# Présentation du bus I2C

L'I2C (Inter-Integrated Circuit), fait partie des bus séries : 2 fils (plus la masse) pour tout faire passer.

A l'origine, au début des années 80, Philips, son concepteur, l'avait créé pour minimiser les liaisons entres les circuits intégrés numériques de ses produits(Téléviseurs, éléments HiFi, magnétoscopes, voire même certains oscillos...), la quantité de donnée à faire circuler était faible d'où ses premières caractéristiques :

- adressage des circuits sur 7 bits
- vitesse plafonnée à 100 kbit/s
- compatibilité avec le CBUS (ancêtre de l'I2C chez Philips)

Depuis, il a fait son petit bonhomme de chemin et de nombreux fabricants l'intègrent dans leurs composants et appareils.

Voici quelques types de circuits disponibles sur le marché :

- Expandeur de bus (entrée/sortie 8 bits)
- Convertisseur A/N et N/A
- Récepteur infra-rouge (télécommande RC5)
- Capteur de température
- Horloges temps réel
- Chargeur de batterie Ni-Cd
- Décodeur télétexte
- PLL pour tuner HF

...

Face à l'explosion du nombre de circuits I2C disponibles et au trafic en très forte augmentation, Philips a publié en 1992 les nouvelles spécifications de l'I2C :

- Compatibilité totale avec l'ancien I2C mais abandon de la compatibilité CBUS
- Adressage étendu sur 10 bits
- Vitesse montée à 400 kbit/s (dans le respect des normes CEM, s'il vous plait)

La communication sur le bus passe par plusieurs phases :

- Le maître du bus (le maître du moment, s'il s'agit d'un bus multi-maîtres) surveille l'état du bus pour déterminer s'il peut engager la conversation avec son ou ses esclaves.
- Si le bus est libre (c'est à dire dans un état bien précis), le maître émet une condition de départ suivie de l'adresse de l'esclave.
- L'esclave cible (target slave) répond par un acquittement (bit d'acquittement, ou ACK) après chaque "mot" (chaque octet) envoyé par le maître.
- Et enfin le maître du bus met fin à la discussion par une condition de stop.

# Glossaire I2C

**I2C** = Acronyme de Inter Integrated Circuit.

**SDA** = Serial DAta, c'est la ligne de données bidirectionnelle sur laquelle sont vehiculées les données (0 ou 1), par paquets de 8 bits. De 0 à 1,5 volts, la ligne est considerée a l'état bas et de 3 à 5 volts, la ligne est considérée à l'état haut.

**SCL** = Serial CLock, c'est la ligne d'horloge sur laquelle sont transmises les impulsions qui conditionnent la validité des bits transmis au même moment sur la ligne SDA. En effet, les bits valides (sur la ligne SDA donc) doivent avoir une durée au moins aussi longue que la durée de l'impulsion SCL à l'état haut. Les changements de valeurs de SDA se faisant quand SCL est à l'état bas.

**Maître** = Circuit qui initialise la communication sur le bus, et par conséquent qui la termine. En général c'est un micro-contrôleur ou un PC. Notons qu'une particularité du maître, est qu'il n'a pas d'adresse.

**Esclave** = Circuit qui exécute les ordres du ou des maîtres. Notez qu'il est possible sur le bus I2C d'avoir des composants (en général des micro-contrôleurs) qui peuvent-être tour à tour des maîtres ou des esclaves. L'esclave ne prend jamais l'initiative d'entamer la conversation avec d'autres éléments, il se contente d'obéir ou de répondre aux demandes du maître.

**ACK** = Acquittement (acknowledge). C'est un bit 0 émis par le destinataire d'un ordre ou d'une donnée.

**NACK** = Non acquittement! C'est un bit 1 émis par le maître avant la condition de stop.

# L'adressage des composants esclaves I2C

L'adressage des composants I2C se fait sur 7 bits, les 7 bits de poids fort d'un octet. (l'adressage 10 bits se fait sur deux octets, les 2 bits supplémentaires étant les bits de poids faible de l'octet de poids fort)

Dans le data-sheet du composant vous trouverez, entre autres le brochage et les caractéristiques, mais également la partie fixe de l'adresse, imposée par le(s) fabricant(s).
Imposée? Afin d'homogénéiser "l'offre produit" et se protéger de l'anarchie des différents fabricants (NEC, INTEL, TOSHIBA, HITACHI, MOTOROLA, FUJITSU, TEXAS INSTRUMENTS, MITSUBISHI, NATIONAL SEMICONDUCTOR, MATSUSHITA, PHILIPS, SAMSUNG, AMD, SGS, SHARP et tant d'autres), un "I2C Committee" veille à établir une liste d'adresses figées en fonction des types de composants. Heureusement pour nous, les 7 bits ne sont pas tous figés (à quelques exceptions près).

Systématiquement, les quatre bits de poids fort sont réservés, bit 4 à bit 7. Reste donc 3 bits pour déterminer une adresse unique sur le bus. Il est donc en théorie possible de placer huit composants de référence identique sur un même bus. Hélas, quelques composants débordent sur ces trois bits disponible et là, vous en mettrez donc moins...
La partie de l'adressage qui vous incombe se réduira à mettre les pattes du composant désignées par A2 (bit 3), A1 (bit 2), et A0 (bit 1) à la masse pour passer ce bit à 0, ou mettre cette patte au + 5 volts pour passer ce bit a 1.
Enfin, le bit 0 n'a pas de rapport direct avec l'adresse mais il conditionne le sens du dialogue de l'octet suivant,l'octet de données.

Voici une ancienne liste non exhaustive de différents circuits I2C :

| Nom | Fonction | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|
| ... | Adresse de communication générale | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | Adresses Réservées | 0 | 0 | 0 | 0 | X | X | X |
| ... | Octet de poids fort en adressage 10 bits | 1 | 1 | 1 | 1 | 1 | A9 | A8 |
| PCD3311 | Générateur de tonalité (DTMF,modem,musique) | 0 | 1 | 0 | 0 | 1 | 0 | A0 |
| PCD3312 | Générateur de tonalité (DTMF,modem,musique) | 0 | 1 | 0 | 0 | 1 | 0 | A0 |
| PCF8200 | Synthétiseur de paroles | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| PCF8566 | Commande d'affichage LCD universelle | 0 | 1 | 1 | 1 | 1 | 1 | A0 |
| PCF8570 | RAM statique (256x8) | 1 | 0 | 1 | 0 | A2 | A1 | A0 |
| PCF8571 | RAM statique (128x8) | 1 | 0 | 1 | 1 | A2 | A1 | A0 |
| PCF8572 | EEPROM (128x8) | 1 | 0 | 1 | 0 | A2 | A1 | A0 |
| PCF8573 | Horloge temps réel / Calendrier | 1 | 1 | 0 | 1 | 0 | A1 | A0 |
| PCF8574 | Port d'Entrées/Sorties 8bits | 0 | 1 | 0 | 0 | A2 | A1 | A0 |
| PCF8574A | Port d'Entrées/Sorties 8bits | 0 | 1 | 1 | 1 | A2 | A1 | A0 |
| PCF8576 | Commande d'affichage LCD universelle | 0 | 1 | 1 | 1 | 0 | 0 | A0 |
| PCF8577 | Commande d'affichage LCD à 64 segments | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| PCF8577A | Commande d'affichage LCD à 64 segments | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| PCF8578/79 | Commande d'affichage matricielle LCD | 0 | 1 | 1 | 1 | 1 | 0 | A0 |
| PCF8582A | EEPROM (256x8) | 1 | 0 | 1 | 0 | A2 | A1 | A0 |
| PCF8583 | Horloge / Calendrier avec RAM statique | 1 | 0 | 1 | 0 | 0 | 0 | A0 |

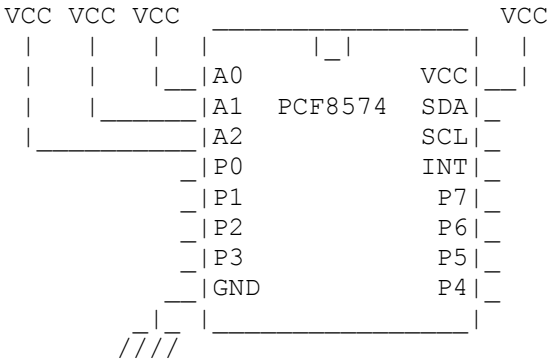
| Nom | Fonction | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|
| PCF8591 | Convertisseur A/N et N/A 8 bits | 1 | 0 | 0 | 1 | A2 | A1 | A0 |
| SAA1064 | Commande de LED à 4 chiffres | 0 | 1 | 1 | 1 | 0 | A1 | A0 |
| SAA1136 | Interface audio PCM | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| SAA1300 | Circuit de commutation pour récepteur | 0 | 1 | 0 | 0 | 0 | A1 | A0 |
| SAA3028 | Transcodeur IR (RC-5) | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| SAA4700 | Processeur de ligne de donnée VPS | 0 | 0 | 1 | 0 | 0 | 0 | A0 |
| SAA5243/45 | Décodeur vidéo texte | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| SAA9020 | Contrôleur de mémoire | 0 | 0 | 1 | 0 | 1 | A1 | A0 |
| SAA9050/51 | Décodeur télévision numérique multistandard | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| SAA9055P/8A | Décodeur télévision numérique SECAM | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| SAA9055P/8E | Décodeur télévision numérique SECAM | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| SAA9062/63/64 | Commande de déviation | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| SAA9068 | Commande d'incrustation d'image dans l'image | 0 | 0 | 1 | 0 | 0 | 1 | A0 |
| SAB3035/36/37 | Syntonisateur TV | 1 | 1 | 0 | 0 | 0 | A1 | A0 |
| SAF1135 | Décodeur de ligne de données | 0 | 0 | 1 | 0 | 0 | A1 | A0 |
| ST24C02 | EEPROM (256x8) | . | . | . | . | A2 | A1 | A0 |
| ST24C04 | EEPROM (512x8) | . | . | . | . | A2 | A1 | A0 |
| ST24C08 | EEPROM (1024x8) | . | . | . | . | A2 | A1 | A0 |
| TDA8370 | Processeur de synchronisation pour TV | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| TDA8400 | Interface pour pré diviseur/synthétiseur | 1 | 1 | 0 | 0 | 0 | A1 | A0 |
| TDA8405 | Processeur de son stéréo pour TV | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| TDA8420/21 | Processeur audio stéréo | 1 | 0 | 0 | 0 | 0 | 0 | A0 |
| TDA8425 | Processeur audio stéréo | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| TDA8440 | Commutateur audio/vidéo | 1 | 0 | 0 | 1 | A2 | A1 | A0 |
| TDA8442 | Interface pour décodeur de couleurs | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| TDA8443A | Interface YUV/RGB | 1 | 1 | 0 | 1 | A2 | A1 | A0 |
| TDA8444 | Octuple convertisseur N/A 6bits | 0 | 1 | 0 | 0 | A2 | A1 | A0 |
| TDA8461 | Décodeur PAL/NTSC | 1 | 0 | 0 | 0 | 1 | 0 | A0 |
| TEA6000 | Syntonisateur FM/FI | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| TEA6300T/10T | Commande de volume/tonalité | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| TEA6330 | Ampli de régulation | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| TEA6360 | Égaliseur | 1 | 0 | 0 | 0 | 0 | 1 | A0 |
| TSA5510(T) | Synthétiseur de fréquence 1,3 GHz | 1 | 1 | 0 | 0 | 0 | A1 | A0 |
| TSA6057(T) | Synthétiseur de fréquence à PLL | 1 | 1 | 0 | 0 | 0 | 1 | A0 |
| UMA1000T | Processeur pour téléphone sans fil | 1 | 1 | 0 | 1 | 1 | A1 | A0 |
| UMA1010T | Synthétiseur universel pour télécommunication | 1 | 1 | 0 | 0 | 0 | 0 | A0 |

# L'adressage des composants esclaves I2C (suite)

A noter, il existe un composant I2C très utilisé qui se décline avec deux adresses fixes, il s'agit du "PCF8574" (c'est un registre à huit ports numériques bidirectionnels, huit entrées/sorties digitales. Les quatre bits de poids fort de la version de base sont positionnées à "0100", soit 64 en décimal et la version "sup", le "PCF8574A" s'appelle "0111" soit 112 en décimal.

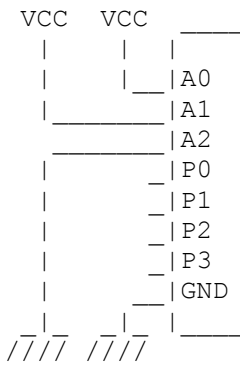
Grâce à cette disposition, le nombre de PCF8574 disponibles sur le bus passe de 8 à 16 composants à huit E/S chacun (soit un total de 128 E/S !!!). Si cela se révélait insuffisant, vous pouvez augmenter le nombre de bus I2C.

Exemple d'adressage d'un PCF8574:




Ce composant répondra à l'adresse :
pour la partie fixe : 0100 (64)
pour la partie définie içi : 111X (14+X)
soit en lecture : X = 1
0100 111 1 (64+14+1 = 79)
et en écriture : X = 0
0100 1110 (64+14+0 = 78)

Exemple d'adressage d'un PCF8574 "A":



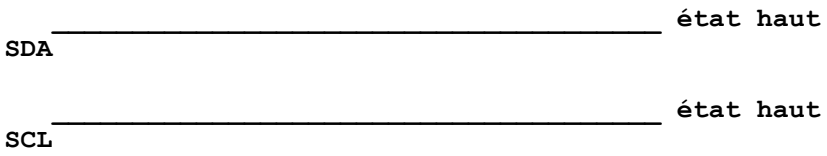

Ce composant répondra à l'adresse :
pour la partie fixe = 0111 (112)
pour la partie définie içi : 011X (6+X)
soit en lecture : X = 1
0111 011 1 (112+6+1 = 119)
et en écriture : X = 0
0111 011 0 (112+6+0 = 118)

# Description du protocole I2C

## L'état de repos du bus I2C

Au repos, c'est a dire quand aucune communication ne circule sur le bus, le bus est dit "au repos". Électriquement, les deux lignes du bus, SDA et SCL sont placées à l'état haut par chaque élément du bus, au +5 volts donc. Il faut savoir néanmoins que l'état haut est valide dès 3 volts, l'état bas étant situé entre 0 et 1,5 volts.

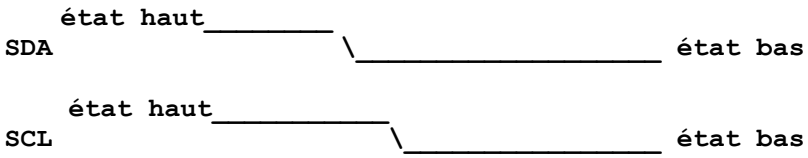
Bus à l'état de repos:




## La condition de départ du bus

La condition de départ (et non pas bit de départ!), ne peut avoir lieu que si le bus est au repos, ou libre (free bus). Le maître du bus surveille donc l'état du bus et commence par placer la ligne SDA (Serial Data, la ligne de données) à l'état bas (au 0 volt) et juste après c'est au tour de la ligne SCL (Serial CLock, la ligne d'horloge) de passer à l'état bas.
A ce moment, le bus est dit "occupé" (busy) et aucun autre maître de bus ne prendra la main. Les autres maîtres attendront la prochaine condition d'arrêt, où le bus sera de nouveau libre.

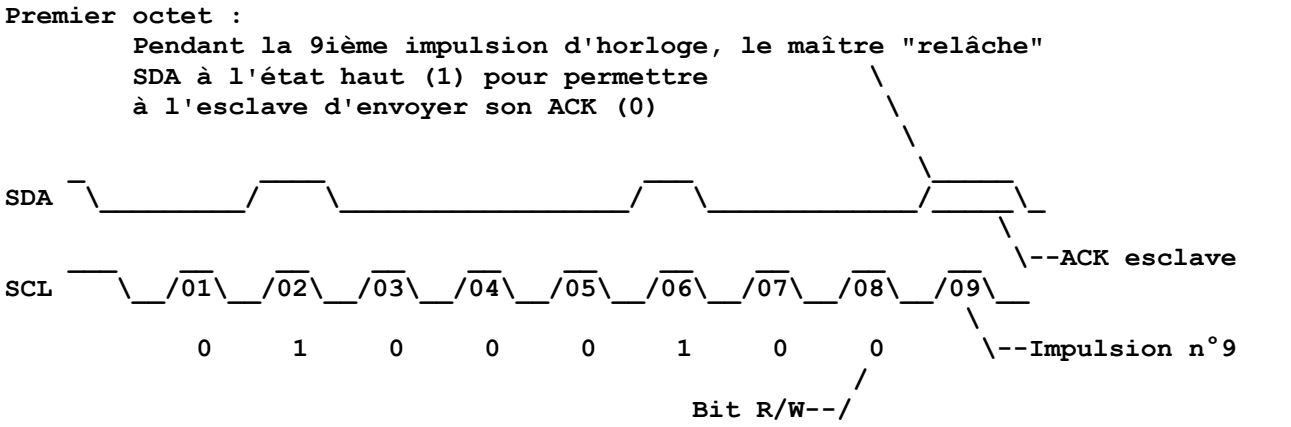
Condition de départ du bus :

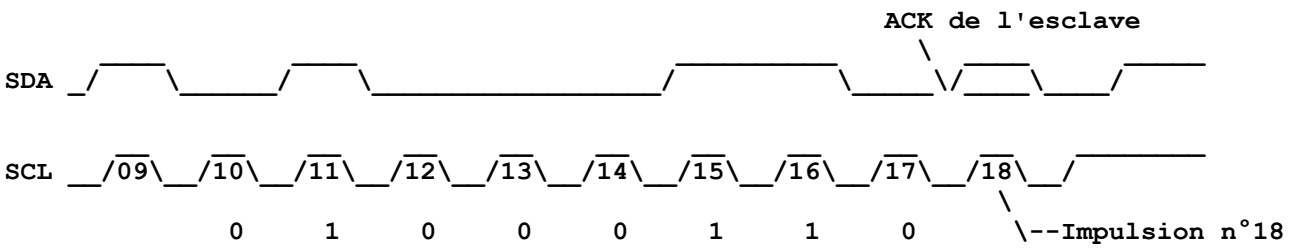


# Description du protocole I2C (suite)

**L'écriture (du maître vers un esclave)**
Après avoir émis la condition de départ, le maître va "s'adresser" à son esclave en envoyant son adresse (composée de la partie fixe, de la partie définissable et du bit de direction (bit R/W, lecture=1 ou écriture=0)). Pendant le même temps, le maître envoie sur le bus les impulsions d'horloge sur la ligne SCL. Un bit de donnée "valide" (0 ou 1) doit être transmis pendant l'etat haut de SCL.

Exemple:
Ecriture de valeur [0100 0110] (70 en décimal) dans un circuit PCF8574 adressé de la manière suivante [0100] pour l'adresse fixe, A2, A0 à la masse et A1 au VCC [010] et le bit R/W à 0 [0] (écriture), ce qui donne [0100 0100] (68 en décimal):




Le premier octet envoyé est l'adresse. Sa partie fixe (déterminée par le fabricant, en accord avec le "I2C Committee") est contenue dans le quartet de poids fort (bits 4 à 7), la partie "libre" (configurable par l'utilisateur) se trouve dans les bits 1 à 3. Et enfin le bit de poids faible, le bit 0 contient la direction de la communication de l'octet suivant (1=lecture, 0=écriture).
Le bit de poids fort est toujours envoyé le premier. Comme cela, le dernier bit
transmis conditionne le sens du prochain octet (écriture d'un ordre du maître vers l'esclave ou transmission de données de l'esclave vers le maître).
Quand l'esclave à qui s'adresse le maître à réceptionné le bit de poids faible (le dernier reçu), il émet un bit d'acquittement (ACK). C'est un bit "0" :
l'esclave met SDA à l'état bas pendant que la ligne SDA est relâchée par le maître à l'état haut pendant la neuvième impulsion d'horloge sur la ligne SCL (qui répétons le, est générée par le maître). Le maître du bus peut donc envoyer
son ordre (deuxième octet), ou recevoir des données de l'esclave. Une fois le dernier bit de données reçu, un bit d'acquittement est émis et le maître peut ainsi générer une condition de stop pour libérer le bus.

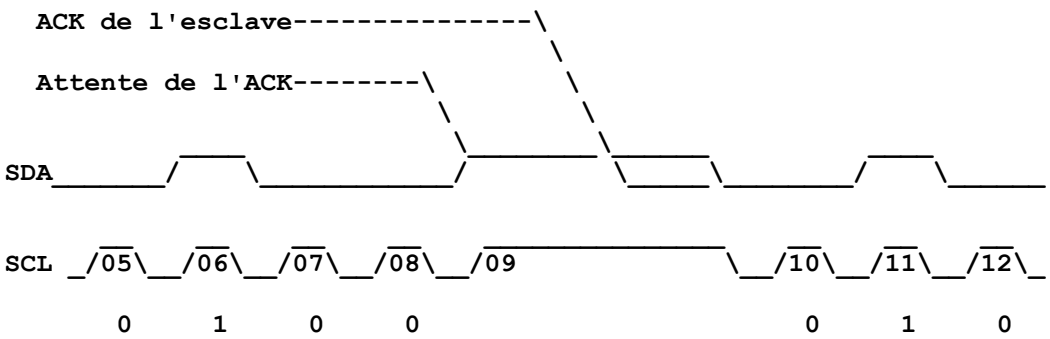
# Description du protocole I2C (suite)

**L'acquittement : ACK (Acknowledge)**

On peut comparer l'acquittement à un accusé de réception de courrier : c'est le destinataire qui renvoie à l'expéditeur l'information selon laquelle le message a bien été reçu (et implicitement qu'il est en mesure d'exécuter l'ordre).
En clair, pour prendre un exemple au hasard, quand le maître demande à l'esclave une info, deux octets circulerons sur le bus, il y aura donc deux acquittements car pour chaque octet transmis, un acquittement doit avoir lieu.
Le premier octet (du maître vers l'esclave) sera donc acquitté par l'esclave. Mais comme le bit de poids faible (de l'octet d'adresse) indique un ordre de lecture, le deuxième octet sera envoyé au maître et ce sera a lui d'acquitter l'octet réceptionné... Mais être maître donnant des privilèges, le maître n'acquitte rien du tout!
Voyons maintenant l'état électrique du bus, qui demande une certaine attention : lors de la condition de départ, et à partir de la première impulsion d'horloge, le bit d'acquittement aura toujours lieu lors d'une impulsion d'horloge multiple de neuf (9ième, 18ième ...). A ce moment précis, l'expéditeur du message (celui qui attend l'acquittement) doit relâcher SDA (à l'état haut donc) a**ussi longtemps** que l'esclave (qui peut-être plus lent à traiter les données que le maître à les envoyer) n'a pas placé SDA à l'état bas.

Quand le maître "voit" SDA à l'état bas, il peut générer une condition de stop ou continuer à garder la main...

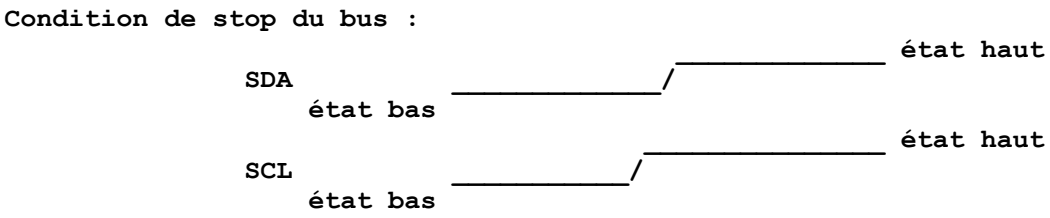
Acquittement par l'esclave :




Dans l'exemple ci-dessus, l'esclave a besoin de rallonger le signal pour acquitter le premier octet.

**La condition de stop du bus**

La condition de stop (ou d'arrêt) est très similaire à la condition de départ, mais dans l'autre sens. Dans la condition de départ, c'est SDA qui passe à l'état bas en 1er, ici SDA repasse à l'etat haut en dernier... Après le dernier acquitement (ACK ou NACK), le bus est de nouveau libéré par le maître.



```
*****************************************
* Exemple de programme écrit en structurée *
*****************************************
```

Nous allons écrire un programme de démonstration d'écriture, puis de lecture dans un circuit PCF8574 (8 portes entrées/sorties).

Pour cela, nous allons utiliser des routines générales qui nous permettrons de gérer le bus I2C.

Pour effectuer cette gestion, nous avons besoin des routines suivantes :

- I2CSTART : Mise du bus I2C dans la condition "START"
- I2CSTOP : Mise du bus I2C dans la condition "STOP"
- I2CTX : Emission d'un octet sur le bus I2C
- I2CRX : Réception d'un octet sur le bus I2C
- ACKTX : Emission d'un "ACK" sur le bus I2C
- ACKRX : Réception d'un "ACK" sur le bus I2C

```
BEGIN

' Définition et initialisations des variables
DECLARE i2caddr   AS byte     'Mot de l'adresse du composant I2C
DECLARE i2cdata   AS byte     'Octet de valeur lue ou écrite sur le bus I2C
DECLARE i2cack    AS bit      'Bit "ACK" du bus I2C
DECLARE i2csda    AS nibble   'N° de pin I2C data
DECLARE i2cscl    AS nibble   'N° de Pin I2C clock
DECLARE i2cad1WR  AS byte     'Adresse d'écriture du composant I2C (PCF8574)
DECLARE i2cad1RD  AS byte     'Adresse de lecture du composant I2C (PCF8574)
DECLARE i2cio     AS byte     'Valeur lue ou écrite dans le composant I2C
DECLARE i         AS nibble   'Variable de comptage
DECLARE j         AS bit      'bit à écrire ou a lire sur le bus I2C
LET i2cad1wr = %01000000
LET i2cad1rd = %01000001
LET i2cio = %01010101

'Exemple d'écriture dans le PCF8574
'La valeur contenue dans i2cio est envoyée dans le PCF8574
GOSUB i2cstart                'Mise du bus en condition "START"
LET i2cdata = i2cad1WR        'valeur de l'adresse du PCF8574
GOSUB i2ctx                   'Ecriture de l'adresse I2C
GOSUB ackrx                   'Attente de l'ACK
LET i2cdata = i2cio           'valeur de la donnée à écrire
GOSUB i2ctx                   'Ecriture de la donnée I2C
GOSUB ackrx                   'Attente de l'ACK
GOSUB i2cstop                 'Mise du bus en condition "STOP"

'Exemple de lecture du PCF8574
'La valeur lue dans le PCF8574 est stockée dans i2cio
GOSUB i2cstart                'Mise du bus en condition "START"
LET i2cdata = i2cad1RD        'valeur de l'adresse du PCF8574
GOSUB i2ctx                   'Ecriture de l'adresse I2C
GOSUB ackrx                   'Attente de l'ACK
GOSUB i2crx                   'Lecture de la donnée I2C
LET i2cio = i2cdata           'Stockage de la valeur dans i2cio
GOSUB i2cstop                 'Mise du bus en condition "STOP"

Affichage de la valeur de i2cio

END
```

```
**************************************************
* Exemple de programme écrit en structurée (suite) *
**************************************************

'Définitions des différentes subroutines



' I2CSTART : Mise du bus I2C dans la condition "START"
SUB I2CSTART
     Mise à 1 de la pin définie par i2csda   'I2C data line
     Mise à 1 de la pin définie par i2cscl   'I2C clock line
     Mise à 0 de la pin définie par i2csda   'I2C data line
ENDSUB



' I2CSTOP  : Mise du bus I2C dans la condition "STOP"
SUB I2CSTOP
     Mise à 0 de la pin définie par i2csda   'I2C data line
     Mise à 1 de la pin définie par i2cscl   'I2C clock line
     Mise à 1 de la pin définie par i2csda   'I2C data line
ENDSUB



' I2CTX    : Emission d'un octet sur le bus I2C
SUB I2CTX
     LET i = 0
     DO WHILE i < 8
          LET j = (i)ème bit de i2cdata
          Mise à la valeur j de la pin définie par i2csda
          Mise à 1 de la pin définie par i2cscl
          PAUSE 1ms
          Mise à 0 de la pin définie par i2cscl
          Mise à 0 de la pin définie par i2csda
     ENDDO
ENDSUB



' I2CRX    : Réception d'un octet sur le bus I2C
SUB I2CRX
     LET i = 0
     DO WHILE i < 8
          Mise à 1 de la pin définie par i2cscl
          LET j = valeur lue sur la pin définie par i2csda
          LET (i)ème bit de i2cdata = j
          PAUSE 1ms
          Mise à 0 de la pin définie par i2cscl
     ENDDO
ENDSUB
```

```
**************************************************
* Exemple de programme écrit en structurée (suite) *
**************************************************

'Définitions des différentes subroutines (suite)



' ACKTX    : Emission d'un "ACK" sur le bus I2C
SUB ACKTX
     Mise à 1 de la pin définie par i2cscl
     Mise "haute-impédance" de la pin définie par i2csda
     PAUSE 1ms
     Mise à 0 de la pin définie par i2csda
     PAUSE 1ms
     Mise à 0 de la pin définie par i2cscl
ENDSUB



' ACKRX    : Réception d'un "ACK" sur le bus I2C
SUB ACKRX
     Mise à 1 de la pin définie par i2cscl
     Mise "haute-impédance" de la pin définie par i2csda
     PAUSE 1ms
     Attente d'un 0 sur la pin définie par i2csda
     PAUSE 1ms
     Mise à 0 de la pin définie par i2cscl
ENDSUB
```

```
'****************************************************************
'* Exemple de programme Basic Stamp2 utilisant les routines I2C *
'****************************************************************
'
'
'Nous avons choisi un exemple dans lequel un PCF8574 (8 entrées/sorties) est
'connecté sur le bus I2C. La connexion SDA (Serial DAta) et la connexion SCL
'(Serial CLock) sont respectivement P12 et P13 du STAMP2.
'Nous allons y faire une écriture, et ensuite une lecture.
'
'
'Définition des variables utilisées

i2caddr    var  byte         'Mot de l'adresse du composant I2C
i2cdata    var  byte         'Octet de valeur lue ou écrite sur le bus I2C
i2cack     var  bit          'Bit "ACK" du bus I2C
i2csda     con  12           'Pin I2C data du Stamp2
i2cscl     con  13           'Pin I2C clock du Stamp2
i2cad1WR   con  %01000000    'Adresse d'écriture du composant I2C (PCF8574)
i2cad1RD   con  %01000001    'Adresse de lecture du composant I2C (PCF8574)
i2cio      var  byte         'Valeur lue ou écrite dans le PCF8574

'initialisation de i2cio
i2cio = 85

'Ecriture de i2cio dans le PCF8574
     GOSUB i2cstart          'Mise du bus en condition "START"
     i2data=i2cad1WR         'Valeur à envoyer = adresse d'écriture du PCF8574
     GOSUB i2ctx             'Envoi de i2cdata sur le bus i2c
     GOSUB ackrx             'Attente de l'ACK
     i2data=i2cio            'Valeur à envoyer = 85
     GOSUB i2ctx             'Envoi de i2cdata sur le bus i2c
     GOSUB ackrx             'Attente de l'ACK
     GOSUB i2cstop           'Mise du bus en condition "STOP"
'Lecture du PCF8574 et stockage de la valeur dans i2cio
     GOSUB i2cstart          'Mise du bus en condition "START"
     i2data=i2cad1RD         'Valeur à envoyer = adresse de lecture du PCF8574
     GOSUB i2ctx             'Envoi de i2cdata sur le bus i2c
     GOSUB acckrx            'Attente de l'ACK
     GOSUB i2crx             'Lecture de la valeur envoyée par le PCF8574
     i2cio=i2cdata           'Mise à jour de la variable i2cio
     GOSUB i2cstop           'Mise du bus en condition "STOP"
'Affichage de la valeur de i2cio
     DEBUG ? i2cio           'Envoi de la valeur de i2cio vers le PC

     END
```

```
'************************************************************************
'* Exemple de programme Basic Stamp2 utilisant les routines I2C (Suite) *
'************************************************************************

' Liste des routines I2C
'***********************


'I2CSTART : Mise du bus I2C dans la condition "START"
'***************************************************
i2cstart:  high i2csda   'Mise à 1 de I2C data
           high i2cscl   'Mise à 1 de I2C clock
           low  i2csda   'Mise à 0 de I2C data
return


'I2CSTOP : Mise du bus I2C dans la condition "STOP"
'**************************************************
i2cstop:   low  i2csda   'Mise à 0 de I2C data line
           high i2cscl   'Mise à 1 de I2C clock line
           high i2csda   'Mise à 1 de I2C data line
return


'I2CTX : Emission d'un octet sur le bus I2C
'******************************************
i2ctx:     shiftout i2csda,i2cscl,1,[i2cdata]
return


'I2CRX : Réception d'un octet sur le bus I2C
'******************************************
i2crx:     shiftin i2csda,i2cscl,0,[i2data]
return


'ACKTX : Emission d'un "ACK" sur le bus I2C
'******************************************
acktx:     shiftout i2csda,i2cscl,1,[%0\1]
return


'ACKRX : Réception d'un "ACK" sur le bus I2C
'******************************************
ackrx:     shiftin i2csda,i2cscl,0,[i2ack\1]
return
```