QUAND LE MAITRE ET SES ESCLAVES PRENNENT LE BUS…

OU INITIATION AU PROTOCOLE I²C

Chère lectrice, cher lecteur,

On est entre nous, alors on va se tutoyer.

L'objet de ce document est de te faire comprendre comment des composants électroniques arrivent à communiquer entre eux grâce à la norme I²C (*inter components communication)* de Philips Composant.

A savoir : Tu n'es pas la seule ou le seul à (vouloir) utiliser le bus I²C. D'autres y ont pensé avant toi. C'est ainsi que si tu démontes ton répondeur, ton téléphone, ton magnétoscope, ta télévision, ta récente voiture ou ta chaîne HIFI, tu as toutes les chances d'y trouver des composants I²C.

Pour commencer, nous allons voir ensemble, sans entrer dans les détails, l'origine du bus I²C. Ensuite, nous pourrons découvrir les principes du protocole. Enfin, nous pourrons nous attaquer gentiment à la compréhension et la construction de messages I²C selon la norme.

1) Où est le problème ?

Dans un projet électronique, on doit connecter différents composants entre eux. Parfois, les composants sont éloignés les uns des autres. Pour des raisons d'économie et de sécurité, on cherche à réduire le nombre de connexions entre les composants, tout en permettant de transmettre n'importe quelle information en n'importe quelle quantité.

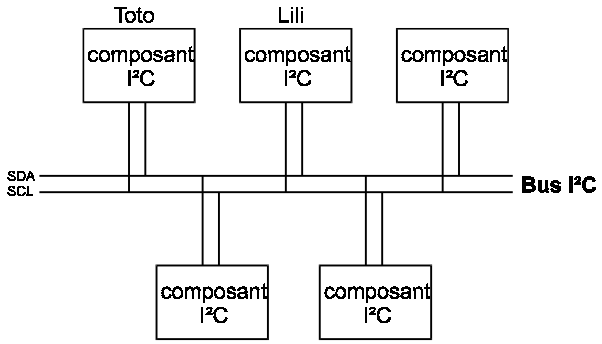
Lors de l'invention du télégraphe, M. Morse avait trouvé une solution en proposant l'alphabet qui porte son nom. Le morse, c'est pour transmettre des

lettres et des chiffres. C'est bien adapté à des messages entre humains. Mais les composants, on ne sait jamais ce qu'ils vont devoir se transmettre.

Lorsque le nombre de fils est réduit, la solution est de découper les informations afin de les transmettre par petits morceaux, les uns après les autres. On parle dans ce cas de communication en série. Bien entendu, les messages seront communiqués plus lentement. On ne peut pas tout avoir. Les petits morceaux les plus élémentaires sont des bits. Chaque bit est soit à l'état haut, soit à l'état bas.

Le bus I²C transmet des bits les uns après les autres.

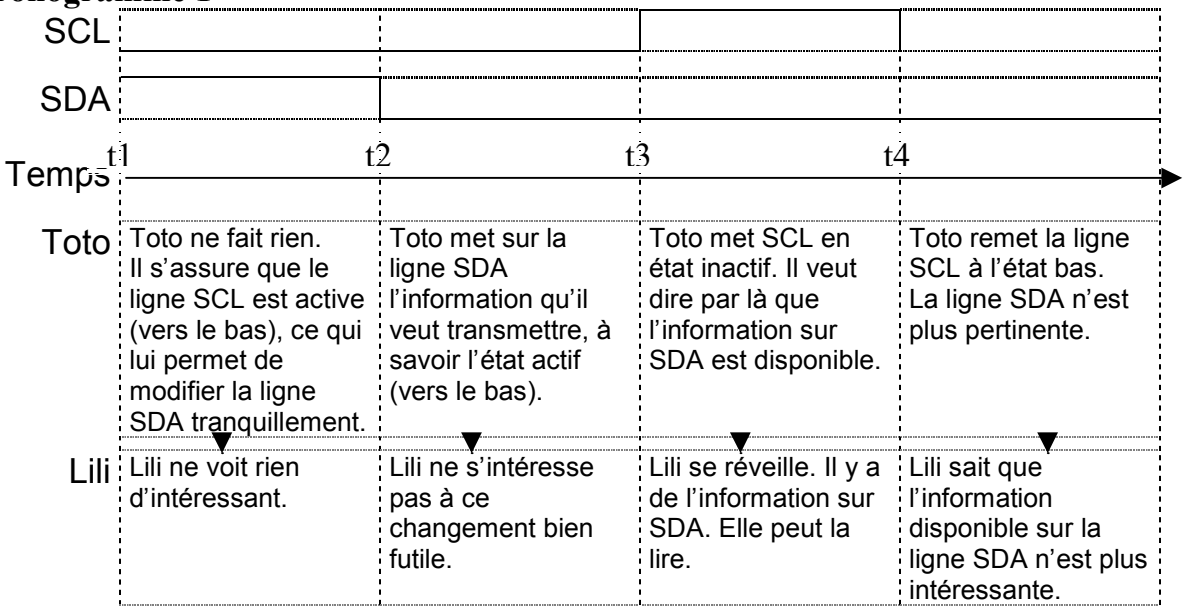
La base du bus I²C, c'est l'existence de deux lignes de communication. Ce sont deux informations qui circulent en même temps, on dit en parallèle. La première, nommée SDA (*signal data*), indique la valeur du bit (état actif ou état inactif). La seconde, nommée SCL (*signal clock*), joue le rôle d'horloge.



Le composant Toto veut transmettre un bit actif au composant Lili.

<u>Remarque</u> : Une ligne (SCL ou SDA) est active lorsqu'elle est dessinée vers le bas.

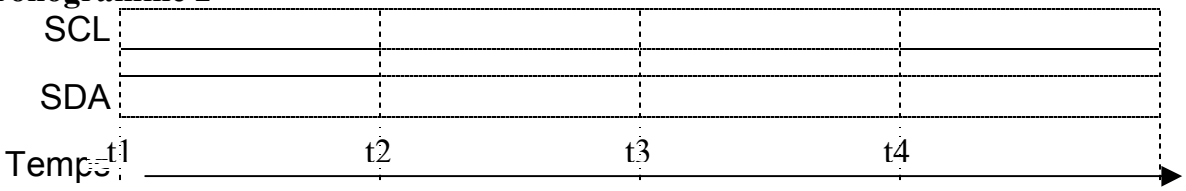
Chronogramme 1



Toto veut transmettre à Lili un bit inactif (et non plus actif comme dans le premier exemple).

Sur le même modèle que ci-dessus, je te propose de compléter le chronogramme 2 (car c'est ainsi qu'on le nomme) de la nouvelle communication.

Chronogramme 2



<u>Pour vérifier :</u>

En t1, la ligne SCL est à l'état actif (bas) et SDA est inactive.

En t2, Toto doit mettre SDA à l'état correspondant à l'information qu'il veut transmettre. Il doit donc être inactif (à l'état haut).

En t3, Toto indique qu'une information est disponible sur la ligne SDA en rendant inactive la ligne SCL. SCL doit être à l'état haut à partir de t3.

En t4, Toto espère que Lili a reçu le message et rend active la ligne SCL (état bas).

En résumé, la ligne SDA reste tout le temps à l'état haut (inactif). La ligne SCL est tout le temps à l'état bas (actif), sauf entre t3 et t4 où elle est à l'état haut (inactif).

Récapitulatif

Le bus I²C permet de connecter des composants électroniques entre eux.

Il comporte 2 lignes. Sur la première (SDA) transitent les données, sur la seconde (SCL) transite l'horloge indiquant la validité des données.

2) Communication en série : je vous en mets combien ?

Tout ça, c'est bien beau, mais pas très intéressant. Toto n'a, jusque là, transmis qu'une information bien élémentaire, un bit dont l'état est actif ou inactif.

C'est là que nous trouvons les combinaisons de 8 bits en octet.

A partir de maintenant, Toto transmettra à Lili un octet et non un bit.

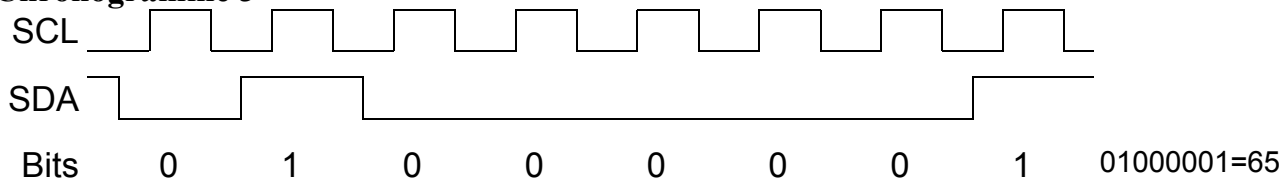
Par exemple, Toto veut transmettre l'octet suivant : 0100 0001, ce qui correspond à la valeur décimale 65 (pour la correspondance entre binaire et décimal, voir en Annexe 1 : conversion entre binaire et décimal).

Remarque : un bit de valeur 1 sera transmis par l'état inactif et 0 par l'état actif.

Valeur	*état de la ligne*	*représentation sur chronogramme*
1	*inactif*	*état haut*
0	*actif*	*état bas*

Attention : ceci est spécifique au protocole I²C. On parle, dans ce cas, de logique inverse.

Chronogramme 3

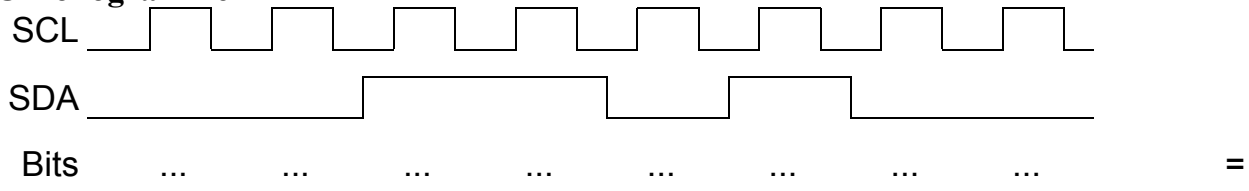


<u>Remarque</u> : SDA est préparé avant que SCL soit à l'état haut. Ainsi s'explique le décalage entre le changement d'état de SCL et celle de SDA pendant la transmission.

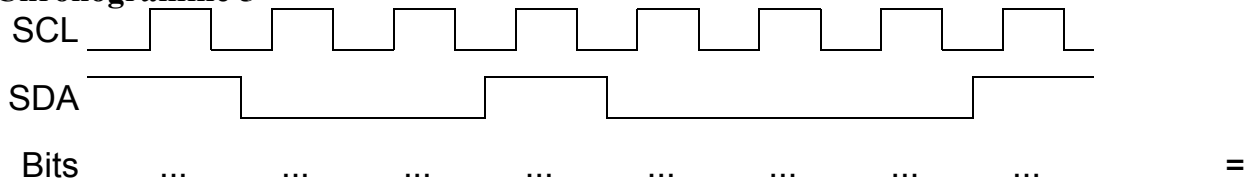
En résumé, à chaque fois que SCL est inactif (état haut), un bit est disponible sur la ligne SDA.

Je te propose de chercher, dans les chronogrammes suivants, les bits qui ont été transmis par Toto.

Chronogramme 4



Chronogramme 5



Pour vérifier : dans le chronogramme 4, tu devrais avoir trouvé la valeur 52 et dans le chronogramme 5, la valeur était 145. Pour être sûr, va voir en annexe 1 : conversion entre binaire et décimal.

Récapi-tulatif	Les informations complexes transitent sur le bus de manière sérielle, en combinant des informations élémentaires (par groupe de 8 bits). A chaque fois que la ligne SCL est à l'état haut, un bit est disponible sur SDA.

3) L'accusé de réception : Tu m'entends ?

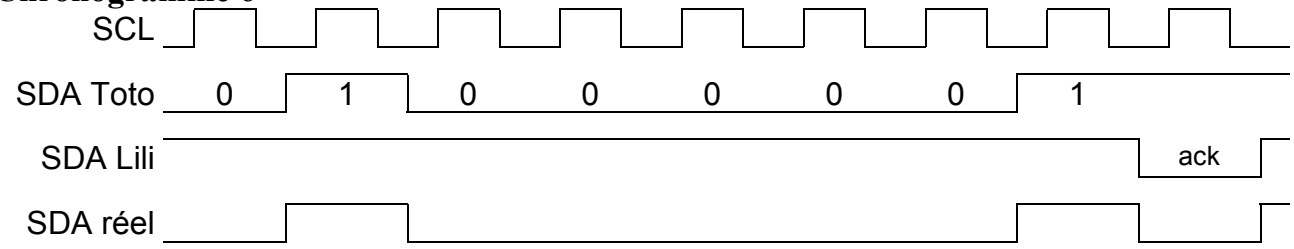
On sait comment Toto va s'y prendre pour transmettre tout plein d'informations en série à Lili. Mais comment Toto fait-il pour savoir si Lili a bien reçu les informations ? D'autant que les deux lignes sont occupées par Toto. Mais Toto est poli et accorde à Lili la permission de parler lorsqu'il le veut bien.

Le seul rôle de Lili est d'indiquer si oui ou non, elle a reçu le dernier octet en entier. On dit dans ce cas que Lili est esclave et Toto est maître.

Nous allons compliquer un peu notre chronogramme. Il va falloir différencier ce que fait Toto, ce que fait Lili et ce qui en résulte sur la ligne SDA. Dans ce cas simplifié, la ligne SCL est toujours contrôlée par Toto.

Toto veut toujours envoyer l'octet 65. A la fin de son envoi, il va laisser la parole à Lili.

Chronogramme 6



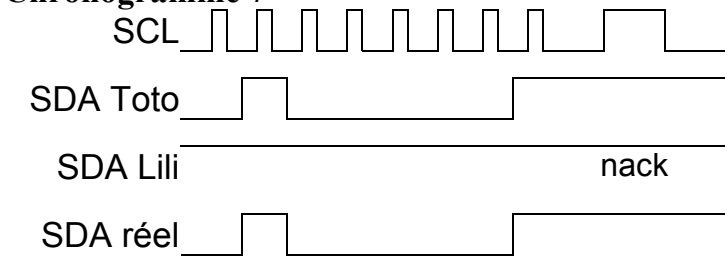
La transmission de l'octet est sans différence avec le chronogramme 3.

La nouveauté, c'est le signal '*ack*', signifiant *acknowledgement* ou signal d'acquittement.

Une fois transmis le huitième bit de l'octet, Toto laisse SDA inactif. De son côté, Lili sait que c'est à elle de jouer et impose SDA à l'état bas (actif). Toto rend SCL inactif et regarde sur SDA réel si Lili l'a mis en activité. Dans le chronogramme 6, Lili a bien reçu les 8 bits, elle transmet le signal d'acquittement. Toto est rassuré, il abaisse SCL et pourra continuer à transmettre ses données.

Lorsque Lili n'a pas reçu les informations correctement, elle ne produit pas le signal d'acquittement. Cela produit le chronogramme suivant :

Chronogramme 7



Le signal '*nack*', signifiant *no-acknowledgement*, est une absence de signal d'acquittement. Dans ce cas, Toto sait que Lili n'a pas reçu correctement les 8 derniers bits transmis. Il y a donc une demande d'acquittement après chaque octet transmis.

Je suis sûr qu'une question te dévore l'esprit : comment Lili et Toto peuvent-ils contrôler tous les deux la ligne SDA ?

Là, c'est la super astuce du protocole I²C.

Imagine que Toto et Lili soient des personnages dans une pièce sombre. Ils ont tous les deux une lampe de poche. S'ils ne font rien, la pièce n'est pas éclairée (SDA inactif ou état haut). Si Lili allume sa lampe (SDA Lili actif ou à état bas), la pièce est éclairée (SDA réel actif ou à l'état bas), quelle que soit l'action de Toto. C'est comme cela que fonctionne chacune des deux lignes I²C.

Cette astuce permet aux deux composants (Toto et Lili) de transmettre des informations tous les deux sur les mêmes lignes.

Récapitulatif	Lorsqu'un composant contrôle le bus I²C, il est maître. Il observe, après l'envoi de chaque octet, la présence d'un signal d'acquittement. Ce signal est produit par un esclave qui confirme par là qu'il a bien reçu l'octet en entier. Un esclave ne prend jamais d'initiative. Il répond seulement à des demandes d'un maître.

4) Un réseau, des adresses : Qui cause à qui ?

Toto est un petit filou. Non seulement il veut causer avec Lili, mais aussi avec Fleur. Bien entendu, il ne veut pas que ses messages arrivent aux deux composants. L'affaire se complique...

En pratique, imaginons que nous ayons un robot et que :

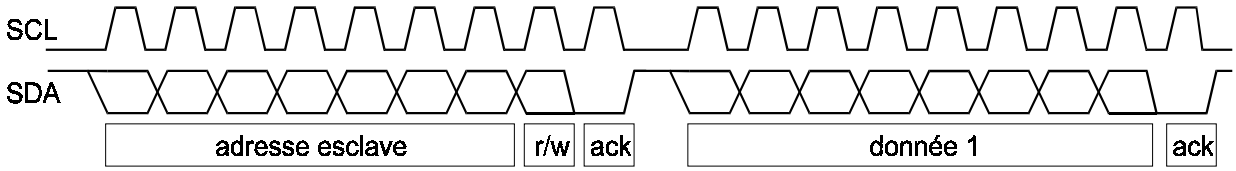
- Toto soit un ordinateur ou un microcontrôleur,
- Lili un périphérique esclave chargé du contrôle de la direction du robot,
- et Fleur un périphérique esclave chargé du contrôle de la vitesse du robot.

Il ne faudra pas confondre la vitesse et la direction, au risque de voir le robot se comporter comme s'il était en état d'ébriété.

Revenons à nos composants. Pour différencier ses deux esclaves, Toto commencera tous ses messages par une adresse, comme s'il écrivait un courrier. Pour cela, une adresse est affectée à chaque composant.

> A savoir : chaque composant électronique I²C reçoit une adresse lors de sa fabrication. Cette adresse a une partie fixe et une partie paramétrable pour permettre de combiner des composants identiques. Pour plus de renseignements, se reporter à l'annexe 3 : les composants I²C courants.

Chronogramme 8



<u>Remarque</u> : les croisillons sur la ligne SDA sont là pour signifier qu'il peut y avoir un changement d'état, suivant le bit que l'on veut envoyer.

Chaque message doit se composer d'un octet d'adressage, d'une demande d'acquittement, suivi des octets de données avec leurs acquittements (comme vu dans le point 3).

L'octet d'adressage se compose des 7 bits d'adresse de l'esclave puis d'un bit r/w (*read/write*). Ce bit informe l'esclave qu'il va recevoir ou émettre des données. Ce bit est mis, par le maître, à l'état actif lorsqu'il veut envoyer des données (écriture), et à l'état inactif lorsqu'il veut recevoir des données de la part de l'esclave (lecture).

Reprenons nos composants favoris. Toto veut envoyer un message à Fleur. Ce message consiste en une suite de 3 nombres : 65, 86 et 69. Imaginons que l'adresse de Fleur soit en binaire 0100 001x.

A savoir : Une adresse de composant s'écrit toujours avec 7 bits suivi d'un 'x' pour prévoir le bit 'r/w'. Cette façon d'écrire permet de manipuler un octet entier (et pas 7 bits seulement). De manière logiquement abusive mais admise, lorsqu'une adresse I²C est convertie en décimal, on fait comme si le 'x' est un 0. Ainsi, l'adresse de Fleur s'écrira en décimal 66.

Toto va envoyer l'adresse de Fleur 66 : **0100 001**x

Puis le bit indiquant qu'il va transmettre : **0**

Ensuite, il va attendre un signal d'acquittement. A ce moment là, Fleur comme Lili ont reçu le message, mais seule Fleur reconnaît son adresse. Lili se tait. Fleur envoie le signal d'acquittement (ack) : **0**

Toto est sûrement content de savoir qu'on l'écoute. Il va pouvoir transmettre ses données.

Il envoie le 65 : **0100 0001**

Il attend le signal d'acquittement, Fleur lui fournit parce qu'elle a bien reçu le dernier octet : **0.**

Toto envoie 86 : **0101 0110**

Signal d'acquittement ? Fleur a bien reçu, elle l'envoie : **0**.

Toto envoie 69 : **0100 0101**

Signal d'acquittement ? Fleur toujours à l'écoute répond : **0**.

On ne s'en lasse pas.

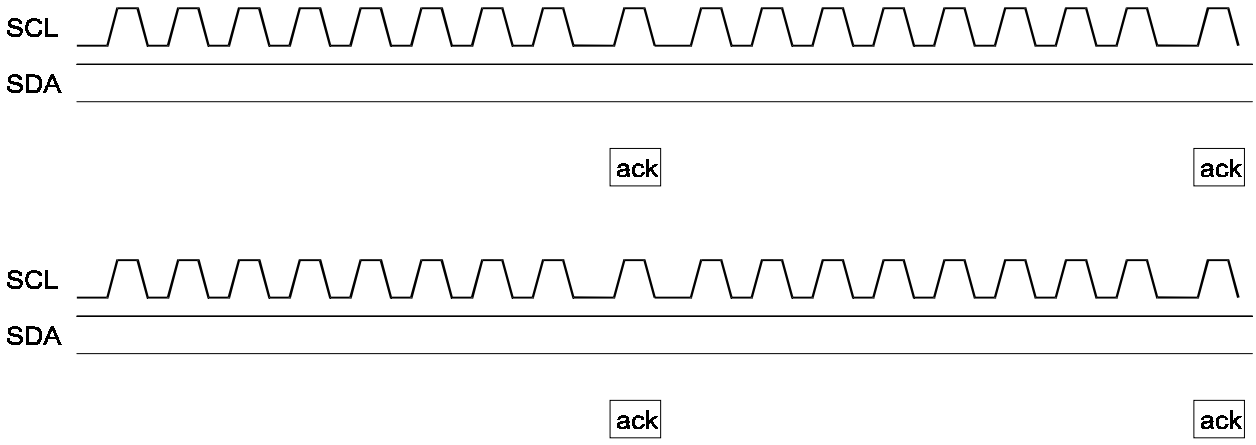
<u>Remarque</u> : Que se passerait-il si Fleur n'avait pas bien reçu les données ? Elle n'aurait pas envoyé un signal d'acquittement. Toto le saurait et pourrait alors faire une nouvelle tentative ou désespérer de ne pas être correctement écouté (mais c'est plus rare). Tout dépend du programmateur qui a défini le comportement de Toto.

Pendant tout ce temps, Lili est restée coite et inactive. Elle voit seulement sur le bus I²C qu'il y a des informations qui ne la concernent pas.

Pour t'amuser, tu peux compléter la ligne SDA (SDA réel) du chronogramme 9 de la conversation que Toto entretient avec Fleur. Pour t'aider, les créneaux de SCL sont regroupés par 8 puis 1 créneaux pour le signal d'acquittement.

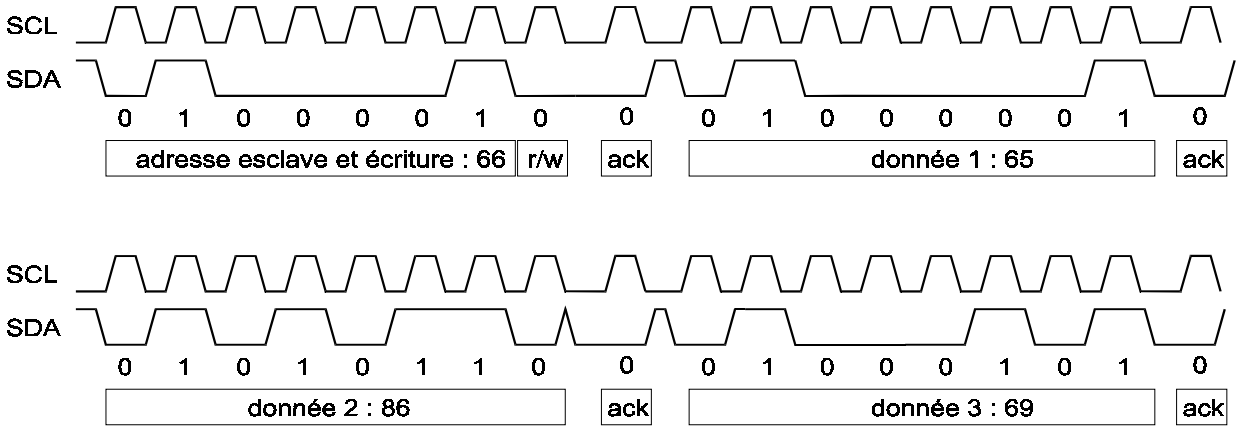
Indication : Les éléments en gras de la page précédente sont les seuls éléments à intégrer et à traduire en créneaux.

Chronogramme 9a



Une solution est à la page suivante.

Chronogramme 9b



<u>Remarque</u> : Avant et après les signaux d'acquittement, la ligne SDA revient à l'état haut. Souviens-toi de la pièce obscure et des deux personnes avec leur lampes : pour savoir si Lili va allumer sa lampe, il faut tout d'abord que Toto éteigne la sienne. De même, après que Lili a envoyé son signal d'acquittement, la pièce redevient obscure un moment.

Récapi-tulatif

Chaque composant I²C a une adresse. Lorsqu'un maître envoie un message, il commence par transmettre l'adresse du composant auquel il veut s'adresser puis le sens de la transmission (écriture ou lecture).

Deux questions te torturent de nouveau l'esprit, n'est-ce pas ?

➔ Comment va faire Toto pour ne plus parler à Fleur mais à Lili ?

➔ Si l'adresse de Lili est 0100000x (64), sachant que la première donnée transmise par Toto à Fleur est 01000001 (65), est-ce que Lili n'aurait pas le sentiment qu'on s'adresse à elle ?

Quelle serait ta proposition ? (Tu peux en cocher plusieurs.)

- ❒ Il suffit de rajouter un fil entre Toto et chaque esclave (pour l'enchaîner) qui préviendra ce dernier que c'est à lui que Toto veut causer.
- ❒ Tu n'as pas d'idées, mais tu me fais confiance pour découvrir ça tout de suite après.
- ❒ Il faut juste faire la différence entre l'adresse et les données, pour que Lili ne confonde pas.
- ❒ Autre (pas la peine de préciser).

Bravo, tu as de toute façon raison !

Mais sachant qu'une contrainte de départ est qu'il n'y a que deux fils reliant les composants, la première proposition n'est pas retenue.

5) Les conditions de départ et de fin

Une astuce supplémentaire permet aux composants de différencier l'adresse des données. L'adresse est toujours en début de message. Il faut donc savoir quand commence le message, et aussi quand il finit.

Remarque : par la suite, j'utiliserai indifféremment message et trame.

Nous allons découvrir la condition de départ et la condition de fin de trame.

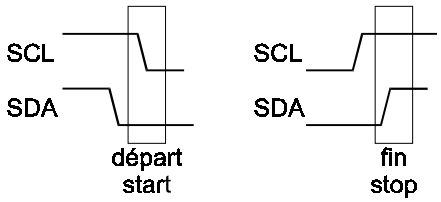
> Le protocole I²C nomme les conditions de départ et de stop réciproquement par *start condition* et *stop condition*.

Tu auras remarqué, judicieux(/se) que tu es, que la ligne SDA ne change d'état que lorsque SCL est actif (à l'état bas). Ça permet à Toto de préparer les données qu'il va envoyer.

Les deux seuls cas où SDA est modifié pendant que SCL est inactif sont ceux des conditions de départ et de stop. Et, comme tous les composants I²C ont été

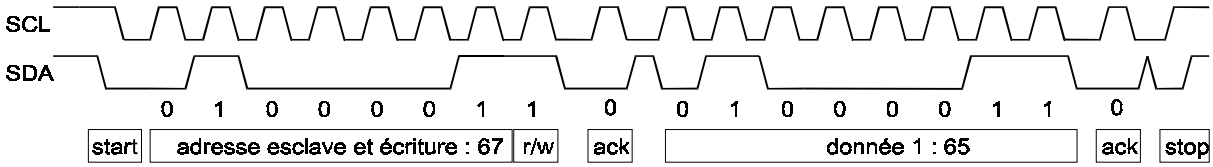
mis au courant dès leur fabrication, ils comprendront qu'il s'agit d'un début ou d'une fin de trame.

Chronogramme 10 : conditions de départ et de fin



Voilà ce que devient le début du chronogramme 9b. On imagine que Toto a le désir de n'envoyer plus qu'un octet.

Chronogramme 11



Au cours du point 4, je t'ai fait savoir que le maître transmet à la suite de l'adresse un bit signifiant à l'esclave qu'il doit être émetteur ou destinataire de données. Dans nos exemples, Toto envoyait toujours des données et n'en recevait jamais.

Mais maintenant, Toto veut demander à Lili de lui transmettre des données.

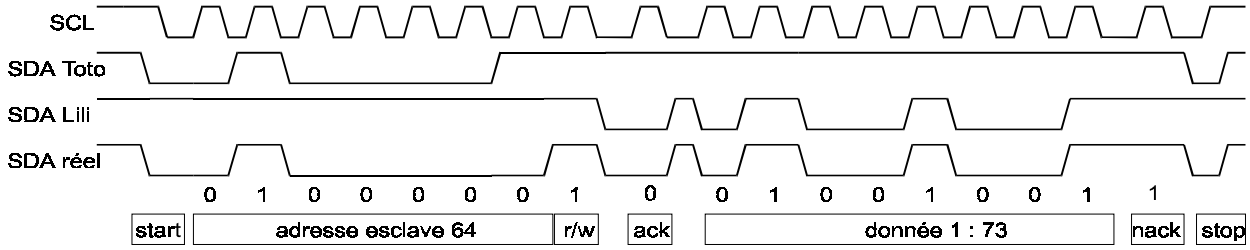
Il va commencer par une condition de départ, l'envoi de l'adresse de Lili (0100 000x) en positionnant le bit r/w à 1 pour une lecture.

Si tout va bien, Lili répond avec un signal d'acquittement.

Et là, comme c'était le cas pour le signal d'acquittement (voir le point 3), l'esclave Lili va transmettre ses données sur la ligne SDA selon l'horloge contrôlée par Toto par la ligne SCL. Pour bien comprendre, je te propose de revoir un chronogramme en dissociant SDA écrit par Toto, SDA écrit par Lili et la ligne SDA qui en résulte et qui est lue par les deux composants.

Pour tout saisir, je te rappelle que l'adresse de Lili est 64 ou 0100000x. Que la donnée qu'elle contient est 73 ou 01001001.

Chronogramme 12



La particularité de cette trame, c'est que pendant la transmission de la donnée, c'est Lili l'esclave qui a contrôlé la ligne SDA. Toto pendant ce temps, a produit les créneaux SCL et lu les données de Lili.

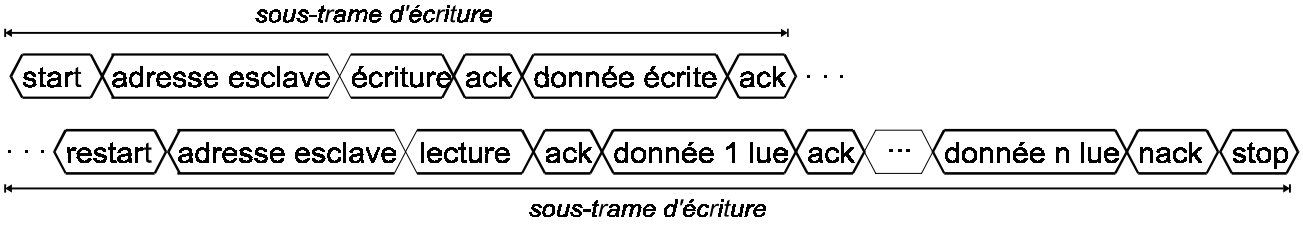
Ta perspicacité n'aura pas laissé s'échapper le 'nack' ou non-acquittement produit par Toto à la suite de la donnée 1 transmise par Lili. Cela ne veut pas dire que Toto n'a pas bien reçu les 8 bits de la donnée, mais il veut signaler à Lili qu'elle doit s'arrêter de transmettre ses données. Si Toto avait envoyé un acquittement, Lili aurait continué à envoyer des données.

<u>Remarque</u> : Certains composants I²C esclaves n'ont qu'une donnée pertinente à transmettre. D'autres ont plus de conversation, il leur est par conséquent utile de pouvoir transmettre plusieurs octets à la suite.

Il est courant, au pays de l'I²C, qu'un maître comme Toto désire écrire une donnée à un esclave comme Fleur puis lui demander des données en retour. Souvent, la première écriture est là pour définir ce que sera la demande de données.

Les trames ressemblent dans ces cas à la trame suivante :

Trame 13



Qu'y voit-on ?

On a reconnu le *start* de départ et le *stop* de fin.

La première partie de la trame débute par le passage de l'adresse de l'esclave avec le bit r/w en écriture (c'est-à-dire à l'état bas), suivi d'un signal d'acquittement de l'esclave. Le maître envoie sa donnée en écriture. L'esclave signale l'acquittement.

Et là, du neuf : ***restart***. Et oui, c'est une nouvelle condition de départ, sans stop avant. Toto garde le contrôle de la ligne I²C parce qu'il n'en n'a pas fini. Mais la sous-trame est terminée. Pour en réamorcer une autre, il fait comme s'il commençait une nouvelle trame.

Et on repart : envoi de la même adresse de Lili, avec cette fois le bit r/w à l'état haut pour signaler une demande de lecture. Lili s'est de nouveau reconnue, elle envoi son signal d'acquittement. Ensuite, c'est à elle de transmettre ses données. Le maître Toto dispense ses signaux d'acquittement. Quand il en a assez, il envoi un signal de non-acquittement. Lili s'arrête. Il ne reste plus, pour Toto, qu'à envoyer une condition de stop pour achever la trame composée.

<u>Remarque</u> : Ce modèle de communication est extrêmement fréquent. Bien entendu, on peut la compléter pour permettre au maître d'envoyer plusieurs données à l'esclave avant d'entamer la sous-trame de lecture. On aurait alors un modèle de communication complet.

Récapitulatif

Dans le cas d'une lecture, un esclave cesse d'envoyer des données lorsque le maître n'a pas envoyé de signal d'acquittement.

Une trame débute par une condition de départ (*start*) et se termine par une condition de stop. Un maître peut enchaîner plusieurs trames en renvoyant une condition de départ (restart) à la suite d'une première trame.

6) Politesse et multi-maîtres

Il reste à comprendre pourquoi Toto envoie une condition de stop alors qu'il pourrait très bien ne se servir que de '*restart'*.

En fait, tout est histoire de politesse. Un maître est, en principe, prévu pour libérer la ligne I²C sitôt qu'il n'en a plus besoin, pour laisser la place à... un autre maître qui en aurait besoin.

Et oui, la communication I²C prévoit non seulement une construction en réseau d'esclaves, mais aussi de maîtres. Comme sur un réseau informatique de type Ethernet, si tu connais, il y a partage de la ligne.

Avant d'utiliser la ligne I²C, un maître doit observer si elle est disponible. Pour simplifier, disons qu'il peut le savoir en observant si les lignes SCL et SDA sont inactives. Si c'est le cas, il peut prendre le contrôle. Comme il aime bien voir sa ligne disponible, il doit penser aux autres maîtres connectés au bus I²C qui sont comme lui. C'est pour cette raison qu'à la fin de son utilisation, chaque maître doit envoyer une condition de stop. Cette opération rend inactives les deux lignes SCL et SDA. N'importe quel maître peut alors prendre la ligne.

Non seulement les esclaves ont une adresse, mais les maîtres aussi. Cette qualité permet à un maître de s'adresse à un autre maître pour lui transmettre ou lui demander des informations.

Nous n'entreront pas dans les détails de l'utilisation multi-maîtres du bus I²C.

Pénultième remarque : je t'épargne la revue des conflits possibles dans une communication I²C, ainsi que leur gestion par les différents composants. Sache que c'est une force du bus I²C. Peut-être pour un article ultérieur qui parlerait de la mise au point d'un logiciel de contrôle du bus.

Récapitulatif

Il est possible de connecter sur une même ligne I²C plusieurs composants I²C maîtres en plus des composants I²C esclaves.

7) Avantages et inconvénients du bus I²C

Le bus I²C, inventé par Philips Composants est une réussite. Il est très économique, contrairement à ce que tu pourrais croire, facile à utiliser. Il permet de connecter entre eux des modules électroniques éloignés de quelques mètres. De nombreux composants I²C existent, couvrant la plupart des utilisations courantes en électronique.

En revanche, il est assez lent. Il ne vaut mieux pas compter, par exemple, sur la transmission de son numérisé par le bus. Il n'est pas fait pour cela.

A savoir : les données peuvent transiter à la vitesse de 100 Kbits par seconde. Mais il faut savoir que parmi les données qui transitent, la moitié environ sont « utiles ». Le reste, ce sont des adresses d'esclaves, des conditions et signaux divers de contrôle de la communication.

Dernière remarque : comme moi, tu t'es peut-être aperçu que les esclaves sont toujours des être féminins et les maîtres des êtres masculins. Que ce soit le hasard ou le fruit d'une tradition phallocratique bien ancrée, tu as le droit de faire les ratures qui s'imposent. Tu peux aussi écrire tes remarques à l'adresse suivante : ld@anstj.mime.univ-paris8.fr. Tu pourras en profiter pour poser tes questions sur le protocole I²C ou sur son utilisation.

A savoir : personne n'est parfait.

Rédacteur : Mgr DAYOT.

Annexe 1 : conversion entre binaire et décimal

Décimal	Binaire
0	0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 1 1
4	0 0 0 0 0 1 0 0
5	0 0 0 0 0 1 0 1
6	0 0 0 0 0 1 1 0
7	0 0 0 0 0 1 1 1
8	0 0 0 0 1 0 0 0
9	0 0 0 0 1 0 0 1
10	0 0 0 0 1 0 1 0
11	0 0 0 0 1 0 1 1
12	0 0 0 0 1 1 0 0
13	0 0 0 0 1 1 0 1
14	0 0 0 0 1 1 1 0
15	0 0 0 0 1 1 1 1
16	0 0 0 1 0 0 0 0
17	0 0 0 1 0 0 0 1
18	0 0 0 1 0 0 1 0
19	0 0 0 1 0 0 1 1
20	0 0 0 1 0 1 0 0
21	0 0 0 1 0 1 0 1
22	0 0 0 1 0 1 1 0
23	0 0 0 1 0 1 1 1
24	0 0 0 1 1 0 0 0
25	0 0 0 1 1 0 0 1
26	0 0 0 1 1 0 1 0
27	0 0 0 1 1 0 1 1
28	0 0 0 1 1 1 0 0
29	0 0 0 1 1 1 0 1
30	0 0 0 1 1 1 1 0
31	0 0 0 1 1 1 1 1
32	0 0 1 0 0 0 0 0
33	0 0 1 0 0 0 0 1
34	0 0 1 0 0 0 1 0
35	0 0 1 0 0 0 1 1
36	0 0 1 0 0 1 0 0
37	0 0 1 0 0 1 0 1
38	0 0 1 0 0 1 1 0
39	0 0 1 0 0 1 1 1
40	0 0 1 0 1 0 0 0
41	0 0 1 0 1 0 0 1
42	0 0 1 0 1 0 1 0
43	0 0 1 0 1 0 1 1
44	0 0 1 0 1 1 0 0
45	0 0 1 0 1 1 0 1
46	0 0 1 0 1 1 1 0
47	0 0 1 0 1 1 1 1
48	0 0 1 1 0 0 0 0
49	0 0 1 1 0 0 0 1
50	0 0 1 1 0 0 1 0
51	0 0 1 1 0 0 1 1
52	0 0 1 1 0 1 0 0
53	0 0 1 1 0 1 0 1
54	0 0 1 1 0 1 1 0
55	0 0 1 1 0 1 1 1
56	0 0 1 1 1 0 0 0
57	0 0 1 1 1 0 0 1
58	0 0 1 1 1 0 1 0
59	0 0 1 1 1 0 1 1
60	0 0 1 1 1 1 0 0
61	0 0 1 1 1 1 0 1
62	0 0 1 1 1 1 1 0
63	0 0 1 1 1 1 1 1
64	0 1 0 0 0 0 0 0
65	0 1 0 0 0 0 0 1
66	0 1 0 0 0 0 1 0
67	0 1 0 0 0 0 1 1
68	0 1 0 0 0 1 0 0
69	0 1 0 0 0 1 0 1
70	0 1 0 0 0 1 1 0
71	0 1 0 0 0 1 1 1
72	0 1 0 0 1 0 0 0
73	0 1 0 0 1 0 0 1
74	0 1 0 0 1 0 1 0
75	0 1 0 0 1 0 1 1
76	0 1 0 0 1 1 0 0
77	0 1 0 0 1 1 0 1
78	0 1 0 0 1 1 1 0
79	0 1 0 0 1 1 1 1
80	0 1 0 1 0 0 0 0
81	0 1 0 1 0 0 0 1
82	0 1 0 1 0 0 1 0
83	0 1 0 1 0 0 1 1
84	0 1 0 1 0 1 0 0
85	0 1 0 1 0 1 0 1
86	0 1 0 1 0 1 1 0
87	0 1 0 1 0 1 1 1
88	0 1 0 1 1 0 0 0
89	0 1 0 1 1 0 0 1
90	0 1 0 1 1 0 1 0
91	0 1 0 1 1 0 1 1
92	0 1 0 1 1 1 0 0
93	0 1 0 1 1 1 0 1
94	0 1 0 1 1 1 1 0
95	0 1 0 1 1 1 1 1
96	0 1 1 0 0 0 0 0
97	0 1 1 0 0 0 0 1
98	0 1 1 0 0 0 1 0
99	0 1 1 0 0 0 1 1
100	0 1 1 0 0 1 0 0
101	0 1 1 0 0 1 0 1
102	0 1 1 0 0 1 1 0
103	0 1 1 0 0 1 1 1
104	0 1 1 0 1 0 0 0
105	0 1 1 0 1 0 0 1
106	0 1 1 0 1 0 1 0
107	0 1 1 0 1 0 1 1
108	0 1 1 0 1 1 0 0
109	0 1 1 0 1 1 0 1
110	0 1 1 0 1 1 1 0
111	0 1 1 0 1 1 1 1
112	0 1 1 1 0 0 0 0
113	0 1 1 1 0 0 0 1
114	0 1 1 1 0 0 1 0
115	0 1 1 1 0 0 1 1
116	0 1 1 1 0 1 0 0
117	0 1 1 1 0 1 0 1
118	0 1 1 1 0 1 1 0
119	0 1 1 1 0 1 1 1
120	0 1 1 1 1 0 0 0
121	0 1 1 1 1 0 0 1
122	0 1 1 1 1 0 1 0
123	0 1 1 1 1 0 1 1
124	0 1 1 1 1 1 0 0
125	0 1 1 1 1 1 0 1
126	0 1 1 1 1 1 1 0
127	0 1 1 1 1 1 1 1
128	1 0 0 0 0 0 0 0
129	1 0 0 0 0 0 0 1
130	1 0 0 0 0 0 1 0
131	1 0 0 0 0 0 1 1
132	1 0 0 0 0 1 0 0
133	1 0 0 0 0 1 0 1
134	1 0 0 0 0 1 1 0
135	1 0 0 0 0 1 1 1
136	1 0 0 0 1 0 0 0
137	1 0 0 0 1 0 0 1
138	1 0 0 0 1 0 1 0
139	1 0 0 0 1 0 1 1
140	1 0 0 0 1 1 0 0
141	1 0 0 0 1 1 0 1
142	1 0 0 0 1 1 1 0
143	1 0 0 0 1 1 1 1
144	1 0 0 1 0 0 0 0
145	1 0 0 1 0 0 0 1
146	1 0 0 1 0 0 1 0
147	1 0 0 1 0 0 1 1
148	1 0 0 1 0 1 0 0
149	1 0 0 1 0 1 0 1
150	1 0 0 1 0 1 1 0
151	1 0 0 1 0 1 1 1
152	1 0 0 1 1 0 0 0
153	1 0 0 1 1 0 0 1
154	1 0 0 1 1 0 1 0
155	1 0 0 1 1 0 1 1
156	1 0 0 1 1 1 0 0
157	1 0 0 1 1 1 0 1
158	1 0 0 1 1 1 1 0
159	1 0 0 1 1 1 1 1
160	1 0 1 0 0 0 0 0
161	1 0 1 0 0 0 0 1
162	1 0 1 0 0 0 1 0
163	1 0 1 0 0 0 1 1
164	1 0 1 0 0 1 0 0
165	1 0 1 0 0 1 0 1
166	1 0 1 0 0 1 1 0
167	1 0 1 0 0 1 1 1
168	1 0 1 0 1 0 0 0
169	1 0 1 0 1 0 0 1
170	1 0 1 0 1 0 1 0
171	1 0 1 0 1 0 1 1
172	1 0 1 0 1 1 0 0

173	1	0	1	0	1	1	0	1
174	1	0	1	0	1	1	1	0
175	1	0	1	0	1	1	1	1
176	1	0	1	1	0	0	0	0
177	1	0	1	1	0	0	0	1
178	1	0	1	1	0	0	1	0
179	1	0	1	1	0	0	1	1
180	1	0	1	1	0	1	0	0
181	1	0	1	1	0	1	0	1
182	1	0	1	1	0	1	1	0
183	1	0	1	1	0	1	1	1
184	1	0	1	1	1	0	0	0
185	1	0	1	1	1	0	0	1
186	1	0	1	1	1	0	1	0
187	1	0	1	1	1	0	1	1
188	1	0	1	1	1	1	0	0
189	1	0	1	1	1	1	0	1
190	1	0	1	1	1	1	1	0
191	1	0	1	1	1	1	1	1
192	1	1	0	0	0	0	0	0
193	1	1	0	0	0	0	0	1
194	1	1	0	0	0	0	1	0
195	1	1	0	0	0	0	1	1
196	1	1	0	0	0	1	0	0
197	1	1	0	0	0	1	0	1
198	1	1	0	0	0	1	1	0
199	1	1	0	0	0	1	1	1
200	1	1	0	0	1	0	0	0

201	1	1	0	0	1	0	0	1
202	1	1	0	0	1	0	1	0
203	1	1	0	0	1	0	1	1
204	1	1	0	0	1	1	0	0
205	1	1	0	0	1	1	0	1
206	1	1	0	0	1	1	1	0
207	1	1	0	0	1	1	1	1
208	1	1	0	1	0	0	0	0
209	1	1	0	1	0	0	0	1
210	1	1	0	1	0	0	1	0
211	1	1	0	1	0	0	1	1
212	1	1	0	1	0	1	0	0
213	1	1	0	1	0	1	0	1
214	1	1	0	1	0	1	1	0
215	1	1	0	1	0	1	1	1
216	1	1	0	1	1	0	0	0
217	1	1	0	1	1	0	0	1
218	1	1	0	1	1	0	1	0
219	1	1	0	1	1	0	1	1
220	1	1	0	1	1	1	0	0
221	1	1	0	1	1	1	0	1
222	1	1	0	1	1	1	1	0
223	1	1	0	1	1	1	1	1
224	1	1	1	0	0	0	0	0
225	1	1	1	0	0	0	0	1
226	1	1	1	0	0	0	1	0
227	1	1	1	0	0	0	1	1
228	1	1	1	0	0	1	0	0

229	1	1	1	0	0	1	0	1
230	1	1	1	0	0	1	1	0
231	1	1	1	0	0	1	1	1
232	1	1	1	0	1	0	0	0
233	1	1	1	0	1	0	0	1
234	1	1	1	0	1	0	1	0
235	1	1	1	0	1	0	1	1
236	1	1	1	0	1	1	0	0
237	1	1	1	0	1	1	0	1
238	1	1	1	0	1	1	1	0
239	1	1	1	0	1	1	1	1
240	1	1	1	1	0	0	0	0
241	1	1	1	1	0	0	0	1
242	1	1	1	1	0	0	1	0
243	1	1	1	1	0	0	1	1
244	1	1	1	1	0	1	0	0
245	1	1	1	1	0	1	0	1
246	1	1	1	1	0	1	1	0
247	1	1	1	1	0	1	1	1
248	1	1	1	1	1	0	0	0
249	1	1	1	1	1	0	0	1
250	1	1	1	1	1	0	1	0
251	1	1	1	1	1	0	1	1
252	1	1	1	1	1	1	0	0
253	1	1	1	1	1	1	0	1
254	1	1	1	1	1	1	1	0
255	1	1	1	1	1	1	1	1

Annexe 2 : les composants I²C courants

Composants maîtres

80C552 : microcontrôleur à interface I²C intégrée.

PCF8584 : contrôleur de bus I²C.

Composants esclaves

PCF8566 et PCF8576 : pilote universel d'afficheur à cristaux liquides.

PCF8570 : mémoire RAM statique 128 x 8 bits/256 x 8 bits.

PCF8573 : horloge-calendrier en temps réel.

PCF8574 : extension à huit entrées-sorties numériques.

PCF8583 : horloge-calendrier avec RAM statique.

PCF8591 : convertisseur analogique/numérique et numérique/analogique.

ST24C08 et X24645 : mémoire EPROM

Annexe 3 : les extensions du bus I²C

- Le bus traditionnel permet de communiquer à une vitesse maximale de 100 Kbits par seconde. Une extension du bus a permis de passer à 400 Kbits par seconde. Les auteurs ont pu tester le bus I²C à une vitesse encore

légèrement supérieure en pilotant la ligne en logiciel pur, les composants esclaves ont suivi.

- L'adressage sur 7 bits permet en théorie la connexion de 128 composants sur un même bus. Il est toujours possible de ruser pour connecter en cascade plusieurs composants en modifiant dynamiquement leur adresse. Ceci permet d'étendre les possibilités.
- Ce mode d'adressage a connu également des développements. Des composants répondent maintenant à un mode d'adressage sur plus de 7 bits.
- le Bus CAN a été développé par Philips Composants. Beaucoup plus évolué, il est particulièrement bien adapté à la mise en connexion de plusieurs réseaux I²C locaux entre eux.
- Régulièrement, de nouveaux composants et microcontrôleurs incluant l'I²C apparaissent sans arrêt sur le marché, répondant à des besoins les plus divers : vidéo, sons, mémoires, téléphonie, HIFI, domotique... On n'a pas fini d'en parler.

Annexe 4 : bibliographie

- PARET Dominique, Le Bus I²C, de la théorie à la pratique, Ed. DunodTech.
- BRODIER Jean-Paul, Le manuel du bus I²C, théorie et pratique avec applications Elekto. Ed. PubliTronic.
- I²C Peripherals, Data Handbook IC12, Philips Semiconductors.
- De nombreux numéros de Microbe publiés par l'ANSTJ - Sciences Techniques Jeunesse.