Le BUS I2C

A / INTRODUCTION

Le bus I2C (**I**nter **I**ntegrated **C**ircuit Bus) est le bus historique, devenu standard, développé par Philips dans les années 80, pour permettre de relier facilement à un microprocesseur divers circuits intégrés (spécialisés dans le stockage et l'affichage de données, dans l'exécution de fonction numériques ou analogiques diverses), en particulier dans un téléviseur.

Il existe d'innombrables périphériques exploitant ce bus, dans les appareils TV et vidéo (récepteur télécommande, réglages ampli BF, tuner, horloge …), mais aussi dans les systèmes audio et radio, postes téléphoniques, systèmes électroniques automobiles, PC, appareils électroménagers, etc.

B / CARACTERISTIQUES GENERALES

Le bus I2C permet de faire communiquer, par une liaison série synchrone, entre eux des composants électroniques très divers grâce à seulement trois fils :

- Un signal de donnée (SDA).
- Un signal d'horloge (SCL).
- Un signal de référence électrique (Masse).

D'autre part, ce bus est multi-maître (plusieurs circuits peuvent prendre le contrôle du bus) et sa longueur peut atteindre 3 à 4 mètres à condition que la charge capacitive n'excède pas 400 pF.

Dans sa version de base, les données sont transmises en série à 100 kbits/s en mode standard. Ce qui ouvre la porte de cette technologie à toutes les applications où la vitesse n'est pas primordiale.

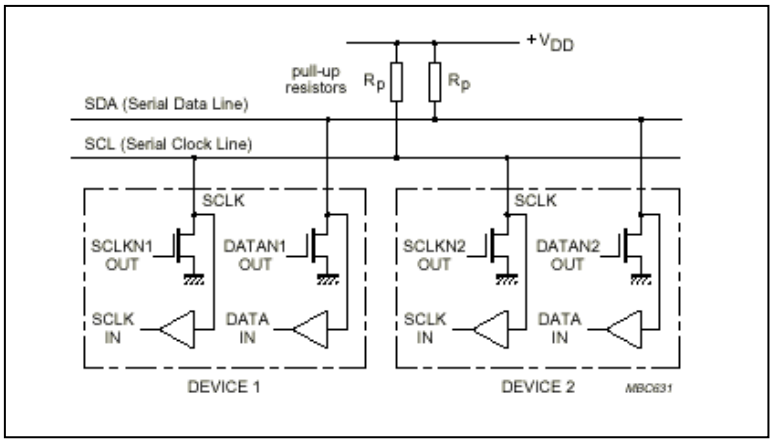
La révision de la norme en 1992 autorise de nouveaux modes de fonctionnement :

- vitesse de transfert portée de 100 à 400 kbits/seconde, voire 3,4Mbits/s en mode HV,
- adressage des circuits étendu de 7 à 10 bits.

De nombreux fabricants ayant adopté le système, la variété des circuits disponibles disposant d'un port I2C est énorme : Ports d'E/S bidirectionnels, Convertisseurs A/N et N/A, mémoires (RAM, EPROM, EEPROM, etc.), Circuits Audio (Egaliseur, Contrôle de volume, ...) et autre drivers (LED, LCD, ...).

C / FONCTIONNEMENT DU BUS I2C

1 / Caractéristiques électriques



Afin de d'éviter les conflits électriques **les Entrées/Sorties, SDA et SCL sont de type "Collecteur Ouvert" (ou "Drain Ouvert").**

Les sorties étant à collecteur (ou drain) ouvert la tension de sortie à l'état haut est la tension « ramenée » par les résistances de rappel (qui sont connectées à la ligne d'alimentation des circuits « VDD »).

Cela permet ainsi la présence de plusieurs maîtres sur le bus.

Cela permet aussi la communication entre dispositifs réalisés dans des technologies différentes et utilisant éventuellement des tensions d'alimentation différentes.

Pour les dispositifs fonctionnant sous une tension de 5V ±10%, les niveaux d'entrée sont : V_{ILmax} = 1,5V et V_{IHmin} = 3V. Dans tous les cas la tension de sortie à l'état bas est V_{OLmax} = 0,4V

2 / Terminologie

Emetteur : Unité qui envoie les données sur le bus.
Récepteur : Unité qui reçoit les données du bus.
Maître : Unité qui démarre un transfert, génère des signaux d'horloge et met fin au transfert.
Esclave : Unité adressée par le maître.
Multi-maître : Possibilité pour plusieurs maîtres de tenter de prendre le contrôle du bus en même temps, sans altérer le message.
Arbitrage : Procédure permettant de résoudre les conflits d'accès des maîtres au bus et d'éviter l'altération du message. Le contrôle du bus n'est accordé qu'à un maître à la fois.
SDA : Ligne des signaux de données.
SCL : Ligne des signaux d'horloge.

3 / Protocole de transmission .

Le protocole de communication doit prendre en compte les règles suivantes :

- une distinction entre les circuits maîtres (ceux qui décident du dialogue) et les circuits esclaves,
- une identification des circuits,
- un acquittement des transferts (confirmation par les circuits de la bonne réception des informations qui leur ont été transmises),
- un système de priorité en cas de conflit.

Dans les applications courantes les modes les plus utilisés sont les suivants :

a / Au repos (en l'absence de transmission)

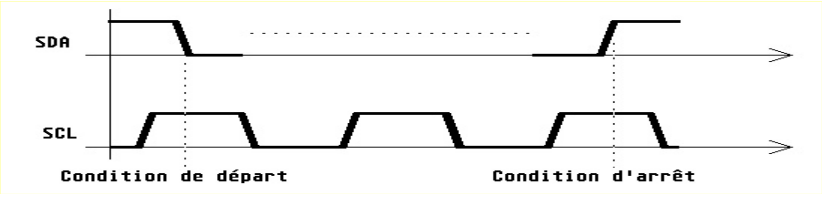
Les lignes SDA et SCL sont au niveau haut

b / Le maître veut lire une information contenue dans un circuit esclave .

Le déroulement de la séquence va être :

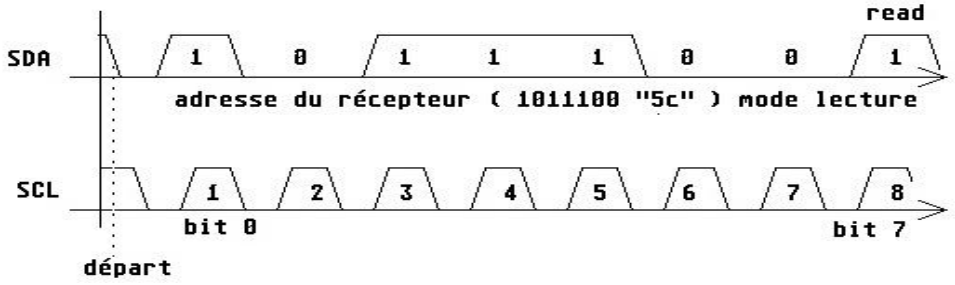
«1 » Condition de départ .

Un front descendant est appliqué sur la ligne SDA quand SCL est encore au niveau haut :



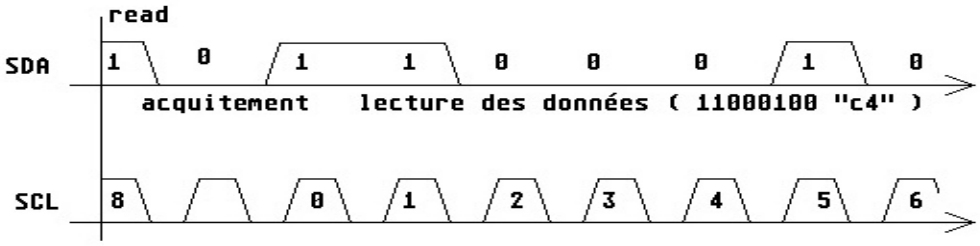
« 2 » Transmission de l'adresse (premier octet) .

A chaque niveau haut de SCL correspond un bit de donnée sur SDA (en commençant par les bits de poids fort). Ce premier octet contient l'adresse du circuit désiré (bit 7 … 1 ce qui autorise 128 possibilités) puis l'information de sens de transfert (bit 0, de niveau un en lecture et de niveau 0 en écriture).



« 3 » Acquittement .

Après chaque octet reçu, le récepteur doit générer un signal d'acquittement. Pour cela, durant le niveau haut de SCL qui suit la transmission de l'octet, le récepteur (l'esclave dans ce cas) met la ligne SDA au niveau bas.



« 4 » Transmission du deuxième octet (lecture) .

Durant chaque niveau haut sur SCL, l'esclave (ici émetteur) envoie un des 8 bits de la donnée (en commençant toujours par le bit de poids fort).
Après une bonne réception, c'est le circuit maître qui doit générer le signal d'acquittement en maintenant la ligne SDA au niveau bas durant le niveau haut de SCL.
Dans le cas où le récepteur n'est pas en mesure de recevoir les données, il doit mettre la ligne SCL au niveau bas pour mettre l'émetteur en attente.

« 5 » Condition d'arrêt .

Un front montant est appliqué sur la ligne SDA quand SCL est au niveau haut.

c / Le maître veut écrire une information dans un circuit esclave.

Les étapes **1**, **2**, **3** et **5** sont identiques au cas précédent. Au niveau de l'étape 4 c'est le maître qui va émettre de la même façon qu'il transmet les adresses.

d/ Le mode multi-maître .

L'horloge (SCL) résultant de ce mode, est le produit des différentes horloges générées par l'ensemble des maîtres. C'est ainsi qu'est générée l'horloge SCL synchronisée, dont la durée du niveau bas est déterminée par le dispositif ayant la période d'horloge au niveau bas la plus longue, et le niveau haut par le dispositif ayant la période d'horloge au niveau haut la plus courte.
En cas de conflit le maître prioritaire est celui qui envoie un niveau bas sur la ligne SDA.
Le maître qui tente d'envoyer un niveau haut sur SDA, et qui cependant ne peut y lire qu'un niveau bas, arrête son transfert et passe en mode récepteur esclave (au cas où ce soit à lui que l'on veuille s'adresser).

D / CONCLUSION

De par sa conception, ce système permet donc un dialogue extrêmement souple (mode multi-maître) et des réalisations très modulaires (seulement deux fils à connecter). Ce type de connexion présente aussi un avantage économique (réduction du prix des boîtiers et de la connectique associée). De plus, au niveau des études, il est facile de réutiliser des modules ou des sous-programmes déjà fabriqués.

En revanche, les priorités sont assez difficiles à gérer. Ainsi l'arrêt des maîtres gênants est seulement possible dans le cas où ils ont une adresse plus basse que celle du maître prioritaire. Parfois, cette difficulté entraîne l'utilisation d'une ligne d'interruption externe nécessitant un troisième fil, ainsi qu'une gestion matérielle et logicielle appropriée.

E / FONCTIONS ARDUINO REALTIVES AU BUS I2C

Références Arduino, la bibliothèque « wire » (fonction I2C)

	Description	Syntaxe	Paramètres	Valeur renvoyée
begin()	Configure la carte Arduino pour utiliser les fonctions I2C. **Cette fonction ne doit être appelée qu'une seule fois.**	Wire.begin(**address**)	**address**: l'adresse I2C de la carte, lorsqu'elle est utilisée en esclave (7bits). Rien, pour utiliser la carte en maître.	Aucune
beginTransmission()	Initie un transfert en mode écriture : Génère une condition de départ et envoie le premier octet (adresse de l'esclave sur 7 bits) et (bit r/w à 0)	Wire.beginTransmition(**address**)	**address**: l'adresse I2C de la carte esclave (7bits).	Aucune
endTransmission()	Génère une condition d'arrêt. (met fin à la transmission). Ou génère un 'restart' (condition d'arrêt + condition de départ)	**status**=Wire.endTransmission(**b**)	**b** : boolean True (ou rien) → stop False → restart *(stop = condition d'arrêt)*	**status** : byte (lecture optionnelle) 0→ pas d'erreur 1→ data too long 2→NACK adresse 3→NACK données 4→ autre erreur
write()	Ecriture d'un ou plusieurs octets du maître vers l'esclave	**nb**=Wire.write(**value**) **nb**=Wire.write(**string**) **nb**=Wire.write(**data,length**)	**value** → l'octet à écrire **string** → une chaine de caractère à envoyer **data** → un tableau de donnée **length** → le nombre d'octets à transmettre	**nb** : byte (lecture optionnelle) Nombre d'octets écrits
requestFrom()	Initie un transfert en mode lecture : Génère une condition de départ et envoie le premier octet (adresse de l'esclave sur 7 bits) et (bit r/w à 1) Puis met fin au transfert par l'envoi d'une condition d'arrêt ou restart	Wire.requestFrom(**address, quantity, stop**)	**address**: l'adresse I2C de la carte esclave (7bits). **quantity :** nombre d'octets à recevoir **stop :** true (ou rien) → stop false → restart	Aucune.
read()	Lecture d'un octet (cette fonction est utilisée après la fonction requestFrom())	**Val**=Wire.read()	Aucun	**Val** : byte Octet reçu

available()	Cette fonction renvoie le nombre d'octets qu'il reste à lire
	
	Exemple d'utilisation :
	
	Wire.requestFrom(Ox58, 6);
	while(Wire.available())
	{
	char c = Wire.read
	Serial.print(c);
	}
		nbo= Wire.available()	Aucun	**nbo** : byte
				
				nombre d'octets non lus

onReceive()	Cette fonction est utilisée lorsque la carte Arduino est en mode esclave. Elle permet de définir la fonction à exécuter lorsque la carte **reçoit** des données.	Wire.onReceive(handler)	**Handler :** la fonction à appeler (voir doc)	Aucune
onRequest()	Cette fonction est utilisée lorsque la carte Arduino est en mode esclave. Elle permet de définir la fonction à exécuter lorsque un maitre **demande** des données.	Wire.onRequest(handler)	**Handler :** la fonction à appeler (voir doc)	Aucune

Références Arduino, quelques fonctions utiles…

	Description	Syntaxe	Paramètres	Valeur renvoyée
pinMode()	Configure la broche spécifiée pour qu'elle se comporte soit en entrée, soit en sortie.	pinMode(broche, mode)	**broche**: le numéro de la broche de la carte Arduino **mode**: soit INPUT (entrée) ou OUTPUT (sortie)	Aucune
digitalWrite()	Si la broche a été configurée en SORTIE : Met un niveau logique HIGH (HAUT) ou LOW (BAS) sur une broche numérique. Si la broche est configurée en ENTREE : Active (1) ou désactive (0) la résistance interne de "rappel au plus" (pullup).	digitalWrite(broche, valeur)	**broche**: le numéro de la broche de la carte Arduino **valeur** : HIGH ou LOW (ou bien 1 ou 0)	Aucune
digitalRead()	Lit l''état (= le niveau logique) d'une broche précise en entrée numérique, et renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais).	Val = digitalRead(broche)	**broche** : le numéro de la broche numérique que vous voulez lire. (int)	Renvoie la valeur HIGH (1) ou LOW (0)
analogRead()	Lit la valeur de la tension présente sur la broche spécifiée. La carte Arduino comporte un convertisseur analogique-numérique 10 bits.	Val = analogRead(br_an)	**br_an** : le numéro de la broche ana-logique sur laquelle il faut convertir la tension.	Val : int (0 to 1023) correspondant au résultat de la mesure effectuée
Analog Reference()	Configure la tension de référence utilisée avec les entrées analogiques, 3 options : DEFAULT: tension de référence de 5 V INTERNAL: référence interne de 1.1 V EXTERNAL: tension sur la broche AREF	analogReference(type)	**type** : le type de référence utilisée soit → DEFAULT, INTERNAL, ou EXTERNAL.	Aucune.
delay(ms)	Réalise une pause dans l'exécution du programme pour la durée (en millisecondes) indiquée en paramètre.	delay (ms)	**ms** (unsigned long): le nombre de millisecondes que dure la pause .	Aucune
Serial.begin()	Fixe le débit de communication en bits par secondes (l'unité est le baud) pour la communication série.	Serial.begin(9600);	int **debit**: debit de communication en bits par seconde (ou baud)	Aucune
Serial.print()	Affiche les données sur le port série. Les valeurs de type int, float, byte sont affichés sous la forme de caractères ASCII. Les caractères et les chaînes sont affichés tels que.	SERIAL.PRINT(VAL) Serial.print(val, format)	**val**: la valeur à afficher. N'importe quel type de données. **format** : spécifie la base utilisée (pour les nombres entiers) ou le nombre de décimales (pour les nombres de type float)	Aucune
Serial.println()	Idem Serial.print() , puis rajoute un saut de ligne .	SERIAL.PRINTLN(VAL) Serial.println(val, format)	Idem Serial.print()	Aucune
LiquidCrystal()	Crée un objet de type LiquidCrystal. Permet d'utiliser un afficheur LCD. *Ne pas oublier de rajouter la ligne :* *#include <LiquidCrystal.h>*	LiquidCrystal lcd(rs, rw, en, d4, d5, d6, d7)	Les paramètres: **rs, rw, en, d4 à d7 sont à remplacer par le** numéro de la broche (de l'Arduino) connectée.	Aucune
Lcd.begin()	Initialise l'afficheur lcd	lcd.begin(col, li)	**col**: le nombre de colonnes de l'af-ficheur **li**: le nombre de lignes de l'afficheur	Aucune
lcd.clear()	Efface l'écran et positionne le curseur dans le coin supérieur gauche de l'écran.	lcd.clear()	Aucun	Aucune
lcd.home()	Positionne le curseur dans le coin gauche de l'écran LCD. Le texte sera dorénavant écrit à partir de cette position.	lcd.home()	Aucun	Aucune
lcd.setCursor()	Positionne le curseur de l'afficheur LCD à la localisation voulue (colonne,ligne), et donc définit la position à laquelle le texte sera dorénavant affiché à l'écran.	lcd.setCursor(col, li)	**col**: colonne à laquelle positionner le curseur (0 est le 1ère colonne) **li**: ligne à laquelle positionner le curseur (0 est le 1ère ligne, 1 la 2ème, etc..).	Aucune
lcd.print()	Affiche texte et nombre sur l'afficheur LCD	lcd.print("texte") lcd.print(data) lcd.print(data, BASE)	Idem Serial.print()	Aucune
lcd.println()	Idem lcd.print() , puis rajoute un saut de ligne .	lcd.println("texte") lcd.println(data) lcd.println(data, BASE)	Idem Serial.print()	Aucune