étant réservé aux dizaines, l'inferieur aux unités. Seuls les bits 7 du registre des secondes 0x00 et 6 du registre 0x02 possèdent une fonction de configuration.

- Bit 7-0 : Au niveau 1 l'oscillateur de l'horloge est stoppé, un niveau 0 représentant l'état normal horloge RTC en fonctionnement. A la mise sous tension initiale ce bit est mis a 1.
- Bit 6-0 : Au niveau 0 les heures sont fournies au format standard 0-24h, au niveau 1 au format anglais 12h AM/PM. Dans ce dernier cas le bit 5=1 indique des heures de journée PM.

Pour éviter les soucis de changement intempestifs des valeurs de date et heure dues aux variations de l'horloge RTC pendant le dialogue I²C il sera préférable d'arrêter l'horloge par la mise a un 1 du bit CH, puis de la redémarrer une fois les changements effectués.

Registre de configuration

Le registre 0x07 de configuration du circuit intégré offre les possibilités suivantes en fonction de l'état de certains bits. Le tableau suivant en donne la signification, les valeurs en gras sont celles utilisée par défaut a l'initialisation du circuit intégré.

bit 7 : OUT	Etat sortie SQW pin 7	**0**	SQW=1
		1	SQW=0
bit 4 : SQWE	Autorisation sortie horloge sur SQW	**0**	Etat SQW déterminé par le bit 7
		1	SQW oscille a la fréquence déterminé par RS
Bits 1 et 0 : RS	Fréquence horloge SQW	00	1 Hertz (F. Quartz / 2^15)
		01	4.096 khz (F. Quartz / 2^3)
		10	8.192 kHz (F. Quartz / 2^2)
		11	32.768 kHz (F. Quartz)

Accès direct par la bibliothèque Wire

La valeur du compteur d'accès aux registres étant permanant il sera nécessaire de s'assurer de sa valeur avant toute lecture du registre voulu, une opération d'écriture sera donc la plupart du temps nécessaire avant toute lecture.

Une lecture de l'ensemble des paramètres horaires de l'horloge RTC pourrait se présenter sous cette forme :

```
#include <Wire.h>
const uint8_t AddDS1307=0x68;                       // Adresse DS1307
uint8_t Heure[8];                                   // Tableau variables heure
uint8_t Idx=0;


void setup() {  // ------------------------------------------------ Initialisation
   Wire.begin();
   Serial.begin(9600);
}

void loop() {  // ------------------------------------------------- Boucle principale

  Wire.beginTransmission(AddDS1307);                //Init cpt registre add 0
  Wire.write(0);
  Wire.endTransmission();

  Idx=0;                                            // Lecture registres add0 a 6
  Wire.requestFrom(AddDS1307, 7);
  while (Wire.available()) {
    Heure[Idx]= Wire.read();
    Idx++;
  }

  if (Idx !=7) {                                    // Impression résultats
    Serial.println ("Erreur lecture DS1307");
  }
  else if (Heure[0] & 0x80) {
    Serial.println ("DS1307 a l'arret");
  }
  else {
    Idx=0;
    while (Idx < 7) {
      Serial.print (Heure[Idx],HEX);
      Serial.print (" - ");
      Idx++;
    }
    Serial.println ();
  }

  delay(1000);

}
```

Un accès universel à n'importe quelle adresse des registres ou de la mémoire utilisateur pourrait être effectué avec ces deux fonctions

```
#include <Wire.h>
const uint8_t AddDS1307=0x68;                       // Adresse DS1307

void setup() {  // ------------------------------------------------ Initialisation
   Wire.begin();
}
```

```
void loop() {  // ------------------------------------------------ Boucle principale
   ........
   WriteDS1307(0x20,0x00);                          // Raz octet mémoire adresse 0x20
   ........
   Toto=ReadDS1307(0x20);                           // Lecture ""            ""
   ........
}

void WriteDS1307 (uint8_t AddReg, uint8_t DataReg) {
  Wire.beginTransmission(AddDS1307); //Init cpt registre add 0
  Wire.write(AddReg);
  Wire.write(DataReg);
  Wire.endTransmission();
}

uint8_t  ReadDS1307 (uint8_t AddReg) {
  Wire.beginTransmission(AddDS1307); //Init cpt registre add 0
  Wire.write(AddReg);
  Wire.endTransmission();

  Wire.requestFrom(AddDS1307, 7);
  return Wire.read();
}
```

Librairie DS1307RTC

Les valeurs brutes en provenance de l'horloge étant au format RTC la bibliothèque `DS1307RTC` peut simplifier la tache de développement d'un applicatif. Cette bibliothèque utilise pour son fonctionnement les fonctions de conversion des formats de date et heure fournie par la bibliothèque `Time`, les deux devront donc être décompressées dans le répertoire libraries de l'IDE.

https://github.com/PaulStoffregen/Time
https://github.com/PaulStoffregen/DS1307RTC

L'adresse du circuit DS1307 étant fixe aucune instance de la bibliothèque ne doit être déclarée, seule une variable personnelle de type tmElements_t recevant la totalité des informations de date et heure devra l'être. Cette structure a le format suivant :

- `tmElements_t VarHorl;` : Déclaration Variable utilisateur VarHorl de type structure tmElements_t.
  - `VarHorl.second` : Secondes en format numérique sans bit CH.
  - `VarHorl.minute` : Minutes en format numérique.
  - `VarHorl.hour` : Heures en format numérique 24h.
  - `VarHorl.Wday` : Jour de la semaine
  - `VarHorl.Day` : Jour du mois au format numérique.
  - `VarHorl.Month` : Mois de l'année en format numérique.
  - `VarHorl.Year` : Année en format numérique 4 chiffres 20xx.

Les principales fonctions de cette bibliothèque sont :

- `Result=RTC.chipPresent()` : Retourne la valeur booléenne 1 si un dialogue avec le circuit DS1307 est possible, 0 dans le cas contraire.
- `result=RTC.read(Variable);` : Lit le circuit DS1307 et enregistre les valeurs horaires dans la variable au format `tmElements_t.` Retourne une valeur booléenne 1 si le résultat et correct, 0 si le dialogue est impossible ou l'horloge RTC a l'arrêt (bit CH).
- `Result=RTC.write(Variable);` : Enregistre dans le circuit DS1307 les valeurs contenues dans la variable de type `tmElements_t.` Retourne une valeur booléenne 1 si le résultat et correct.

- `RTC.setCalibration(uint8_t Val);` : Ecrit a l'adresse de configuration de la sortie SQW situé a l'adresse 0x07 les 5 bits bas de la valeur fournie (masque 0x1F effectué).

- `Val=RTC.getCalibration();` : Récupère la valeur des 5 bits inferieurs du registre de configuration.

Mémoires AT24Cxx

Si la mémoire EEprom embarquée dans les microcontrôleurs ATmega est généralement suffisante pour l'enregistrement des paramètres ou des options de fonctionnement du programme, les mémoire EEProm a accès sériel I&C de la gamme 24Cxx peuvent être une solution efficace pour le stockage de données, la constitution d'un historique ou de relevés de mesure.

Pour rappel les mémoires de type EEprom peuvent voir chaque octet effacé individuellement, avec une durabilité de 100 000 cycles pour les microcontrôleurs ATmega ou 1 million de cycles pour certaines mémoire 24Cxx, la mémoire programme flash des microcontrôleurs ne peut être effacée que par pages de 2560 octets et dispose d'une durée de vie beaucoup plus courte de 10 000 cycles environ (ces valeurs n'ayant qu'une indication statistique minimale).

Les mémoires 24Cxx ont vu leur capacité évoluer au fil du temps, les deux derniers chiffres de leur référence indique leur capacité. Selon les versions leurs tension d'alimentation peut etre comprise dans les plages 2.7/5.5v ou 4.5/5.5v voire 1.8/3.6v pour les modeles basse consommation.

De la manière la vitesse de transfert I²c maximale varie selon le type de la mémoire, est généralement au moins égale au mode fast 400kHz et le mode fast+ 1MHz souvent permis.

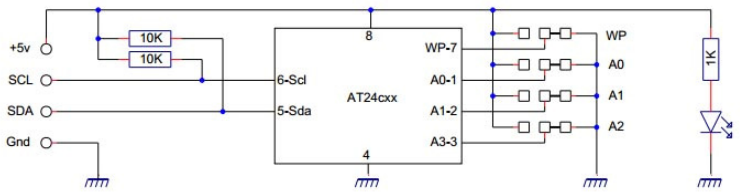
Il sera donc important de bien prendre en compte la marque et référence complète lors de l'utilisation d'une mémoire et en particulier les indices suivant la référence générique.

Toutes ces mémoires sont proposées dans un boiter 8 broches, outre l'alimentation et le bus I²C, l'entrée WP mise a l'etat 1 inhibe toute modification des données, et selon les modèles les entrées A0, A1, A2 permettent de modifier l'adresse racine I²C a utiliser pour y accéder.

Le tableau suivant résume les capacités mémoire obtenues et les modes d'adressage I²C a utiliser pour accéder a la mémoire.

	Capacité mémoire (octets)	Fonction A2-A1-A0	Adresse I²c	Nb bits adresse mémoire	Nb octets max mode page
ATmega 328 (Uno)	1 kio				
ATmega 2560 (Mega)	4 kio				
24c01	128	A2-A1-A0	1010 A2 A1 A0	7bits	8
24c02	256	A2-A1-A0	1010 A2 A1 A0	8bits	8
24c04	512	A2-A1-Nc	1010 A2 A1 P0	8bits x 2 blocs	16
24c08	1 kio	A2-Nc-Nc	1010 A2 P1 P0	8bits x 4 blocs	16
24c16	2 kio	Nc-Nc-Nc	1010 P2 P1 P0	8bits x 8 blocs	16
24c32	4 kio	A2-A1-A0	1010 A2 A1 A0	4 bits + 8 bits	32
24c64	8 kio	--	--	5 bits + 8 bits	32
24c128	16 kio	--	--	6 bits + 8 bits	64
24c256	32 kio	--	--	7 bits + 8 bits	64
24c512	64 kio	--	--	8 bits + 8 bits	128
24c1024	128 kio	Nc-A1-Nc	1010 0 A1 P0	8 bits + 8 bits x 2 blocs	256

Seuls les modèles supérieurs a la 24c16 offrent une différence sensible par rapport aux quantités de mémoire intégrés aux microcontrôleurs utilisés sur les cartes Arduino. D'ailleurs la plupart des modules disponibles sur eBay pour quelques euros et dont le schéma est donnée ci-dessous sont généralement équipés de 24c256.



Comme pour l'horloge DS1307 décrite précédemment l'accès a chaque octet de la mémoire EEprom est réalisé par l'intermédiaire d'un compteur / pointeur d'addresse interne au circuit integré.

Celui ci devra donc être positionné a la valeur d'adresse de l'octet dont l'on desire lire ou ecrire les valeurs. Ces opérations de lecture ou écriture peuvent être réalisées de manière aleatoire octet par octet, ou par pages de plusieurs octets le compteur d'adresse s'auto-incrementant dans cas (et uniquement dans ce cas).

Extrait datasheet 24C32 - 24C64

Figure 2. Byte Write

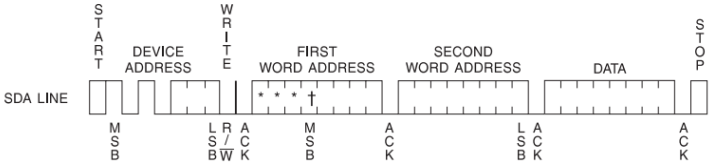
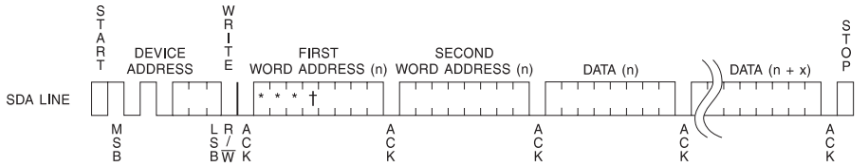


Figure 3. Page Write



Note: 1. * = DON'T CARE bits 2. † = DON'T CARE bits for the 32K

Les séquences de dialogue I²C ressembleront a ceci :

Op. Ecriture Add-a	Add EEprom / Write	Add-a	Data Add-a		
Op. Lecture Aléatoire Add-b	Add EEprom / Write	Add-b			
	Add EEprom / Read	Data Add-b			
	Add EEprom / Read	Data Add-b			
Op. Lecture par page Add-c	Add EEprom / Write	Add-c	Stop		
	Add EEprom / Read	Data Add-c	Data Add-c +1	Data Add-c +2	....
Op. Ecriture par page Add-d	Add EEprom / Write	Add-d	Data Add-d	Data Add-d +1	....

En fonction du modèle de l'EEprom la valeur fournie au pointeur d'adresse sera sur un (24c01 a 24c16) ou deux octets (24c32 a 24c1024) comme indiqué dans le tableau de la page précédente. Dans le cas ou cette valeur ne peut couvrir la plage d'adressage de la mémoire, celle-ci sera découpée en blocs et considérée comme plusieurs composants séparés accessibles par l'adresse I²C. En revanche les éventuels bits surnuméraires ne sont pas pris en compte par la mémoire EEprom et peuvent avoir un etat indifferent. Le tableau ci-dessous resume cette situation et le plan d'adressage pour plusieurs versions de ces circuits integrés

24c08 (1kio)		
Add I²c	Add	Mémoire
1010 A2 00	00 - FF	000 - 0FF
1010 A2 01	00 - FF	100 - 1FF
1010 A2 10	00 - FF	200 - 1FF
1010 A2 11	00 - FF	300 - 3FF

24c32 (4 kio)		
Add I²c	Add	Mémoire
1010 A2 A1 A0	x000-xFFF	000 - FFF

24c1024 (1Mio)		
Add I²c	Add	Mémoire
10100 A1 0	0000 - FFFF	00000-0FFFF
10100 A1 1	0000 - FFFF	10000-1FFFF

En mode page l'incrémentation du pointeur d'adresse n'est assurée que dans la limite du buffer de page de la mémoire EEprom, passé cette valeur le compteur est rebouclé sur lui-même et les valeurs réécrites ou relues. Par exemple la lecture de 35 octets consécutifs d'une mémoire 24C16 à partir d'une valeur de pointeur d'adresse initial Add donnera les résultats suivant :

	1	2	3	...	31	32	33	34	35
Add EEprom / Read	Data Add	Data Add+1	Data Add+2	...	Data Add+30	Data Add+31	Add	Data Add+1	Data Add+2

La taille des buffers utilisée par défaut par la librairie Wire Arduino etant de 32 octets il ne sera pas possible de provoquer des rebouclages de l'incrémentation automatique du pointeur en mode page avec la majorité des ces mémoires, cette situation pourra néanmoins se rencontrer avec les plus anciens modeles, se reporter au tableau recapitulatif des caracteristiques d'adressage precedent.

Temps de latence d'écriture

En simplifiant fortement les EEprom comme les mémoires flash ne peuvent être modifiées en écriture que sur des cellules mémoire vierges, l'effacement de celles-ci ne peut se faire que par page par injection d'une haute tension.

Lors d'une requête I²C d'écriture en mémoire les données sont stockées dans un buffer temporaire, la logique interne de l'EEprom active le générateur haute tension et son cycle d'écriture, pendant cette durée *tWr* le bus I²C est désactivé rendant tout autre envoi ou lecture de données impossible.

Les constructeurs indiquent dans la documentation de leurs composants la valeur maximale de ce temps de cycle d'écriture *tWr*, celle-ci est généralement pour les mémoires de type 24Cxx de 10ms ou 5ms pour certains modèles rapides. Cette indication est toutefois une valeur maximale garantie, le temps de cycle reel peut être inferieur a 2ms, l'utilisation d'un test d'acces au bus I²C peut alors optimiser les temps de fonctionnement du programme par rapport a une temporisation utilisant la valeur de *tWr* par défaut.

Ce temps d'écriture important à l'échelle informatique rendra l'utilisation des mémoires EEprom série déconseillée pour certaines applications, l'écriture octet par octet d'une 24C256 pourra prendre plusieurs secondes. L'écriture des programmes devra lui aussi privilégier le buffer interne et l'envoi de plusieurs octets simultanés, l'écriture d'un bloc de 32 octets prendra théoriquement 32 fois moins de temps que ces octets envoyés un a un.

A titre d'exemple, la première version de la fonction de formatage qui sera vu dans le chapitre suivant ne disposait pas de contrôle de disponibilité de l'EEprom, les deux envois suivant celui d'adresse 0x0030 ne sont pas pris en compte par l'EEProm et la plage mémoire comprise entre 0x0040 et 0x005F n'est ni écrite ni effacée.

Write protect

La mise a l'etat 1 de l'entrée WP des EEproms 24Cxx inhibe la logique interne d'ecriture, en revanche les données recues sont bien stockées dans la mémoire tampon et aucunne signalisation sur le bus I²C ne signale cet etat. Dans le cas ou cette entrée serait cablée et utilisée par un autre process ou un simple cavalier il pourra etre necessaire de proceder a une relecture des octets ecrits pour confirmer que cette option n'est pas activée.

Utilisation Arduino

L'accès a ce type de mémoire ne pose pas de problème particulier, les algorithmes ou les blocs fonctionnels devront néanmoins faire l'objet de traitements différentiés entre les mémoires utilisant un ou deux octets pour le pointeur d'adresse et celles découpées en blocs utilisant l'adresse I²C. Les exemples qui suivent ne seront donc valables que pour les références d'EEprom allant de la 24c32 à la 24c512 utilisant une adresse 16bits.

L'adresse I²C à utiliser pour accéder au mémoire 24Cxx dépend de l'état de leurs entrées A0 à A2, le tableau suivant indique les valeurs hexadécimales correspondantes.

A2 - A1 - A0	000	001	010	011	100	101	110	111
Add hex	50	51	52	53	54	55	56	57

Si des bibliothèques toute faites permettant la gestion de ce type de mémoire, celles-ci sont parfois adaptées qu'a une seule taille de mémoire. La faible difficulté de conception de routines d'écriture ou de lecture fait qu'il sera (amha) préférable de passer par ses propres process adaptés a ses besoins.

Exemple d'utilisation basique 24c32

Les essais ayant été realisés a partir du module RTC vu precedemment, le programme est prevu pour utiliser cette mémoire de 4ko mais sa conception est telle que son fonctionnement est possible jusqu'à la 24c512 ou la 24c1024 en utilisant deux adresses I²C.

Le but ayant été de verifier le fonctionnement du bus a l'aide d'un analyseur logique, 4 routines sont activées par la mise a la masse d'une des entrées tout ou rien dig7 a dig5. La sortie des informations est realisée sur la console serie.

Deux constantes caracterisant le modele d'EEprom sont utilisées, EE_I²C definissant l'adresse du peripherique sur 7 bits, et EE_Type egal a sa taille en kio. L'analyse rapide des fonctions principales est la suivante :

Loop : Gere les entrées sorties de lancement de test avec

- Dig7 : Ecriture de valeurs fixes a quelques adresses mémoire, l'utilisation de l'emplacement 0x1000 est intentionnelle pour verifier le rebouclage des palges d'adresse en cas de deplacement, la valeur 0xEE etant bien ecrite a l'adresse reelle 0x0000.
- Dig6 : Affichage sur la console seriede la totalité du contenu de la mémoire au format hexadecimal par lignes de 16 octets, quelques instruction supplementaires et la fonction PrintHex8() sont utilisées pour formater l'affichage en colonnes de largeur identiques.
- Dig5 : Lancement du formatage de l'EEprom, la durée d'execution de celui-ci est d'environ 1 seconde.

EEWrite(adresse, data) : Ecriture d'un octet .

L'adresse au format 16bits est convertie en deux octets transmis a la mémoire avant l'octet de donné. Aucun contrôle n'est effectué, seule une relecture serait garante du bon deroulement de l'operation, une reponse zero a la fonction Wire.endTransmission n'etant pas suffisante. La temporisation de 5ms est la pour eviter toute nouvelle ecriture avant ecoulement de tWr, cette instruction n'est pas forcement obligatoire si le delai entre chaque appel de EEWrite est geré dans la boucle principale du programme.

Data=EERead(adresse) : Lecture d'un octet.

Comme pour EEWrite l'adresse est fournie sous 16bits et transmise sous forme de deux octets a la mémoire, une lecture est ensuite effectuée sur le bus I²C.

EEClear(Taille EEprom en kio) : Effacement mémoire complet.

Pour limiter le nombre de cycles d'ecriture et la multiplication des delais tWr l'effacement est realisé par l'envoi de 16 octets a 0xFF, il serait tout a fait possible d'utiliser une salve de 32 octets ou plus avec certaines memoires, dans ce cas une boucle for/next serait plus adaptée que l'utilisation d'un tableau. Les adresses sont envoyées toujours sur deux bits a partir du compteur CptAdd mais avec une multiplication par 16. Pour optimiser le temps de traitement, l'utilisation d'une temporisation tWr fixe est remplacée par un contrôle de la transmission I²C, le compteur d'adresse n'est incrementé que si cette ecriture a reussie, dans le cas contraire une petite temporisation est effectuée avant un nouvel essai, la valeur de 2ms est un compromis avec le temps de reponse de la mémoire pour limiter le taux d'occupation du bus.

```
#include <Wire.h>
const uint8_t ETest1=7;
const uint8_t ETest2=6;
const uint8_t ETest3=5;

const uint8_t EE_I2C=0x50;                          //Add EEprom
const uint8_t EE_Type=4;                            //24C32 = 4k

uint16_t cptWr=0;

void setup() { // ------------------------------------- Initialisation ports et E/S
  Wire.begin();
  Serial.begin(115200);
  Serial.println (" ________ Start");
  pinMode(ETest1, INPUT);
  digitalWrite(ETest1, HIGH);
  pinMode(ETest2, INPUT);
  digitalWrite(ETest2, HIGH);
  pinMode(ETest3, INPUT);
  digitalWrite(ETest3, HIGH);
}


void loop() { // ------------------------------ Boucle principale, test des entrées

  if (digitalRead(ETest1)==LOW) { uint16_t cpt=0;
    EEWrite(0x10,0x10);
    EEWrite(0x11,0x11);
    EEWrite(0x200,0x33);
    EEWrite(0xFFF,0x44);
    EEWrite(0x1000,0xEE);
    delay(500);
  }

  if (digitalRead(ETest2)==LOW) { // ------------------------------------------Dump
    uint16_t CptAdd=0;
    Serial.print ("Dump EEprom ");
    Serial.print (EE_Type);
    Serial.println ("kio");
    do {
      Serial.print ("  -  ");                        //Affichage adresse xxxx
      if(CptAdd < 0x100) {Serial.print ("0");}
      if(CptAdd < 0x10) {Serial.print ("0");}
      Serial.print(CptAdd,HEX);
      Serial.print("0 = ");

      Wire.beginTransmission(EE_I2C);               //Envoi adresse a l'EEprom
      Wire.write((uint8_t)(CptAdd >> 4));
      Wire.write((uint8_t)((CptAdd <<4) & 0xFF));
      Wire.endTransmission();

      if (Wire.requestFrom(EE_I2C, 16) !=16) {Serial.print ("Fail");}
      else {
        while (Wire.available()) {                  //Affichage 16 octets lus xx
        PrintHex8(Wire.read());
        Serial.print(" - ");
        }, la durée
      }
```

```
      CptAdd++;
      Serial.println();
    } while (CptAdd >> 6 < EE_Type );

    delay(1000);
  }


  if (digitalRead(ETest3)==LOW) { // ----------------------------------------Format
    Serial.println("Effacement EEprom !!!!!!!!!!!");
    uint32_t Temps=millis();                        //Memo heure de départ
    EEClear(EE_Type);
    Serial.print("Ok : ");                          //Affichage durée format
    Serial.print (millis() - Temps);
    Serial.println ("ms");
  }
}

//================================================================================


void EEWrite(uint16_t EE_Add, uint8_t EE_Data) { // -------------------Ecriture octet
  Wire.beginTransmission(EE_I2C);
    Wire.write((uint8_t)(EE_Add >> 8));             //Envoi Adresse XX..
    Wire.write((uint8_t)(EE_Add & 0xFF));           //Envoi adresse ..XX
    Wire.write(EE_Data);                            //Envoi data
  Wire.endTransmission();
  delay(5);                                         //Tempo tWr EEprom
}

uint8_t EERead(uint16_t EE_Add) { // ----------------------------Lecture octet EEprom
  uint8_t Tmp=0xFF;
  Wire.beginTransmission(EE_I2C);
    Wire.write((uint8_t)(EE_Add >> 8));
    Wire.write((uint8_t)(EE_Add & 0xFF));
  Wire.endTransmission();

  Wire.requestFrom(EE_I2C, 1);
  if (Wire.available()) { Tmp = Wire.read(); }      //Lecture data
  return  Tmp;
}

void EEClear(uint8_t Taille) { // -------------------------Effacement complet EEprom

  uint16_t CptAdd=0;
  uint8_t
BuffRaz[16]={0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF
,0xFF};

  do {
  Wire.beginTransmission(EE_I2C);
    Wire.write((uint8_t)(CptAdd >> 4));             //Envoi Adresse XX..
    Wire.write((uint8_t)((CptAdd <<4) & 0xFF));     //Envoi Adresse ..X0
    Wire.write(BuffRaz,16);                         //Envoi 16 octets = 0xFF
  if (Wire.endTransmission() == 0) {CptAdd++;}      //Si ok Cpt+1(adresse=+16)
  else { delay(2);}                                 //Si échec attente
  } while (CptAdd >> 6 < Taille );
}


void PrintHex8(uint8_t Val) {  //Impression série octet sous forme hexa 2 caractères
  if (Val < 0x10) {Serial.print("0");}
  Serial.print (Val,HEX);
}
```

Liens et révisions document

Liens

Révision

v1.00	30/08/2016	Première diffusion.

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