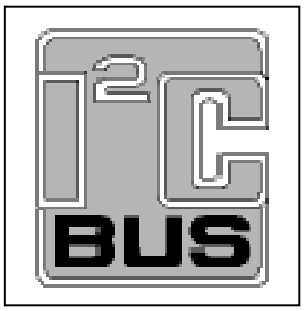
SOMMAIRE

1) INTRODUCTION



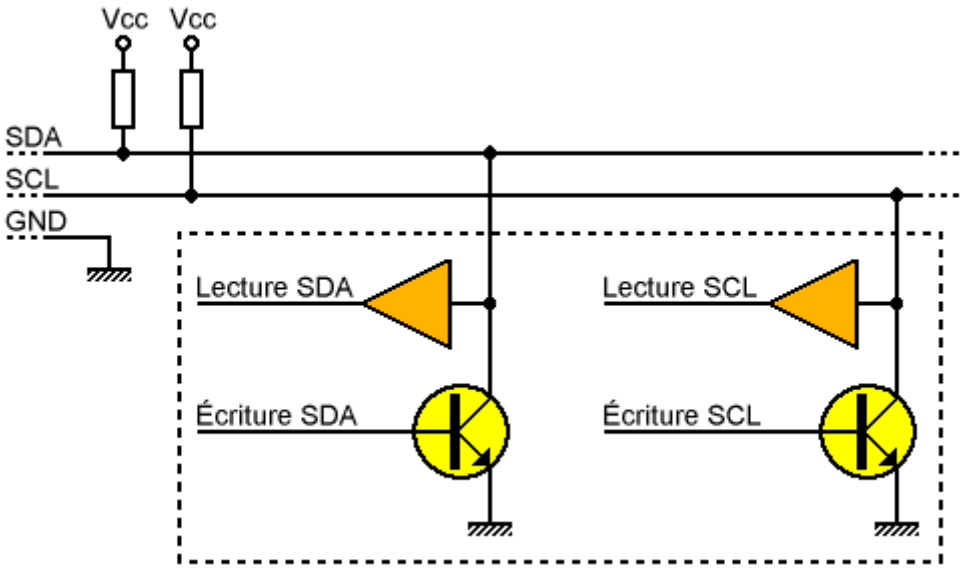
Le bus I ² C (Inter Integrated Circuit) fait partie des bus série : 3 fils pour faire tout passer. Il a été développé au début des années 1980, par Philips pour minimiser les liaisons entre les circuits intégrés numériques de ses produits (Téléviseurs, éléments Hi Fi, magnétoscopes, ...).
Aujourd'hui, Philips a dans son catalogue plus de 150 circuits intégrés CMOS et bipolaires qui sont compatibles I ² C. D'autres fabricants ont aussi développé des circuits intégrés qui peuvent être connectés au bus I ² C. Le protocole utilisé est simple et rapide.
Que demander de plus ? Multi master ? Pas de problèmes, c'est l'une des caractéristiques standard du bus I²C.

2) CARACTERISTIQUES

Le bus I ² C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils : un signal de données (SDA), un signal d'horloge (SCL), et un signal de référence électrique (masse).
Il s'agit d'une liaison en mode série, ce qui signifie que la vitesse de transfert sera plus faible qu'avec un bus de type parallèle. Le bus I ² C permet cependant des échanges à la vitesse de 100 k bits par seconde. Certes, la vitesse de transfert du bus I ² C n'est pas fulgurante, mais dans bien des cas, la vitesse n'est pas l'élément prédominant.
L'utilisation d'un bus I ² C permet de réduire la complexité des circuits imprimés à réaliser. Par exemple, pour connecter une EEPROM ou une RAM à un microcontrôleur classique, il faut relier entre eux les bits de données et les bits d'adresses des différents composants, et, en plus, il faut bien souvent ajouter une logique de sélection. Avec des composants prévus pour se connecter au bus I ² C, il suffit de les relier par deux pistes seulement. Si, par la suite, on souhaite ajouter des composants sur le circuit, le nombre de pistes à ajouter sera vraiment plus réduit (essayez d'ajouter une EEPROM sur un circuit existant pour voir).
De nombreux fabricants ayant adopté le système la variété des systèmes disponibles disposant d'un port I ² C est énorme :
microcontrôleurs
expandeurs de bus (entrée/sortie 8 bits)
convertisseurs A/N et N/A
mémoires (RAM, EPROM, EEPROM, etc.)
récepteurs infrarouges (télécommande RC5)
capteurs de température
circuits audio (égaliseur, contrôle de volume, etc.)
drivers d'affichage LCD ou à LEDS
décodeurs télétexte
chargeurs de batterie
PLL pour tuner HF
etc.
Le nombre de composants qu'il est possible de relier est essentiellement limité par la charge capacitive des lignes SDA et SCL : **400 pF**.
Le bus I²C a encore le vent en poupe car il est de plus en plus utilisé dans l'électronique grand public, parfois déguisé sous une norme pour des besoins particuliers comme le SMBus qui est implanté dans tous les nouveaux PC, ou encore le fameux DDC qui équipe tous les moniteurs et cartes vidéos récents.

2.1) Le support physique utilisé

Comme indiqué précédemment, pour se connecter à un bus I²C il faut une masse, et deux fils de communication. Le premier fil, SDA (**S**ignal **DA**ta), est utilisé pour transmettre les données. L'autre fil, SCL (**S**ignal **CL**ock) est utilisé pour transmettre un signal d'horloge synchrone (signal qui indique le rythme d'évolution de la ligne SDA). Les tensions associées aux niveaux logiques vont dépendre de la technologie des circuits en présence (CMOS, TTL). Il faudra que tous les circuits connectés au bus I²C utilisent les mêmes potentiels pour définir les niveaux haut et bas. En définitive, cela implique que tous les composants connectés à un même bus soient alimentés de façon identique. Cela ne signifie pas que les composants doivent utiliser la même source pour s'alimenter ; il suffit que la tension d'alimentation soit à la même valeur pour tous les composants, le fil de masse permettant d'unifier les références.
Il reste maintenant un problème crucial. Comment permettre à plusieurs circuits logiques de connecter leurs sorties ensemble, sachant que certains circuits voudront imposer un niveau haut tandis que d'autres voudront imposer un niveau bas ? La réponse est connue depuis longtemps. Il faut utiliser des sorties à collecteur ouvert (ou à drain ouvert pour des circuits CMOS). Le niveau résultant sur la ligne est alors une fonction « ET » de toutes les sorties connectées.



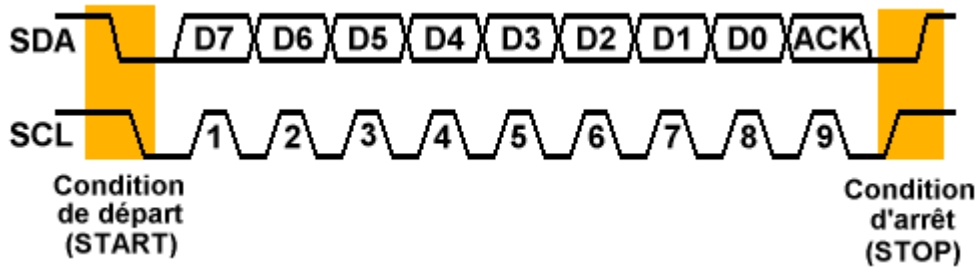
Les résistances de rappel au potentiel VCC permettent aux signaux SDA et SCL d'être à 1 si toutes les sorties à, collecteurs ouverts sont aussi au niveau 1 (résultat de la fonction « ET »). Si une ou plusieurs sorties tentent d'imposer un niveau bas sur une ligne, le ou les transistors associés vont conduire, ce qui entraîne un niveau bas sur la ligne correspondante (ce qui est conforme au résultat de la fonction « ET »).
En ce qui concerne la lecture des signaux SDA et SCL, cela ne pose pas de problème. Les signaux peuvent être lus en permanence sans risque d'interférer sur le niveau de la ligne.
Au repos, tous les circuits connectes doivent imposer un niveau haut sur leurs sorties respectives. Si les lignes SDA et SCL sont au niveau haut dans ces conditions, cela signifie qu'aucun circuit ne tente de prendre le contrôle du bus. Si une des lignes SDA ou SCL passe à un niveau bas dans les mêmes conditions, c'est qu'un des circuits désire prendre le contrôle du bus. Mais il peut aussi y avoir deux circuits qui tentent de prendre le contrôle du bus en même temps (ou à quelques nanosecondes d'écart près). Il faut donc mettre en place un protocole pour gérer les conflits possibles.

2.2) Le protocole I²C

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

2.2.1) La prise de contrôle du bus

Pour transmettre des données sur le bus I ² C, il faut surveiller deux conditions particulières : la condition de départ et la condition d'arrêt.



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

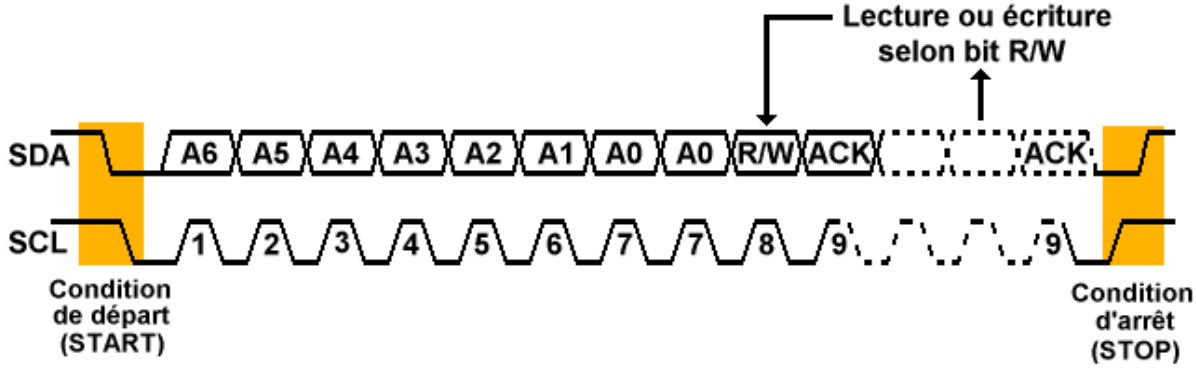
La transmission d'un octet

Avant de placer les bits qui forment l'octet à transmettre sur le bus, le maître doit placer la ligne d'horloge SCL à 0. Tant que la ligne SCL est au niveau haut, la ligne SDA ne doit pas changer d'état, sinon cette condition sera interprétée comme la condition d'arrêt. La condition arrêt peut survenir même au milieu de la transmission d'un octet, pour abandonner la transmission et libérer le bus pour les autres circuits. Pour transmettre correctement les bits sur la ligne SDA, le maître doit donc tout d'abord placer la ligne SCL à 0. Ensuite, le maître peut placer la ligne SDA au niveau correspondant au bit à transmettre et replacer la ligne SCL au niveau 1 pour indiquer que le bit est présent sur la ligne SDA. La même opération va se répéter autant de fois que nécessaire pour transmettre les 8 bits de données. Notez que c'est le bit de poids fort qui est transmis en premier.

Une fois les 8 bits transmis, le circuit qui vient de recevoir les données doit imposer un bit d'acquittement ACK sur la ligne SDA. Pour cela, pendant que la ligne SCL est au niveau bas, le maître place sa propre sortie au niveau haut, tandis que le récepteur (aussi appelé l'esclave) place sa sortie au niveau bas. Puisque les sorties sont à collecteur ouvert, la ligne SDA restera au niveau bas à cause de l'esclave. Le maître relit ensuite la ligne SDA une fois qu'il a passé la ligne SCL au niveau haut. Si la valeur lue pour le bit ACK est 0, c'est que l'esclave s'est bien acquitté de l'octet reçu, sinon c'est qu'il y a une erreur et le maître doit générer la condition arrêt.

2.2.2) La transmission d'une addresse

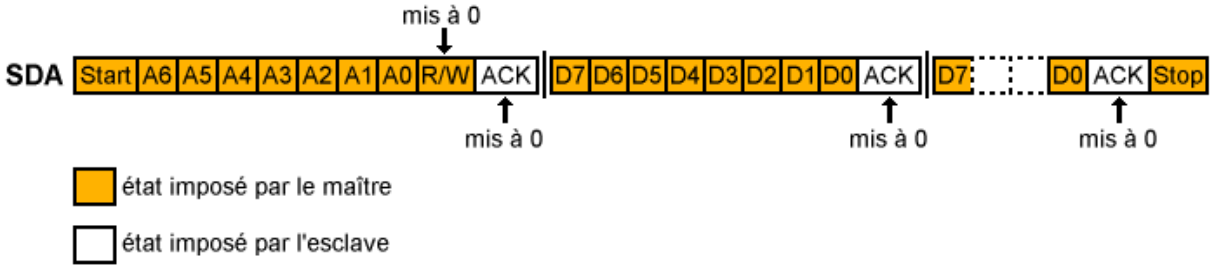
Le nombre de composants qu'il est possible de connecter sur un bus I²C étant largement supérieur à deux, le maître doit pouvoir choisir quel esclave est censé recevoir les données. Dans ce but, le premier octet que transmet le maître n'est pas une donnée mais une adresse. Le format de l'octet d'adresse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



Chaque circuit connecté au bus I²C possède une adresse, qui doit être unique. L'adresse associée à un composant est définie en partie par l'état de broches de sélections et d'autre part par sa fonction. Par exemple, le circuit PCF8574, qui est un port d'entrées/sorties bidirectionnel 8 bits, décompose son adresse de la façon suivante : [0] [1] [0] [0] [A2] [A1] [A0] [R/$\overline{W}$]. Les bits A2, A1 et A0 reflètent l'état des broches 1, 2 et 3 du circuit. Cela permet de placer 8 circuits PCF8574 sur le bus I²C. Lors de la conception d'un système, il faut donc veiller à l'unicité des adresses attribuées aux différents composants. Une fois l'adresse envoyée sur le bus, l'esclave concerné doit répondre en plaçant le bit ACK à 0. Si le but ACK vaut 1, le maître comprend qu'il y a une erreur de sélection et il génère la condition arrêt. En revanche, si le bit ACK vaut 0, le maître peut continuer les opérations.
Note : Les adresses 0000 0xxx et 1111 11xx sont réservées à des modes de fonctionnement particuliers.

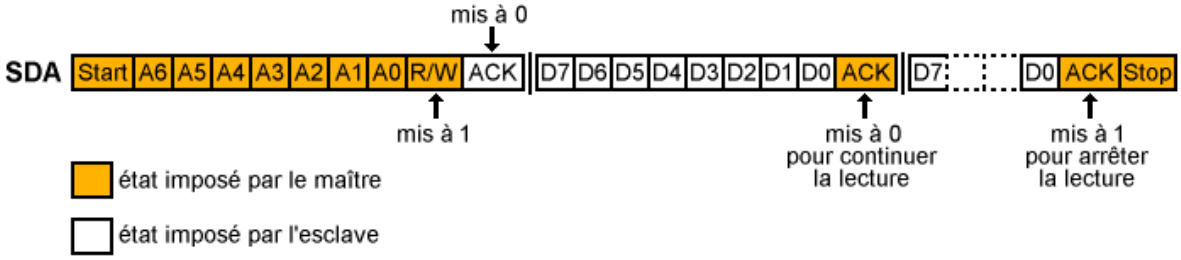
2.2.3) Écriture d'une donnée

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



2.2.4) Lecture d'une donnée

Si le bit R/$\overline{W}$ transmis en même temps que l'adresse est à 1, cela signifie que le maître veut lire des données issues de l'esclave. C'est toujours le maître qui va générer le signal d'horloge SCL. En revanche, après le bit ACK de l'adresse, c'est l'esclave qui va garder le contrôle de la ligne SDA. Pour cela, le maître va placer sa propre sortie SDA au niveau haut pour permettre à l'esclave de prendre le contrôle de la ligne SDA. L'esclave doit alors scruter la ligne SCL et attendre le niveau bas pour changer l'état de la ligne SDA, faute de quoi le maître détectera une condition arrêt et abandonnera le transfert (l'électronique intégrée dans l'esclave se doit de détecter aussi qu'il y a eu une condition arrêt, bien entendu).
Après que l'esclave a transmis les 8 bits de données, c'est le maître, cette fois-ci, qui va générer un bit d'acquittement. Si le maître désire lire des octets supplémentaires, il placera le bit d'acquittement à 0. En revanche, si le maître décide que la lecture est terminée, il placera le bit ACK au niveau 1. L'esclave comprendra alors que le transfert est terminé. Cette fois-ci, bien que le bit ACK soit au niveau 1, cela ne correspond pas à une condition d'erreur mais à une fin de transfert.

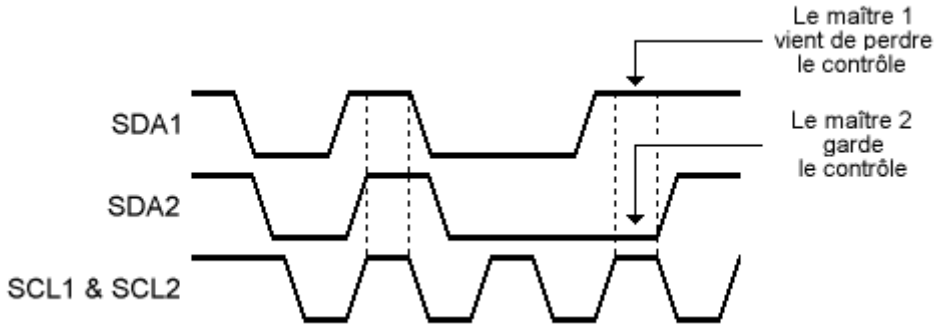


2.2.5) Restart

Le protocole du bus I²C ne s'arrête pas là. Il est possible d'enchaîner écriture et lecture de l'esclave sans avoir à passer par une condition arrêt. Par exemple, dans le cas de la lecture d'une RAM, le maître commence par envoyer l'adresse du composant avec le bit R/$\overline{W}$ positionné sur l'écriture. La RAM adressé, en esclave renvoie ACK = 0. Ensuite, le maître transmet l'adresse interne de la case mémoire demandée. Une fois encore, l'esclave répond par ACK = 0. Le maître envoie alors à nouveau une condition de départ (sans passer par une condition d'arrêt), puis de nouveau l'adresse du composant sélectionné mais en plaçant le bit R/$\overline{W}$ sur la position lecture. L'esclave va répondre par ACK = 0 et enchaîner par la transmission du contenu de la case mémoire demandée. C'est toujours le maître qui impose l'horloge SCL mais c'est l'esclave, en l'occurrence la RAM, qui contrôle la ligne SDA. Une fois les 8 bits de données transmis par la RAM, si le maître veut lire le contenu de la case mémoire suivante, il placera le bit ACK au niveau 0. Dans ce cas, la RAM recommence la lecture avec la case mémoire suivante. En revanche, si le maître souhaite en terminer avec la lecture, il placera le bit ACK au niveau 1 et il générera ensuite la condition arrêt.
Le contenu des octets de données lus ou écrits aura une signification qui dépend du composant sélectionné. Mais le protocole reste le même.

2.2.6) La gestion des conflits

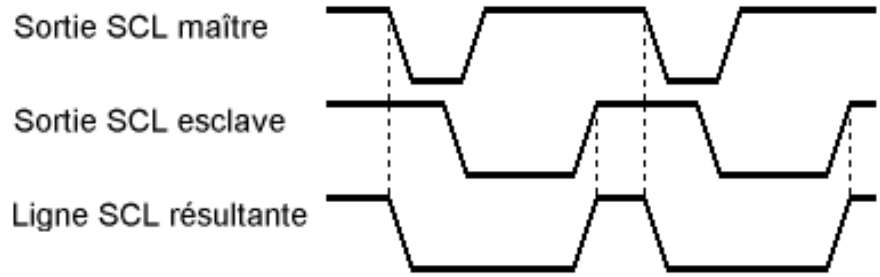
La structure même du bus I²C a été conçue pour pouvoir y accueillir plusieurs maîtres. Se pose alors le problème commun à tout les réseaux utilisant un canal de communication unique : la gestion des conflits. En effet, chaque maître pouvant prendre possession du bus dès que celui ci est libre, c'est à dire tant que les lignes SDA et SCL sont au niveau haut depuis suffisamment de temps, (4,7 µs), il existe le possibilité que deux maîtres prennent le contrôle du bus en même temps. Si cela ne pose pas de problème sur le plan électrique grâce à l'utilisation de collecteurs ouvert, il faut pouvoir détecter cet état de fait pour éviter la corruption des données transmises. Comment cela se passe-t-il ?
Une fois qu'il est établi que le bus est libre, le circuit qui souhaite prendre le contrôle du bus place la ligne SDA à 0 puis il relit l'état réel de la ligne SDA pour le comparer avec l'état qu'il souhaite imposer. Si dans un laps de temps très proche un autre maître décide aussi de prendre le contrôle du bus, il placera lui aussi la ligne SDA à 0. Du fait de la configuration à collecteur ouvert, le niveau de la ligne SDA sera à un niveau résultant d'une fonction "ET" entre les niveaux demandés par chaque maître. Pour l'instant, le résultat sera bien un niveau 0, de sorte qu'aucun maître ne peut savoir qu'il y a un conflit.
Chaque circuit maître va donc commencer normalement à générer le signal d'horloge, de sorte que l'état de la ligne SCL sera aussi le résultat d'une fonction « ET » des différents états imposés par chaque maître. Pendant l'état bas de la ligne SCL, chaque maître va modifier sa propre sortie SDA. Si tous les maîtres qui ont pris le contrôle du bus placent le même état sur la ligne SDA, le conflit n'étant pas visible, la transmission se poursuit normalement, comme si chacun était seul. En revanche, dès qu'un maître place un niveau différent des autres sur la ligne SDA, il y aura forcément un état bas sur la ligne SDA, tandis qu'un ou plusieurs maîtres souhaitaient imposer un niveau haut. Tous les maîtres qui ont demandé un niveau haut, tandis que la ligne SDA reste à 0, vont perdre immédiatement le contrôle du bus. A partir de cet instant, le ou les maîtres qui viennent de perdre le contrôle du bus vont continuer à lire les états successifs de la ligne SDA. Ils vont continuer la lecture comme ils l'auraient fait en tant qu'esclaves, pour le cas où le maître qui reste encore en course souhaite justement adresser l'un d'eux.



Si les différents maîtres en compétitions tentent d'adresser le même composant, la ligne SDA sera toujours au bon niveau. La procédure d'arbitration va alors continuer avec les données à transmettre. Si les différents maîtres qui adressaient le même composant écrivent la même donnée, le conflit n'a pas lieu d'exister puisque tout le monde veut faire la même chose. La procédure d'arbitration va alors se poursuivre jusqu'à ce que l'un des maîtres demande un niveau différent des autres. En poussant le raisonnement à l'extrême, on peut imaginer le cas où tous les maîtres vont demander les mêmes niveaux jusqu'à la condition d'arrêt. Mais la probabilité d'un tel cas est bien faible. Quoi qu'il en soit, si le cas se produit, la procédure d'arbitration garantit que le résultat final est correct, chaque maître ayant réalisé la même opération exactement en même temps.

Ralentissement de la vitesse

Il est possible de ralentir la vitesse de transmission du bus I²C. Si un circuit esclave a besoin de ralentir les échanges sur le bus, il lui suffit de maintenir la ligne SCL à état bas. Le circuit maître scrute en permanence la ligne SCL pour la comparer avec l'état qu'il souhaite lui-même imposer. Quand le circuit maître détecte un niveau bas tandis qu'il vient de placer sa sortie SCL à l'état haut, il passe dans une boucle d'attente. L'attente se poursuivra jusqu'à ce que la ligne SCL asse au niveau haut.



2.2.7) Les nouvelles spécifications

Face à l'explosion du nombre de circuits I²C disponibles en très forte augmentation, Philips a publié en **1993** les nouvelles spécifications de l'I²C :

* Compatibilité totale avec l'ancien I²C (qui date de 1982)
* Vitesse de **400 k bit/s**
* **Adressage étendu sur 10 bits** (jusqu'à 1024 circuits) répartis dans deux octets de la façon suivante :

* Utilisation d'entrées à trigger de Schmitt afin de limiter la sensibilité au bruit
* Diminution du temps entre une condition de stop et une condition de départ à 1,3 µs
* Mise en haute impédance d'un circuit non alimenté afin d'éviter de bloquer le bus si un circuit n'est pas alimenté

Très récemment, Philips a introduit une nouvelle extension de la norme I²C qui étend la vitesse à **3,4 Mbits/s**

2.2.8) Les adresses réservées

Les adresses 0000 0xxx ne sont pas utilisées pour l'adressage de composants. Elles ont été réservés par Philips pour effectuer certaines fonctions spéciales.

2.2.8.1) Adresse d'appel général : 00000000

Après l'émission d'un appel général, les circuits ayant la capacité de traiter ce genre de demande d'appel émettent un acquittement.
Le deuxième octet permet de définir le contenu de l'appel comme nous allons le voir:

* Reset

Deuxième octet : 00000100
Remet tous les registres de circuits connectés dans leur état initial (mise sous tension). Les circuits qui le peuvent rechargent leur adresse esclave.
Les circuits définissant leur adresse de façon matérielle réinitialisent leur adresse esclave. Cela ne réinitialise pas les circuits.

* Interdit

Deuxième octet : 00000000

* Interruption

Deuxième octet : xxxx xxx1
Cette commande joue le rôle d'interruption. xxxx xxx peut être l'adresse du circuit qui a généré l'interruption.

Les autres valeurs du second octet ne sont pas définies et sont tout simplement ignorées.

2.2.8.2) Octet de start : 00000001

Cet octet est utilisé pour synchroniser les périphériques lents avec les périphériques rapides.

2.2.8.3) Début d'adressage CBUS : 0000 001x

L'émission de cet octet permet de rendre sourd tous les circuits I²C présents sur le le bus. À partir de ce moment, on peut transmettre ce que l'on désire sur le bus, en utilisant par exemple un autre protocole. Le bus repasse en mode normal lors de la réception d'une condition d'arrêt.

2.2.8.4) Autre : 00000110 à 00001111

Ces adresses ne sont pas définies et sont ignorées par les circuits I²C. Elles peuvent être utilisées pour débuggeur un réseau multi master.

3) QUELQUES COMPOSANTS I²C

3.1) Le PCF8574

Le PCF8574 est un port d'entrées/sorties 8 bits quasi-bidirectionnel pour le bus I²C.
Il est à noter qu'il existe deux versions de ce circuit intégré, le PCF8574 et le PCF8574A qui se distinguent uniquement par la partie fixe de l'adresse I²C. La partie variable étant composée de 3 bits, cela fait donc 16 adresses différentes, ce qui représente 128 entrées/sorties.

3.2) Le PCF8583

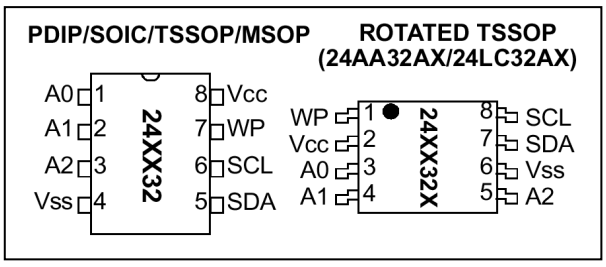
Le PCF8583 est une horloge avec calendrier combinée à une RAM de 240 octets. Les années sont gérées sur 4 ans, ce qui permet de tenir compte des années bisextiles. Il possède aussi une fonction alarme avec une sortie d'interruption. L'horloge de référence peut être soit un quartz d'horloger cadencé à 32,768 kHz soit une entrée TTL cadencée à 50 Hz (pour la synchronisation sur le secteur).

3.3) Le PCF8591

Le PCF8591 est un quadruple convertisseur analogique/numérique 8 bits combiné avec un convertisseur numérique analogique 8 bits. Les convertisseurs analogique/numérique sont de type approximation successives. La référence de tension est externe (comprise entre Vss et Vdd), et la fréquence d'échantillonnage est controlé par l'horloge du bus I²C. Les entrées sont configurables logiciellement, soit en entrées simple, soit en entrées différentielles.

3.4) Le 24LC32A/P

Le 24LC32A/P est une mémoire série I²C.
C'est une mémoire EEPROM de capacité 32kbits
32 octets max à la suite en lecture ou écriture (128 pages)



3.5) Adresses des principaux circuits

	Bits 3 à 0							
bits 7-4	**0 0 0 x**	**0 0 1 x**	**0 1 0 x**	**0 1 1 x**	**1 0 0 x**	**1 0 1 x**	**1 1 0 x**	**1 1 1 x**
0 0 0 0								
0 0 0 1								
0 0 1 0	SAA5240 SAA4700 SAF1134 SAF1135	SAA5240 SAA5241 SAA5243 SAA5244 SAA5246 SAA9041 SAA4700 SAF1134 SAA1135	SAA5240 SAB9070 SAF1134 SAA5244	SAF1134 SAF1135 SAF1135	SAA5252 SAA9020	SAA9020	SAA9020	SAA9020
0 0 1 1	SAA7250 PCB5020 PCB5021 PCB5032	SAA7250 PCB5020 PCB5021 PCB5032			SAA1136 PCF1810	PCF1810	PCF1810 SAA1770	PCF1810 SAA1770
0 1 0 0	SAA1137 PCD4430 SAA7194 PCF8574 TDA8444	SAA1137 PCD4430 SAA7194 PCF8574 TDA8444	PCA1070 PCF8574 TDA8444	PCA1070 PCF8574 TDA8444	PCD3311 PCD3312 PCF8574 TDA8444	PCD3311 PCD3312 PCF8574 TDA8444	SAB3028 PCF8574 TDA8444	PCD5002 PCF8574 TDA8444
0 1 0 1								
0 1 1 0								
0 1 1 1	PCF8576 SAA1064 PCF8574A	PCF8576 SAA1064 PCF8574A	PCF8577 SAA1064 PCF8574A	PCF8577A SAA1064 PCF8574A	PCF8578 PCF8579 PCF8574A	PCG8566 PCF8574A	PCG8566 PCF8574A	
1 0 0 0	TEA6320 TEA6330 TDA8420 TDA8421 TDA8960 NE5751	TDA8424 TDA8425 TDA8426 TDA8420 TDA8421 TDA9860 NE5751	TDA8425 TDA8415 TDA8416 TDA8440 TDA8940 TDA8417 TDA6360 TDA8480	TDA6360 TDA8480				
1 0 0 1	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591	TDA8440 TDA8540 PCF8591
1 0 1 0	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582 PCF8583	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582 PCF8583	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582	PCF8570 PCF8571 PCF8580 PCF8581 PCF8582
1 0 1 1	SAA7199 PCF8570	SAA7199 SAA7152 PCF8570	TDA8416 PCF8570	PCF8570	TDA2518 SAA7186 PCF8570	PCA8510 PCA8516 PCF8570	SAA7186 PCF8570	SAA9065 PCF8570
1 1 0 0	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010 TEA6000 TEA6100	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010 TSA6057 PCA8516 TSA6060 TSA6061	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010 TSA6057 PCA8516 TSA6060 TSA6061	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010	TSA5510 TSA5511 TSA5512 TSA5514 TSA5519 SAB3035 SAB3036 SAB3037 UMA1009 UMA1010
1 1 0 1	TDA8443 PCF8573	TDA8443 PCF8573	TDA8443 PCF8573	TDA8443 PCF8573	TDA8443 UMA1000 TDA1551	TDA8443 UMA1000	TDA8443 UMA1000 PCD4440	TDA8443 UMA1000 PCD4440
1 1 1 0	SAA7192	SAA7192						
1 1 1 1								

Pour en savoir plus sur le bus I²C :
Vous pouvez consulter le document *The I²C-bus and how to use it (including specifications)* de Philips Semiconductors.