# BUS I2C

1. Quelques définitions
2. Bus I2C, description
3. Bus I2C, définitions
4. Trames de communication
5. Périphériques I2C
6. Extension du protocole
7. Problèmes réels et solutions
8. Mesures
9. Questions, discussion

# 1. Quelques définitions

Selon Wikipédia, un protocole de communication:

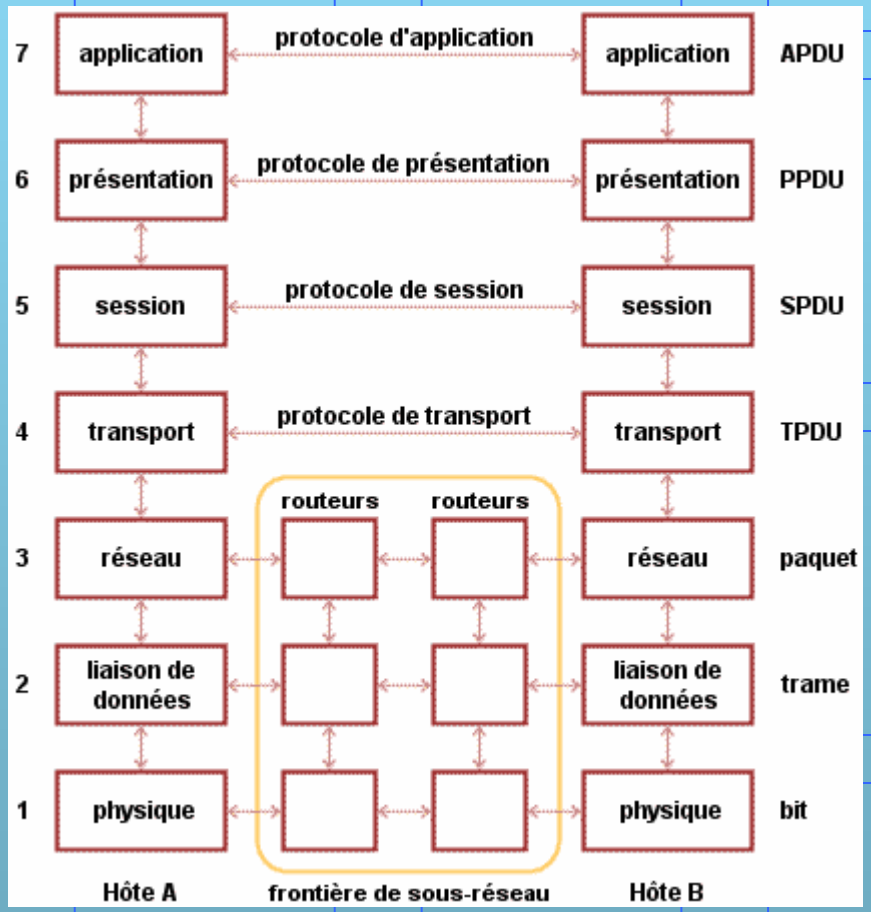
Dans les réseaux informatiques et les télécommunications, un protocole de communication est une spécification de plusieurs règles pour un type de communication particulier.

Initialement, on nommait protocole ce qui est utilisé pour communiquer sur une même couche d'abstraction entre deux machines différentes. Par extension de langage, on utilise parfois ce mot aussi aujourd'hui pour désigner les règles de communication entre deux couches sur une même machine.

# 1. Quelques définitions

## Le modèle OSI

Le protocole I2C contient quelques spécifications de la couche 4, notamment pour la gestion du multi-master



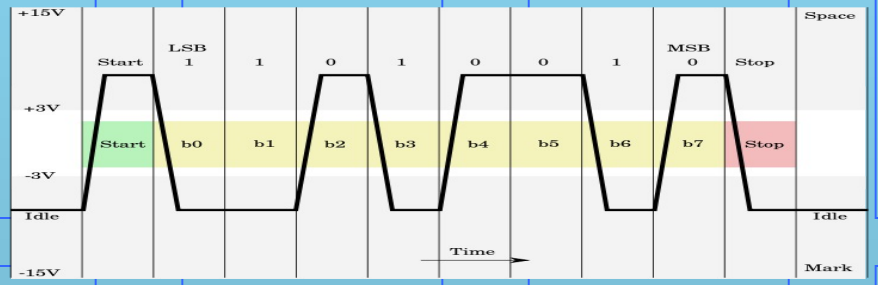

PS: cette manière de décomposer une communication est fortement critiquée, surtout pour le modèle TCP/IP

Ref: http://www.frameip.com/osi/

# 1. Quelques définitions

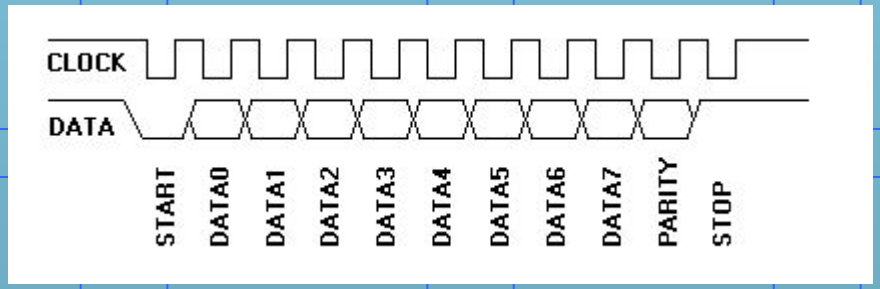
Communication asynchrone, ex RS232, USB…

RS232




Communication synchrone, ex I2C, PS/2 (souris), SPI…

PS/2



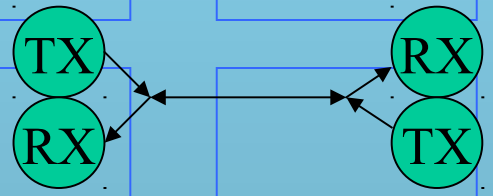

(Remarque: I2C bus = Inter-IC bus)

Ref: https://fr.wikipedia.org/wiki/RS-232

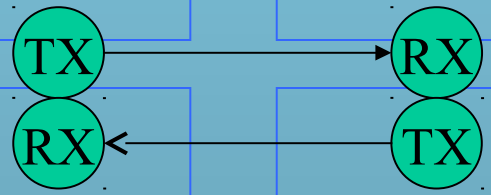
# 1. Quelques définitions

Transmission simplex


Transmission half-duplex




Transmission full-duplex

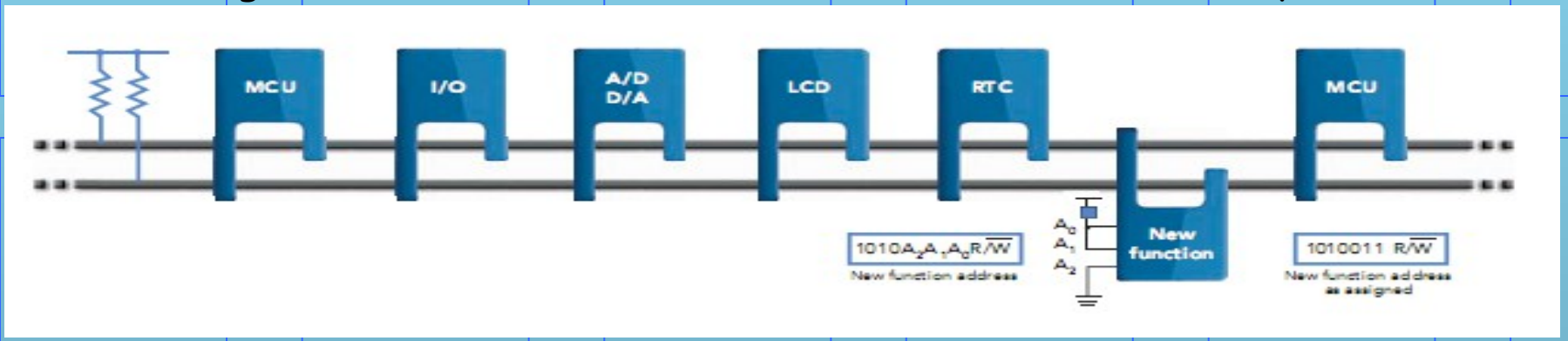


# 2. Bus I2C, description

- Bus synchrone
- Topologie réseau
- Communication half-duplex
- Protocole MASTER – SLAVE
- Gestion MULTI-MASTER
- Deux lignes de communication: SDA et SCL

NXP, anciennement Philips, est le créateur de ce protocole de communication.

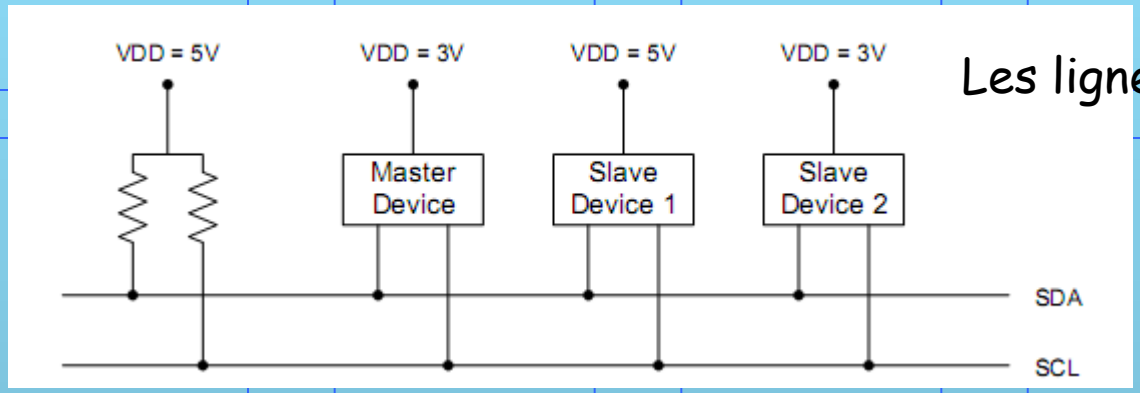
A l'origine, en 1982, ce bus était uniquement localisé sur une carte électronique.




- Le réseau (bus) est mis dans un état idle par deux pull-up.
- Adresse unique de chaque unités sur le réseau (MCU, I/O, LCD, etc).
- Communication toujours dans le sens MASTER -> SLAVE.
- La communication peut commencer que si le bus est en idle.
- Plusieurs MCU sur le réseau -> mécanisme d'arbitration.
- Si une unité reconnait sa propre adresse, elle devient SLAVE le temps du transfert.

Ref: NXP: I2C-bus solutions 2013

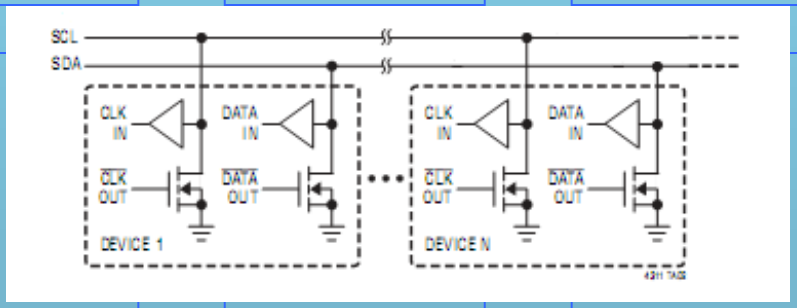
# 2. Bus I2C, description




Les lignes SDA et SCL sont bi-directionnelles.

Les deux pull-up servent à forcer SDA et SCL à 1, qui est l'état idle du bus.

La configuration des pins SDA et SCL de chaque device (unité) sur le bus peut être symbolisée ainsi:



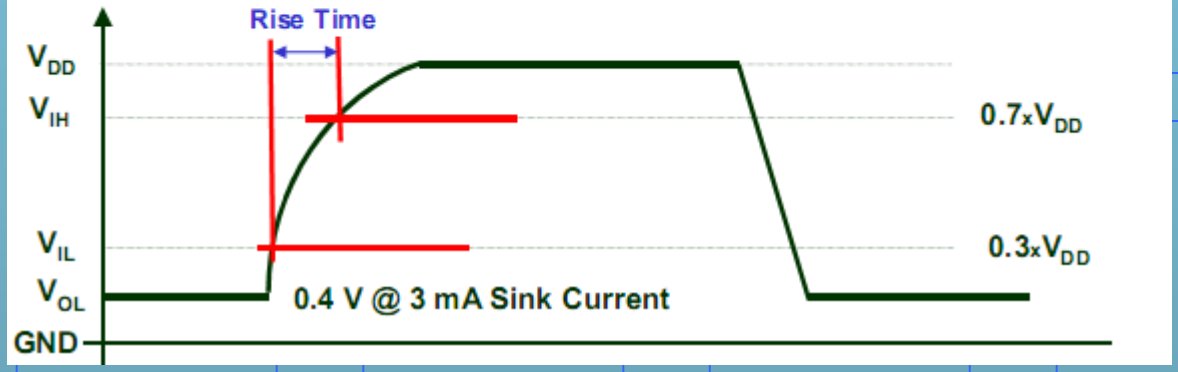

Un driver d'entrée et une sortie OC (Open Collector).

**Remarque**: pour les gens dont je fais partie, qui travaillent dans le domaine du temps réel (< 1[us]), ce bus n'apporte pas de solution simple aux demandes d'interruptions. Un SLAVE ne peut utiliser le bus pour demander un traitement prioritaire, c'est à nous d'implanter physiquement un système de gestion.

# 3. Bus I2C, définitions

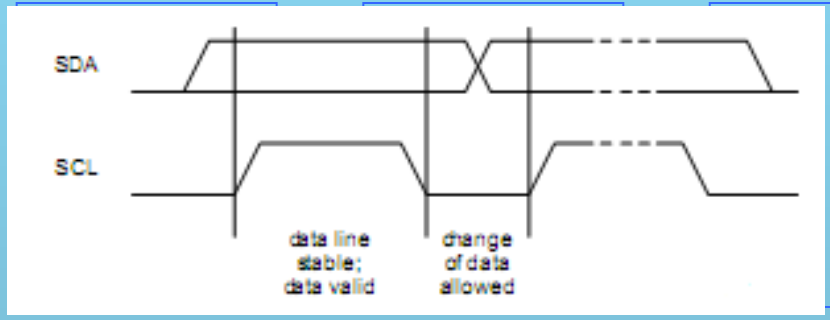
## Trois modes de fonctionnement

| | Standard-Mode | Fast-Mode | High-Speed-Mode | |
|---|---|---|---|---|
| Bit Rate (kbits/s) | 0 to 100 | 0 to 400 | 0 to 1700 | 0 to 3400 |
| Max Cap Load (pF) | 400 | 400 | 400 | 100 |
| Rise time (ns) | 1000 | 300 | 160 | 80 |
| Spike Filtered (ns) | N/A | 50 | 10 | |
| Address Bits | 7 and 10 | 7 and 10 | 7 and 10 | |



# 3. Bus I2C, définitions

SDA




Règle 1

La modification de SDA ne peut se faire que sur SCL=0.

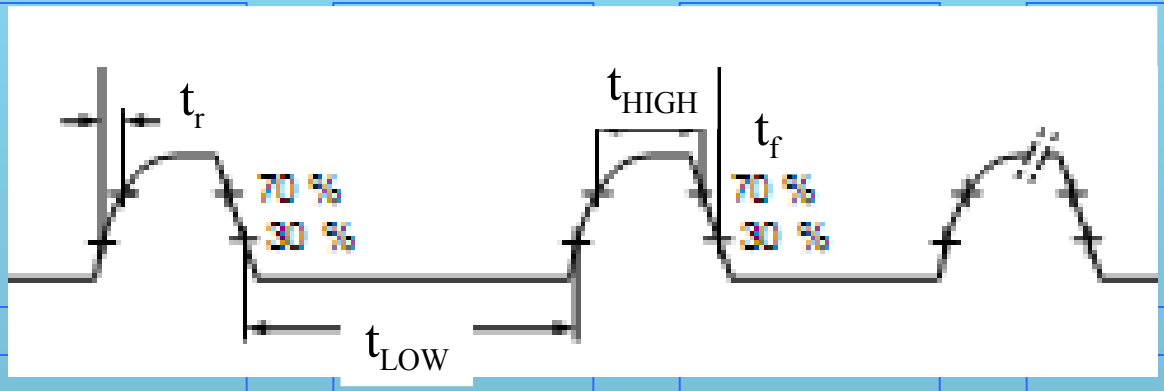
Table 6. Characteristics of the SDA and SCL bus lines for Standard, Fast, and Fast-mode Plus I²C-bus devices[1]

| Symbol | Parameter | Conditions | Standard-mode | | Fast-mode | | Fast-mode Plus | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| $t_{HD;DAT}$ | data hold time[2] | CBUS compatible masters (see Remark in Section 4.1) | 5.0 | - | - | - | - | - | µs |
| | | I²C-bus devices | 0[3] | -[4] | 0[3] | -[4] | 0 | - | µs |
| $t_{SU;DAT}$ | data set-up time | | 250 | - | 100[5] | - | 50 | - | ns |
| $t_r$ | rise time of both SDA and SCL signals | | - | 1000 | 20 + 0.1$C_b$[6] | 300 | - | 120 | ns |
| $t_f$ | fall time of both SDA and SCL signals[3][6][7][8] | | - | 300 | 20 + 0.1$C_b$[6] | 300 | - | 120[8] | ns |
| $t_{VD;DAT}$ | data valid time[10] | | - | 3.45[4] | - | 0.9[4] | - | 0.45[4] | µs |

Ref: UM10204 I2C-bus specification and user manual

# 3. Bus I2C, définitions

SCL

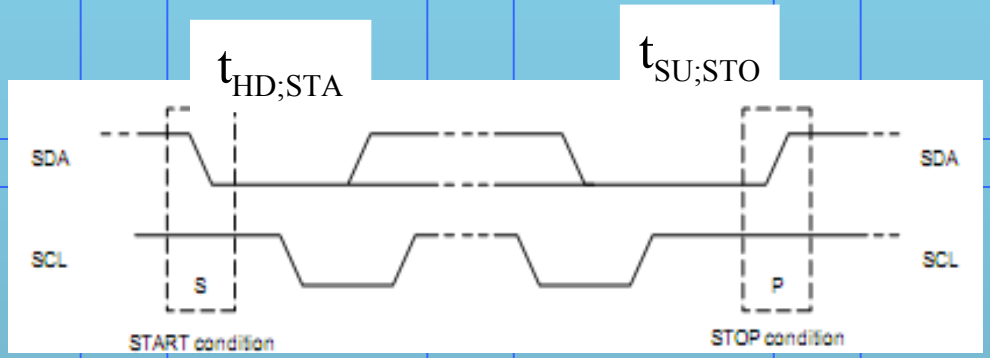



| Symbol | Parameter | Conditions | Standard-mode | | Fast-mode | | Fast-mode Plus | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| $f_{SCL}$ | SCL clock frequency | | 0 | 100 | 0 | 400 | 0 | 1000 | kHz |
| $t_{LOW}$ | LOW period of the SCL clock | | 4.7 | - | 1.3 | - | 0.5 | - | µs |
| $t_{HIGH}$ | HIGH period of the SCL clock | | 4.0 | - | 0.6 | - | 0.26 | - | µs |
| $t_r$ | rise time of both SDA and SCL signals | | - | 1000 | 20 + 0.1$C_b$[8] | 300 | - | 120 | ns |
| $t_f$ | fall time of both SDA and SCL signals[3][6][7][8] | | - | 300 | 20 + 0.1$C_b$[8] | 300 | - | 120[8] | ns |

Ref: UM10204 I2C-bus specification and user manual

# 3. Bus I2C, définitions

## START et STOP



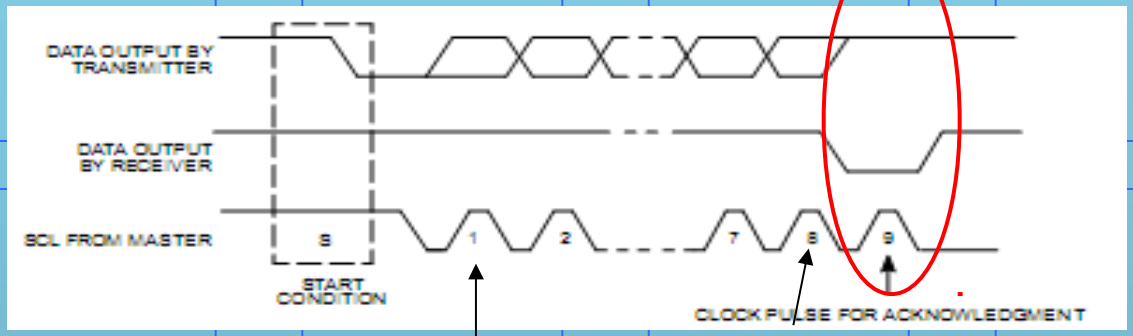

| Symbol | Parameter | Conditions | Standard-mode | | Fast-mode | | Fast-mode Plus | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| $t_{HD;STA}$ | hold time (repeated) START condition | After this period, the first clock pulse is generated. | 4.0 | - | 0.6 | - | 0.26 | - | µs |
| $t_{SU;STO}$ | set-up time for STOP condition | | 4.0 | - | 0.6 | - | 0.26 | - | µs |

Pour détecter les conditions START (S) et STOP (P) on viole la règle 1, modification de SDA quand SCL=1

Ref: UM10204 I2C-bus specification and user manual

# 3. Bus I2C, définitions

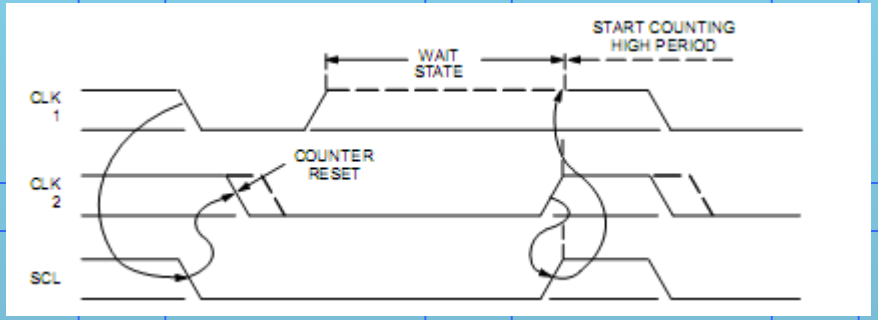
## Format d'un byte




Un byte est composé de 8 bits suivi d'un 9ème, appelé acknowledge. Ce bit est généré par SLAVE. Si ACK=0 (A), les 8 bits sont acquittés, et si ACK=1 (N), un problème se pose. Il sera par la suite appelé NACK (N).
C'est toujours MASTER qui contrôle SCL.

# 3. Bus I2C, définitions

## Clock stretching, clock synchronisation

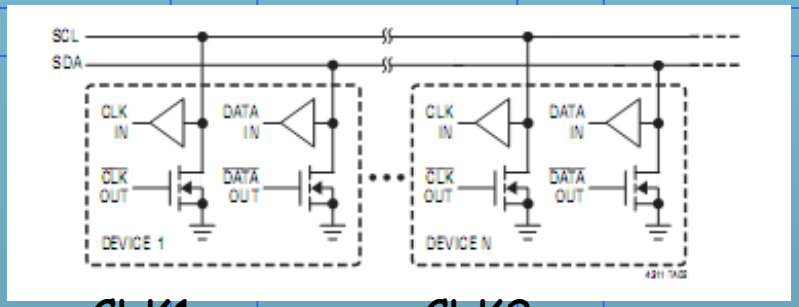



Master

Slave

SLAVE peut ralentir SCL du MASTER en tirant son CLK2 à 0. MASTER va essayer de mettre son CLK1 à 1, mais n'y parvient pas. Il se met en attente jusqu'à ce que SLAVE relâche son CLK2

Rappel:



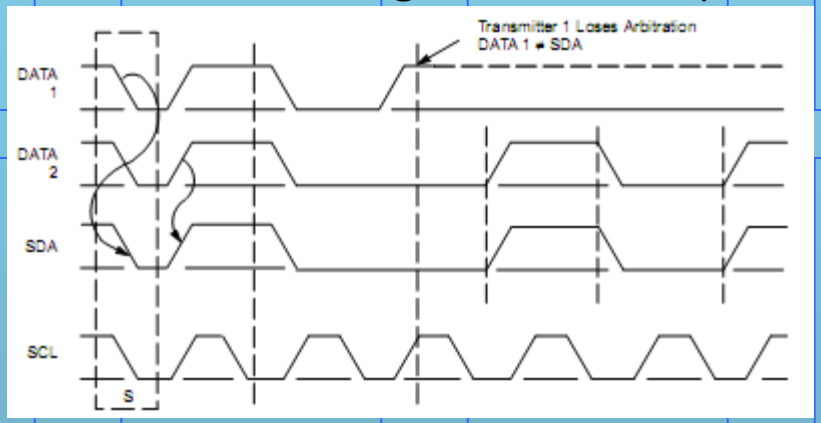

CLK1 CLK2

Ref: UM10204 I2C-bus specification and user manual

# 3. Bus I2C, définitions

## Arbitration

Une méthode similaire au clock stretching est utilisée pour l'arbitration.



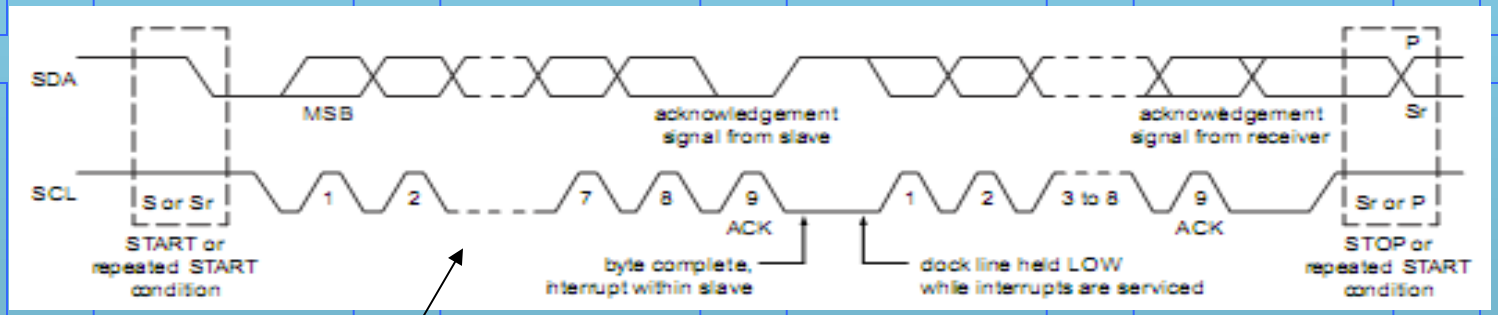

Master 1

Master 2

Dès que les MASTER ont détecté que le bus I2C est libre, ils commencent le transfert. Chaque contrôleur met son data sur la ligne et le relit systématiquement. Si un des circuits détecte une différence entre ce qu'il a sorti et sa relecture, il se déconnecte immédiatement.
L'arbitration est faite sur SDA, pendant que SCL=1. Les SLAVE ne sont pas concernés.

# 4. Trames de communication

## Format d'une trame quelconque



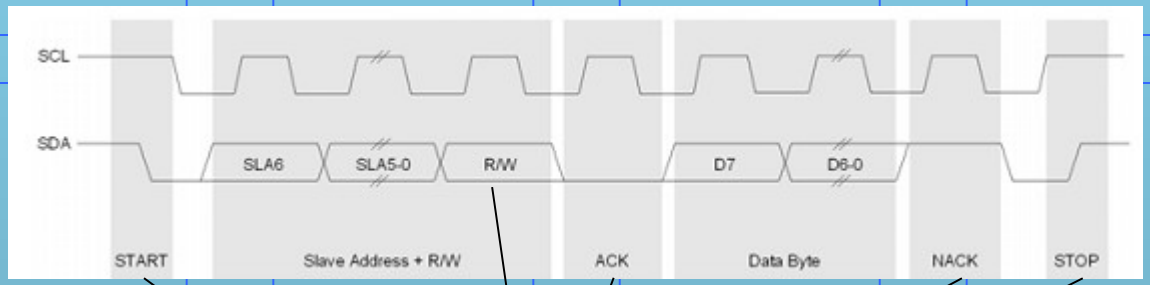

1er byte=commande

Data

# 4. Trames de communication

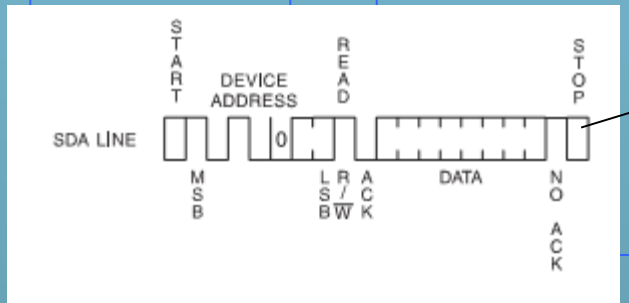
## Format d'une trame I2C

Pour simplifier la présentation, une notation plus synthétique est utilisée:




| S | SLAVE ID | 1 | A | DATA | N | STOP |
|---|---|---|---|---|---|---|

Souvent noté P



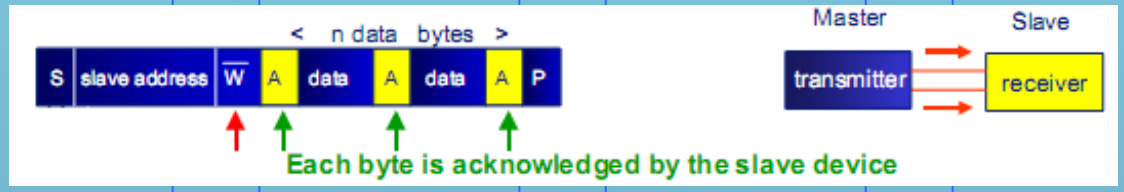

Ref: Silabs: AN113
Atmel AT24C128/256

# 4. Trames de communication

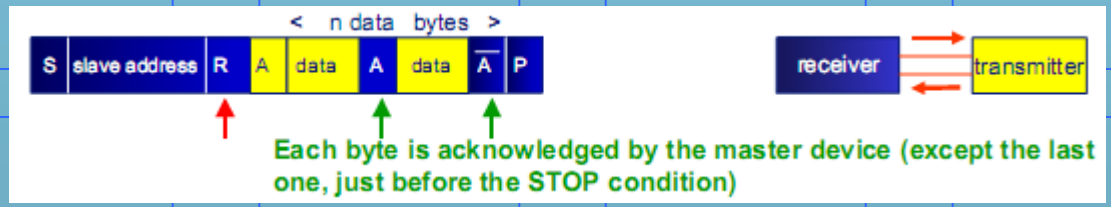
## I2C Read and Write Frame

Le premier byte de la trame est la commande. Le bit 0 indique si on a une trame de lecture (=1) ou d'écriture (=0), tandis que les 7 premiers bits indiquent l'adresse du SLAVE sélectionné. Les autres datas sont quelconques.

Write




Read

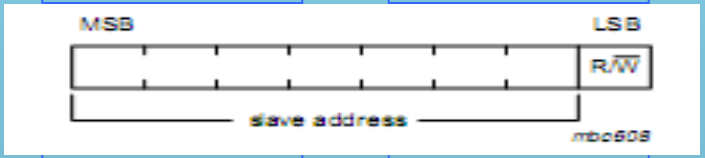



La taille d'une trame n'est pas définie. Elle va de 1 (minimum) à N, N dépendant du SLAVE

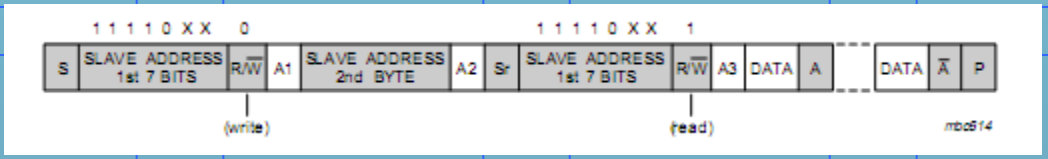
# 4. Trames de communication

## Format d'une trame I2C

Le premier byte après le START code le type de la trame ET l'adresse du périphérique que l'on veut atteindre.




Attention: certaines adresses ou combinaison d'adresse peuvent être réservées. Par exemple: 10-bit addressing.




Ici, 1111$0xx0b indique une adresse codée sur 10 bits.

# 4. Trames de communication

## Format d'une trame I2C

### 3.12 Reserved addresses

Two groups of eight addresses (0000 XXX and 1111 XXX) are reserved for the purposes shown in Table 3.

**Table 3. Reserved addresses**

| Slave address | R/W bit | Description |
|---|---|---|
| 0000 000 | 0 | general call address[1] |
| 0000 000 | 1 | START byte[2] |
| 0000 001 | X | CBUS address[3] |
| 0000 010 | X | reserved for different bus format[4] |
| 0000 011 | X | reserved for future purposes |
| 0000 1XX | X | Hs-mode master code |
| 1111 1XX | X | reserved for future purposes |
| 1111 0XX | X | 10-bit slave addressing |

[1] The general call address is used for several functions including software reset.

[2] No device is allowed to acknowledge at the reception of the START byte.

[3] The CBUS address has been reserved to enable the inter-mixing of CBUS compatible and I²C-bus compatible devices in the same system. I²C-bus compatible devices are not allowed to respond on reception of this address.

[4] The address reserved for a different bus format is included to enable I²C and other protocols to be mixed. Only I²C-bus compatible devices that can work with such formats and protocols are allowed to respond to this address.

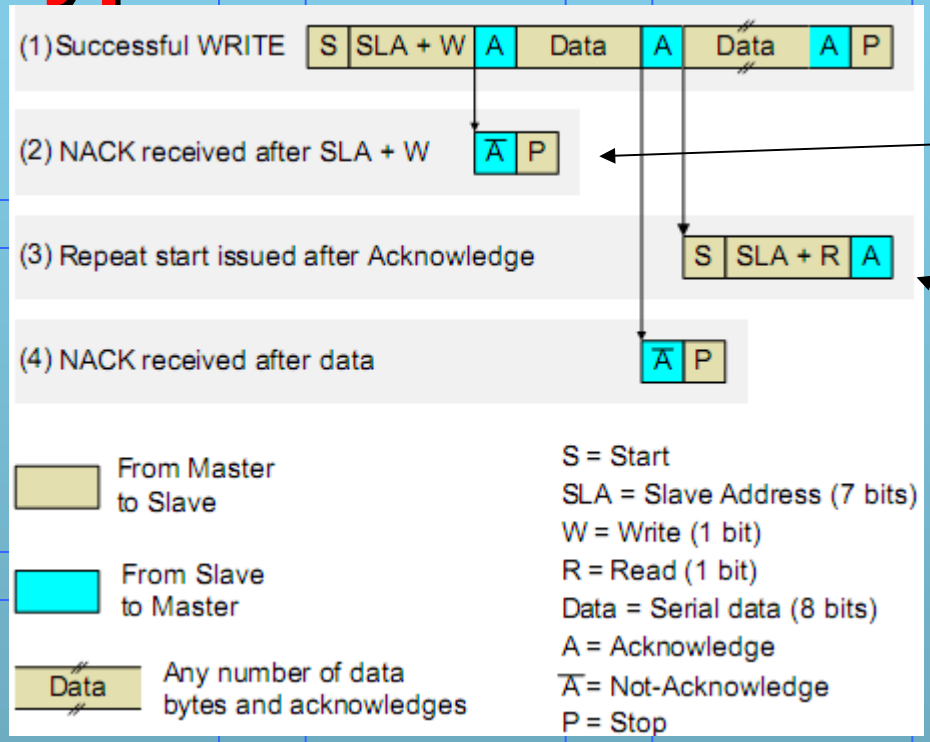
Toujours garder en tête que ces adresses peuvent être utilisées. Lire attentivement la documentation, surtout ce qui est écrit en petit caractère!

Pour plus de renseignements, allez sur internet et cherchez UM10204, la norme officielle I2C:
nxp.com/documents/user_manual/UM10204.pdf

Ref: UM10204 I2C-bus specification and user manual

# 4. Trames de communication

## Typical write frame



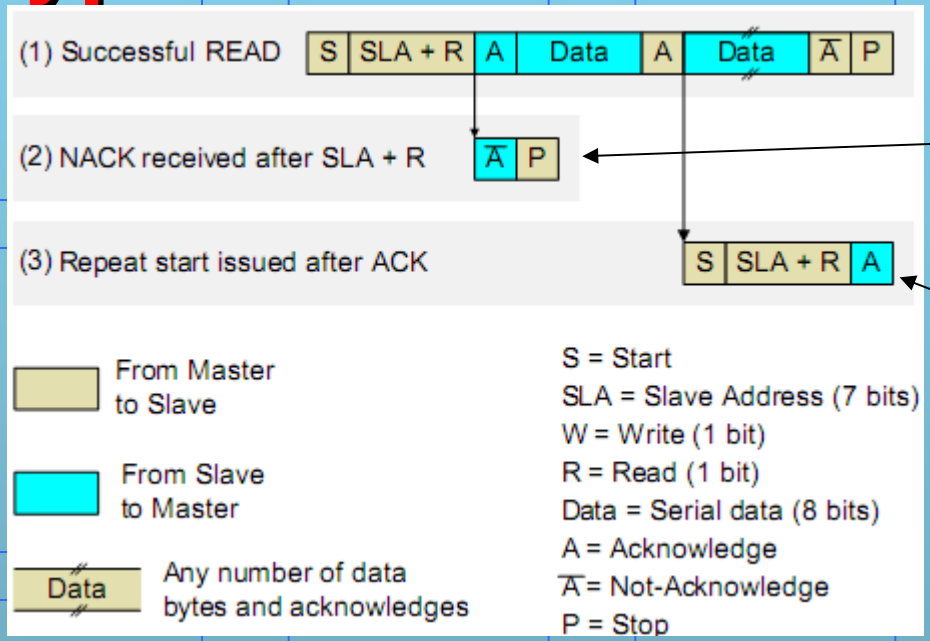

Ca cas correspond souvent à un SLAVE non branché.

Ce cas sera étudié plus tard

Ref: Silabs: AN113

# 4. Trames de communication

## Typical read frame




Ce cas correspond souvent à un SLAVE non branché.

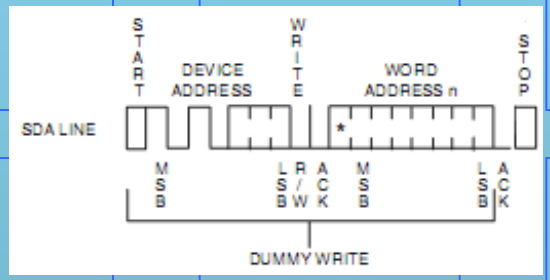
Ce cas sera étudié plus tard.

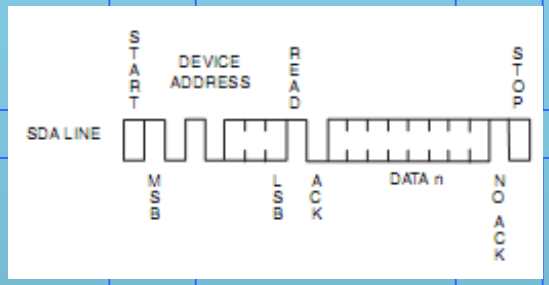
Ref: Silabs: AN113

# 4. Trames de communication

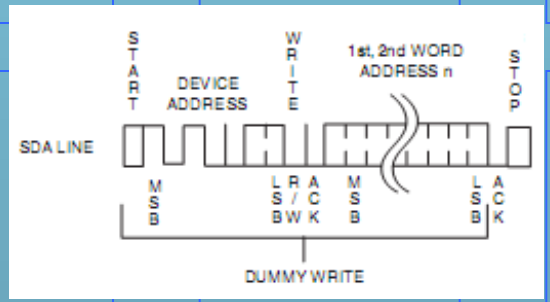
## Exemple, lecture dans une flash

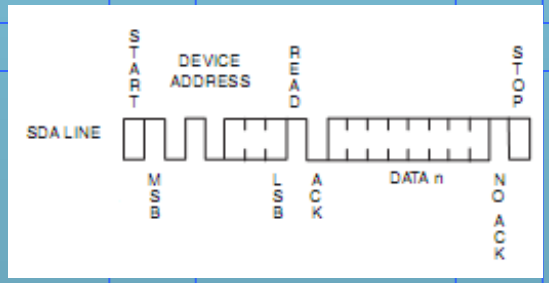
Par exemple, AT24C01B d'Atmel (128 bytes).




Par exemple, AT24C512x d'Atmel (64kbytes).



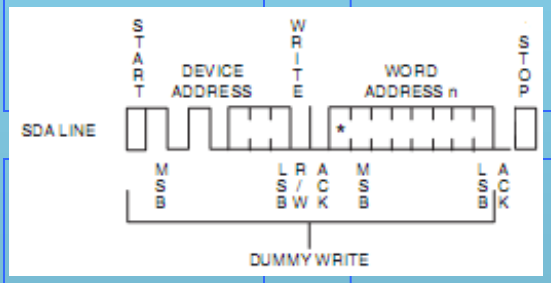




Ref:Atmel: Upgrading Two-Serial EEPROM Memory

# 4. Trames de communication

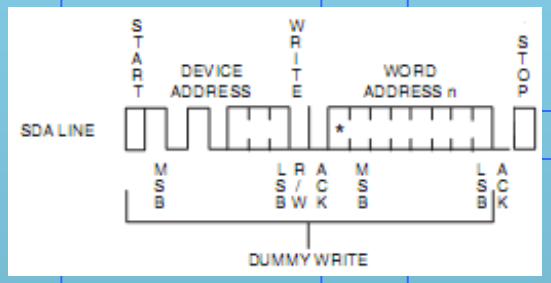
## Exemple, lecture multi-master dans une flash

Master1



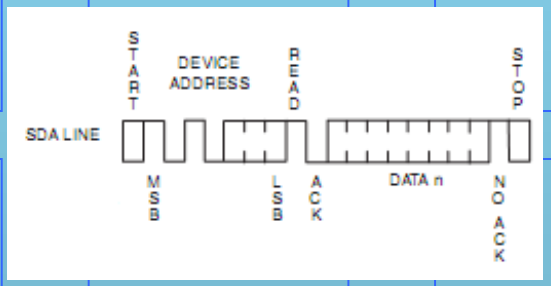

Bus released

Master2




Bus released

Master1



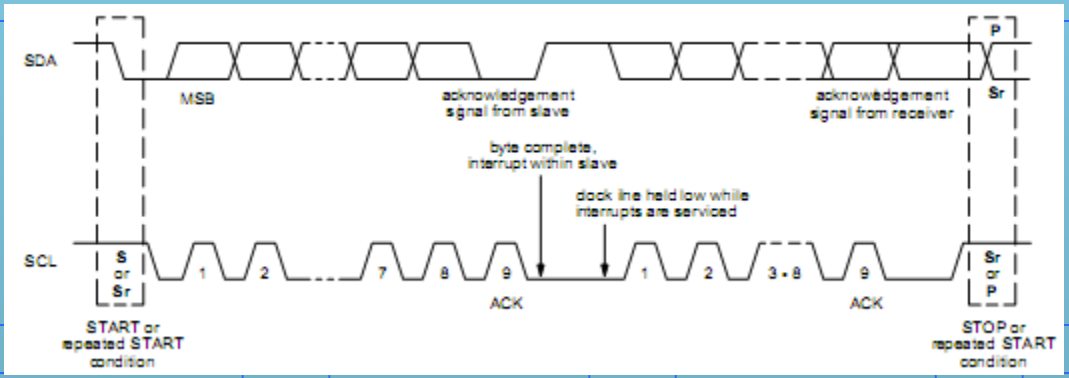

Master1 demande de lire à une adresse et relâche le bus. Master2 peut tout à fait démarrer sa demande de lecture et relâche son bus. Puis master1 fait sa lecture.
On voit bien que le data qu'il lit ne correspond pas à son adresse.
Il faut donc empêcher de relâcher le bus entre deux trames.

Ref:Atmel: Upgrading Two-Serial EEPROM Memory

# 3. Bus I2C, définitions

## Repeated START

Pour ne pas relâcher le bus, une condition « STOP » spéciale est introduite. Elle permet de relancer une trame SANS relâcher le bus I2C, ainsi un autre MASTER ne peut pas s'insérer entre les trames.



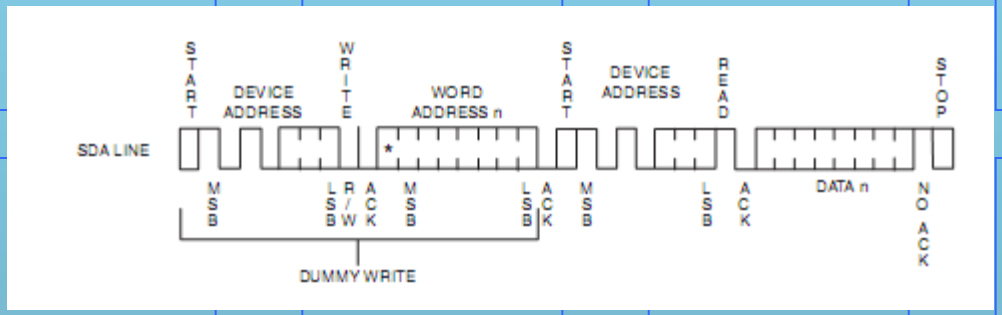

A la fin de la trame, le contrôleur génère à nouveau une condition START sans relâcher le bus I2C.

Ref: UM10204 I2C-bus specification and user manual

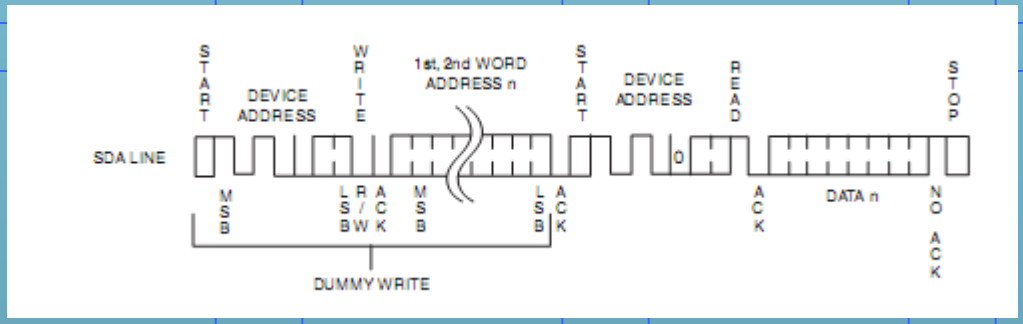
# 4. Trames de communication

## Lecture dans une flash en multi-master

Par exemple, AT24C01B d'Atmel (128 bytes).



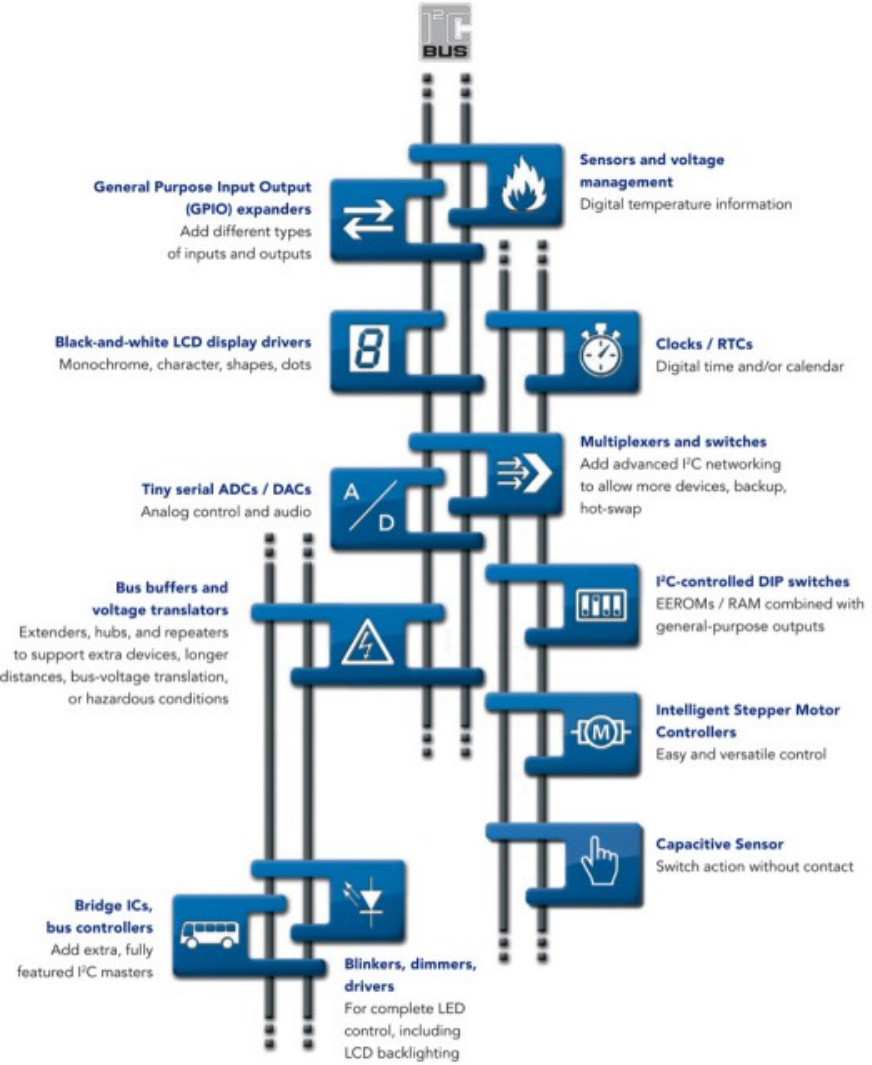

Par exemple, AT24C512x d'Atmel (64kbytes).




Ref:Atmel: Upgrading Two-Serial EEPROM Memory

# 5. Périphériques I2C

Un exemple de réseau avec différents capteurs I2C.




NXP dit que son portfolio est groupé autour de 12 familles de composants:

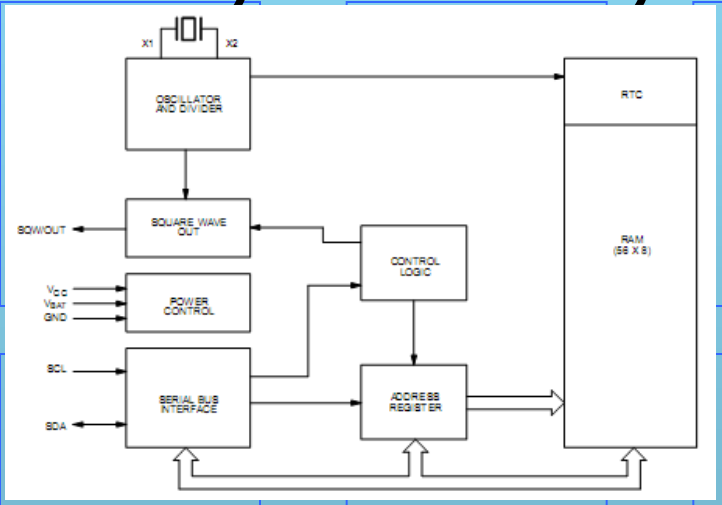
- Sensors (temperatur, capacitive sensor, pressure, …)
- Voltage management
- GPIO
- LCD display driver
- RTC, clock
- ADC, DAC
- Switchs, MUX
- Buffers, voltage translator
- Bridge

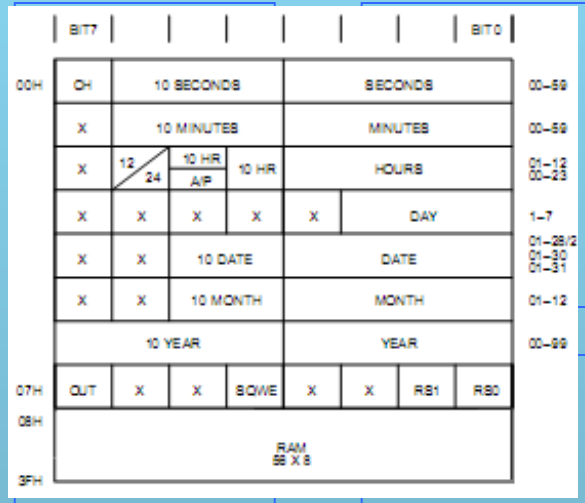
And the familie grow everyday…

Ref: NXP: I2C-bus solutions 2013

# 5. Périphériques I2C

## RTC, DS1307, 8 pins




**DATA WRITE – SLAVE RECEIVER MODE** Figure 6

| <Slave Address> | | <R/W> | | <Word Address (n)> | | <Data(n)> | | <Data (n+1)> | | <Data (n+X)> | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | 1101000 | 0 | A | XXXXXXXX | A | XXXXXXXX | A | XXXXXXXX | A | XXXXXXXX | A | P |

S – START
A – ACKNOWLEDGE
P – STOP

DATA TRANSFERRED (X+1 BYTES + ACKNOWLEDGE)

**DATA READ – SLAVE TRANSMITTER MODE** Figure 7

| <Slave Address> | | <R/W> | | <Data(n)> | | <Data(n+1)> | | <Data (n+2)> | | <Data (n+X)> | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | 1101000 | 1 | A | XXXXXXXX | A | XXXXXXXX | A | XXXXXXXX | A | XXXXXXXX | $\overline{A}$ | P |

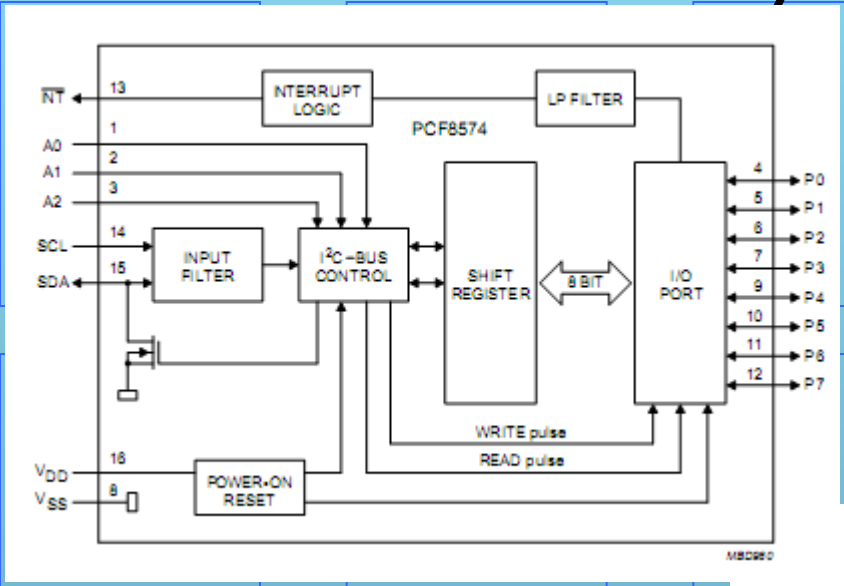
S – START
A – ACKNOWLEDGE
P – STOP
$\overline{A}$ – NOT ACKNOWLEDGE

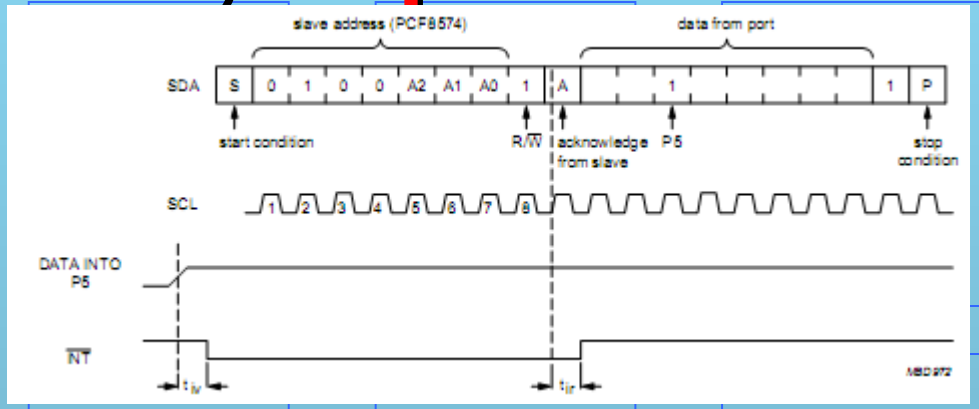
DATA TRANSFERRED (X+1 BYTES + ACKNOWLEDGE)

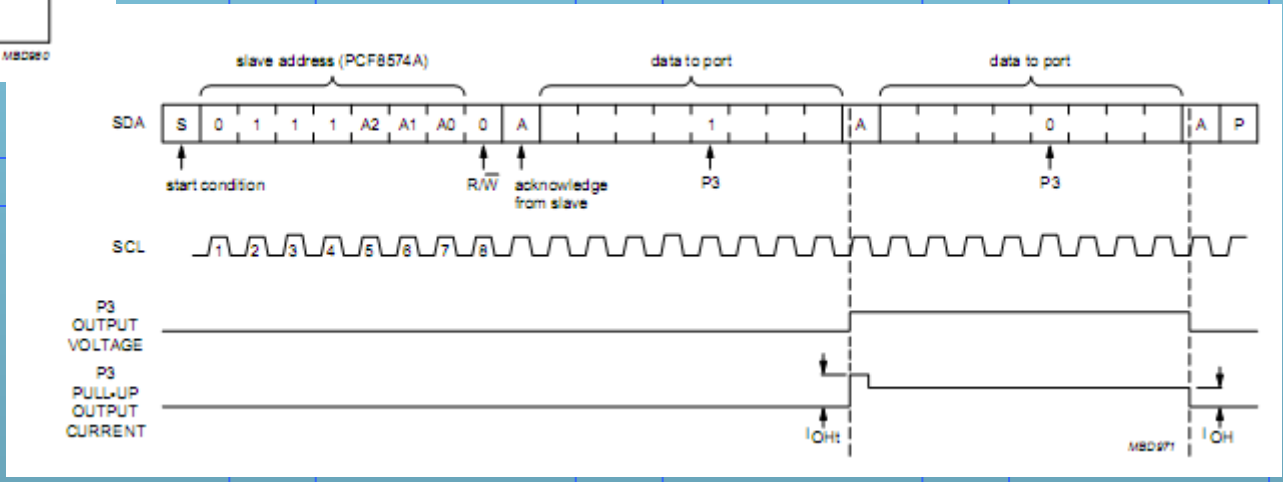
Ref:Dallas, DS1307/DS1308

# 5. Périphériques I2C

## Remote 8-bit I/O, PCF8574, 16 pins



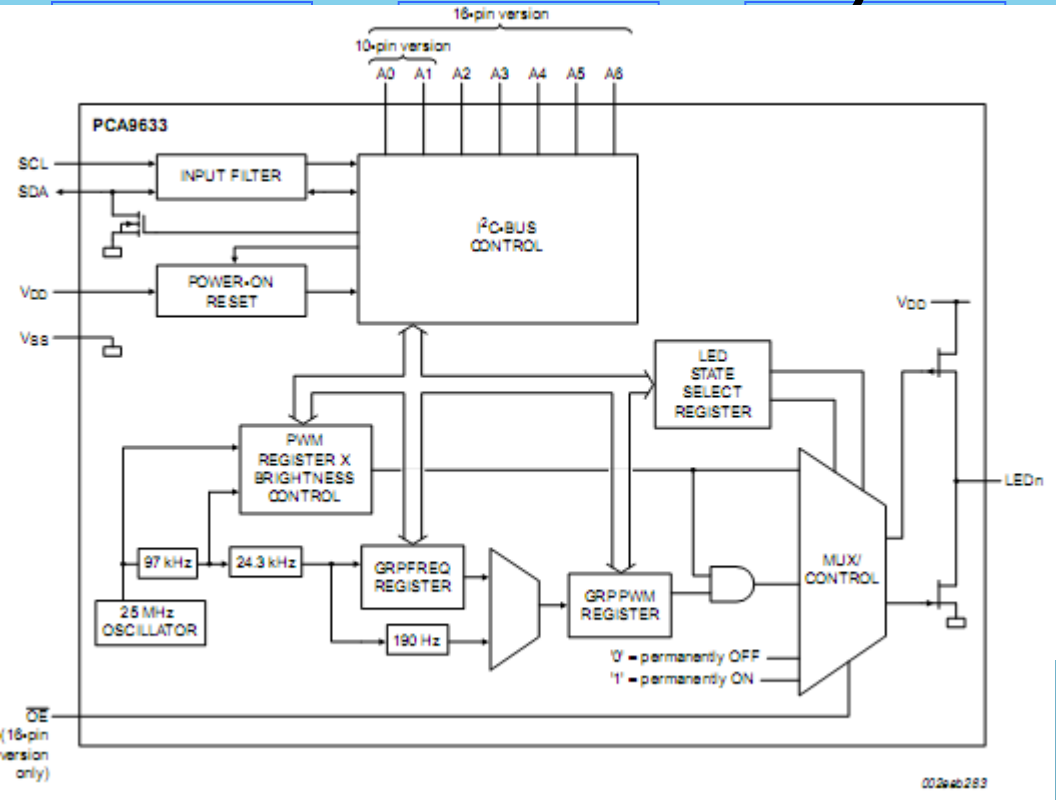




Ref: Philips, PCF8574

# 5. Périphériques I2C

# 4 bits LED driver, PCA9633, 8 pins




## 7.3 Register definitions

Table 7. Register summary[1][2]

| Register number (hex) | D3 | D2 | D1 | D0 | Name | Type | Function |
|---|---|---|---|---|---|---|---|
| 00h | 0 | 0 | 0 | 0 | MODE1 | read/write | Mode register 1 |
| 01h | 0 | 0 | 0 | 1 | MODE2 | read/write | Mode register 2 |
| 02h | 0 | 0 | 1 | 0 | PWM0 | read/write | brightness control LED0 |
| 03h | 0 | 0 | 1 | 1 | PWM1 | read/write | brightness control LED1 |
| 04h | 0 | 1 | 0 | 0 | PWM2 | read/write | brightness control LED2 |
| 05h | 0 | 1 | 0 | 1 | PWM3 | read/write | brightness control LED3 |
| 06h | 0 | 1 | 1 | 0 | GRPPWM | read/write | group duty cycle control |
| 07h | 0 | 1 | 1 | 1 | GRPFREQ | read/write | group frequency |
| 08h | 1 | 0 | 0 | 0 | LEDOUT | read/write | LED output state |
| 09h | 1 | 0 | 0 | 1 | SUBADR1 | read/write | I²C-bus subaddress 1 |
| 0Ah | 1 | 0 | 1 | 0 | SUBADR2 | read/write | I²C-bus subaddress 2 |
| 0Bh | 1 | 0 | 1 | 1 | SUBADR3 | read/write | I²C-bus subaddress 3 |
| 0Ch | 1 | 1 | 0 | 0 | ALLCALLADR | read/write | LED All Call I²C-bus address |

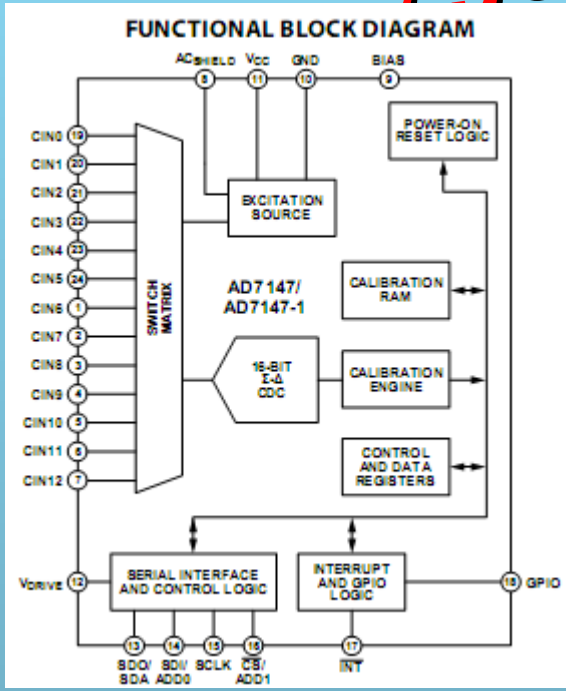
[1] Only D[3:0] = 0000 to 1100 are allowed and will be acknowledged. D[3:0] = 1101, 1110, or 1111 are reserved and will not be acknowledged.

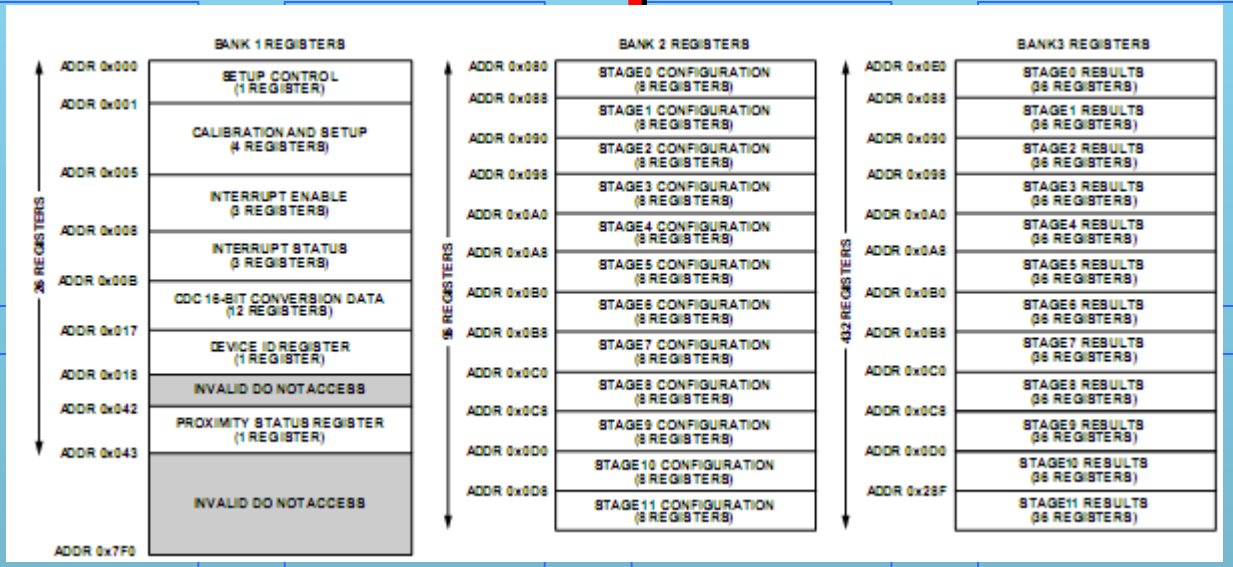
[2] When writing to the Control register, bit 4 must be programmed with logic 0 for proper device operation.

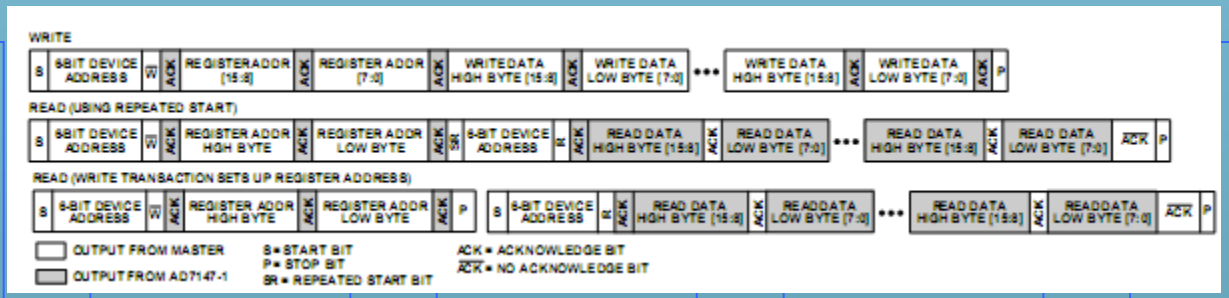

Ref: NXP, PCA9633

# 5. Périphériques I2C

# AD7147, gestion de clavier capacitif



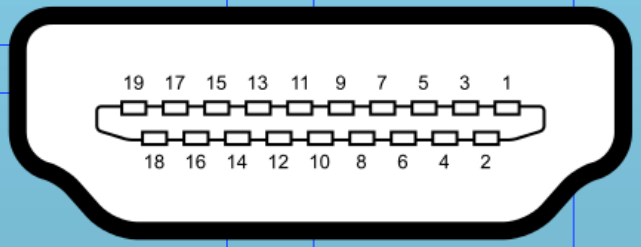




Ref: Analog Device, AD7147

# 5. Périphériques I2C

Il suffit de taper « circuit I2C » sur google, et on obtient 903'000 résultats!
Ce bus est devenu ZE standard, il est même utilisé dans la norme HDMI




| | | | |
|---|---|---|---|
| 1 | TMDS Data2+ | 2 | TMDS Data2 Shield |
| 3 | TMDS Data2– | 4 | TMDS Data1+ |
| 5 | TMDS Data1 Shield | 6 | TMDS Data1– |
| 7 | TMDS Data0+ | 8 | TMDS Data0 Shield |
| 9 | TMDS Data0– | 10 | TMDS Clock+ |
| 11 | TMDS Clock Shield | 12 | TMDS Clock– |
| 13 | CEC | 14 | HEC (HDMI 1.4) |
| 15 | SCL | 16 | SDA |
| 17 | DDC/CEC/HEC Ground | 18 | +5 V Power |
| 19 | Hot Plug Detect ou HEC (HDMI 1.4) | | |

Voilà, on a terminé la partie « naïve » ou de débutant. Maintenant essayons de fiabiliser tout ce bazard.

Ref: https://en.wikipedia.org/wiki/HDMI

# 6. Extension du protocole

Dans toute communication, il y a TOUJOURS des erreurs, quelque soit le soin que l'on mette pour le hardware. Faire un PCB respectant toutes les contraintes EMC, blinder les câbles, correctement adapter les lignes, n'empêcheront jamais des erreurs. Cela minimisera leur taux, mais il ne sera jamais nul.

Comment faire pour « durcir » le software?

Malheureusement, pour des composants « standards » (RTC, ADC, DAC, …) on ne peut pas faire grand-chose. En lecture, il est possible de lire deux fois le data et de comparer. Faire attention avec des ADC dont les valeurs d'entrées changent rapidement! Pour des DAC des contrôleurs de moteurs, ou autres sorties, il sera toujours obligatoire de rajouter du hardware pour relire indirectement ce qui a été écrit. Cela rajoute de la complexité, du prix et diminue la fiabilité de l'ensemble (plus de composants = plus de pannes possibles).

# 6. Extension du protocole

Pour les mémoires, qu'elles soient RAM, FRAM ou FLASH, ne pas écrire « naïvement » les informations. Rajouter systématiquement un CRC. Beaucoup de méthodes sont à disposition: CRC sur chaque data, CRC sur bloc ou CRC sur tout le composant physique.
La méthode que j'utilise assez fréquemment, sur des données, est la suivante:

Toutes mes données sont codées sur 32 bits, terminées par un CRC 8 bits

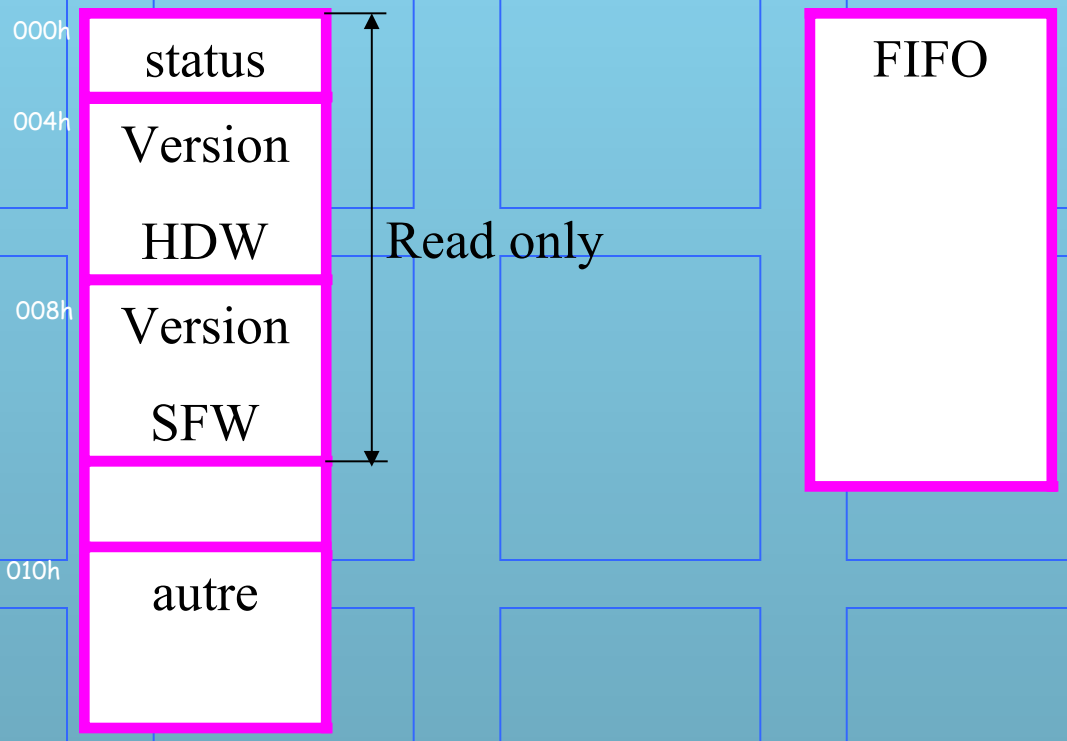
```
Var1:   ds   5  ; variable 1, 4 bytes + CRC
Var2:   ds  5  ; variable 2 ….
```

Et un tableau résume les plages (min..max) que peut prendre chaque variable, et sa valeur par défaut. Par exemple:

```
Tabvar1:  db   000h, 523h, 0ffffh, crc1     ; min, default, max, crc
Tabvar2:  db   100h, 100h, 01000h, crc2     ; min, default, max, crc
```

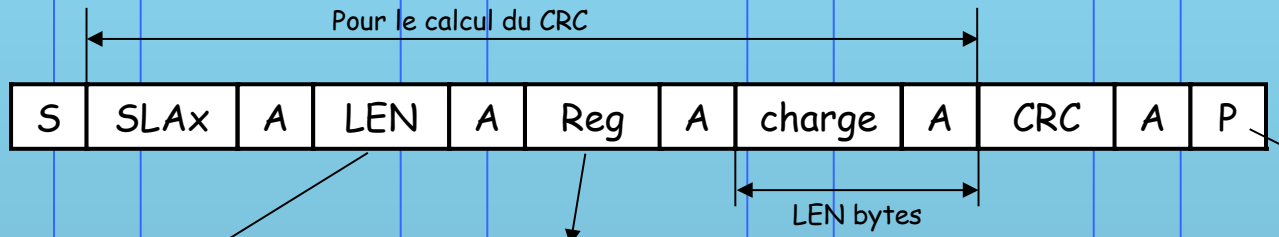
# 6. Extension du protocole

Dans pratiquement toutes les applications que j'ai développé avec des microcontrôleur, j'ai organisé les registres I2C internes du SLAVE ainsi:



# 6. Extension du protocole

On garde le principe de base des trames (on ne peut pas le modifier) et on introduit:



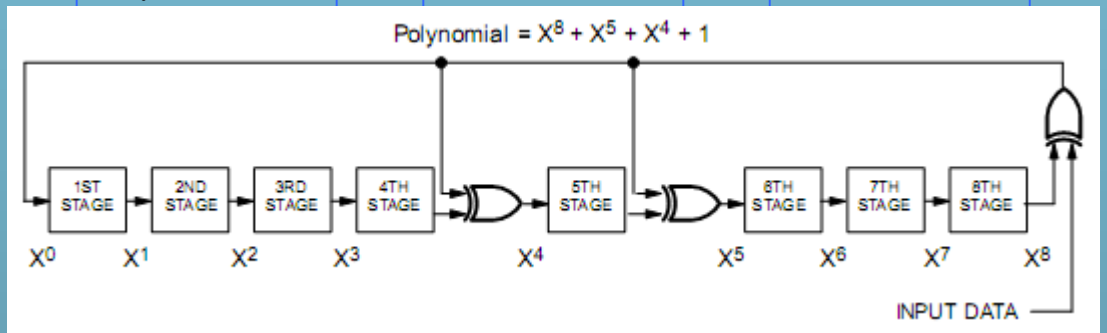

La lecture ou l'écriture de « charge » dans REG se fait dès que STOP (ou RepeatedStart) est détecté ET que la trame est OK.

codage

000h: commande
001h: read
002h: écriture dans FIFO
003h: réservé
010h-0ffh: registres I2C

| | codé | |
|---|---|---|
| | /LEN | LEN |
| exemple | 1101 | 0010 |

Le CRC est celui que Dallas utilise dans ses circuits « 1-wire bus »:




Ref: Dallas, AN27

# 6. Extension du protocole

Dallas propose cette procédure pour calculer le CRC:

```
Var
  CRC : Byte;
Procedure Do_CRC(X: Byte);
{
  This procedure calculates the cumulative Dallas Semiconductor 1-Wire CRC of all bytes passed to it.  The result
accumulates in the global variable CRC.
}
Const
  Table : Array[0..255] of Byte = (
      0,   94,  188,  226,   97,   63,  221,  131,  194,  156,  126,   32,  163,  253,   31,   65,
    157,  195,   33,  127,  252,  162,   64,   30,   95,    1,  227,  189,   62,   96,  130,  220,
     35,  125,  159,  193,   66,   28,  254,  160,  225,  191,   93,    3,  128,  222,   60,   98,
    190,  224,    2,   92,  223,  129,   99,   61,  124,   34,  192,  158,   29,   67,  161,  255,
     70,   24,  250,  164,   39,  121,  155,  197,  132,  218,   56,  102,  229,  187,   89,    7,
    219,  133,  103,   57,  186,  228,    6,   88,   25,   71,  165,  251,  120,   38,  196,  154,
    101,   59,  217,  135,    4,   90,  184,  230,  167,  249,   27,   69,  198,  152,  122,   36,
    248,  166,   68,   26,  153,  199,   37,  123,   58,  100,  134,  216,   91,    5,  231,  185,
    140,  210,   48,  110,  237,  179,   81,   15,   78,   16,  242,  172,   47,  113,  147,  205,
     17,   79,  173,  243,  112,   46,  204,  146,  211,  141,  111,   49,  178,  236,   14,   80,
    175,  241,   19,   77,  206,  144,  114,   44,  109,   51,  209,  143,   12,   82,  176,  238,
     50,  108,  142,  208,   83,   13,  239,  177,  240,  174,   76,   18,  145,  207,   45,  115,
    202,  148,  118,   40,  171,  245,   23,   73,    8,   86,  180,  234,  105,   55,  213,  139,
     87,    9,  235,  181,   54,  104,  138,  212,  149,  203,   41,  119,  244,  170,   72,   22,
    233,  183,   85,   11,  136,  214,   52,  106,   43,  117,  151,  201,   74,   20,  246,  168,
    116,   42,  200,  150,   21,   75,  169,  247,  182,  232,   10,   84,  215,  137,  107,   53);
Begin
  CRC := Table[CRC xor X];
End;
```

Ultrasimple à implanter et très rapide. Très fiable. Regarder aussi:

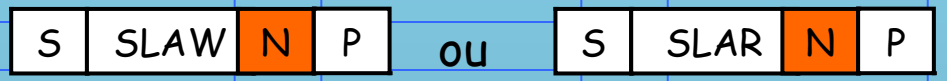
https://en.wikipedia.org/wiki/Polynomial_representations_of_cyclic_redundancy_checks

Ref: Dallas, AN27

# 6. Extension du protocole

La communication se fait toujours du MASTER vers le SLAVE. La seule manière au SLAVE d'indiquer au MASTER qu'un problème est survenu est d'agir sur le bit ACK. Dès qu'une erreur est détectée, SLAVE répond NACK. C'est au MASTER de savoir ce qu'il doit faire avec cette réponse.

Par exemple, MASTER envoie une trame vers un périphérique qui n'existe pas ou n'est pas branché:




MASTER "reçoit" NACK après SLAW ou SLAR. Le bit NACK vient "naturellement" si personne est branché sur le bus I2C, grâce à la pullup montée sur SDA.
Dans ce cas, que doit faire MASTER? Que veut dire cette erreur? On a les possibilités suivante:

1) MASTER fait une recherche des périphériques branchés sur le bus I2C: NACK indique qu'il n'y a pas de SLAVE.
2) MASTER fait une transmission normale: NACK indique que SLAVE n'est plus branché. S'est-il déconnecté? Fait-il un RESET? Est-il tombé en panne? Pire, est-ce une erreur « software » (trame fausse)?

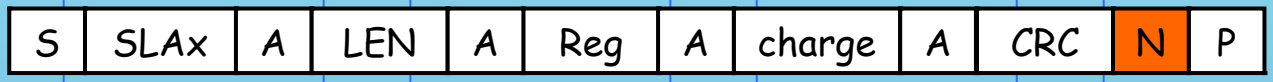
# 6. Extension du protocole

Autre cas possible:


SLAVE répond NACK sur le byte correspondant à REG. Ce byte indique le registre qui va être sélectionné, par exemple de 0 à 30. Si ADDR>30, il y a une erreur, d'où la génération du NACK et le MASTER doit terminer la transmission par un STOP. Il se pourrait aussi que le byte REG est pollué, et que la valeur résultante est plus grande que 30 (dans notre exemple).
A ce stade, on sait que la trame est fausse, c'est tout ce qui nous importe. Il faut réémettre la trame.

# 6. Extension du protocole

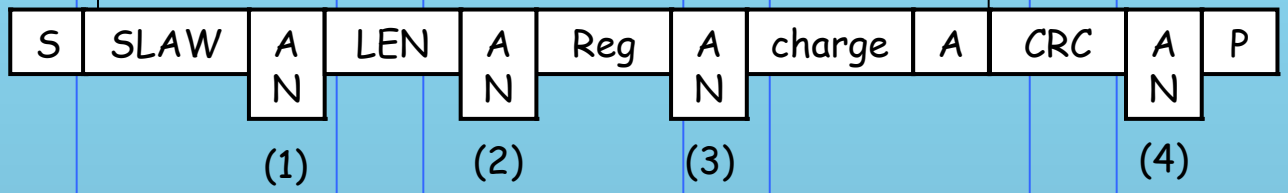
Autre cas possible:




Si NACK vient après le CRC, cela veut dire qu'un des éléments de la trame est faux. MASTER doit renvoyer celle-ci. Les données des registres ne sont pas modifiés.

# 6. Extension du protocole

Les erreurs possibles peuvent intervenir qu'à certaines positions:



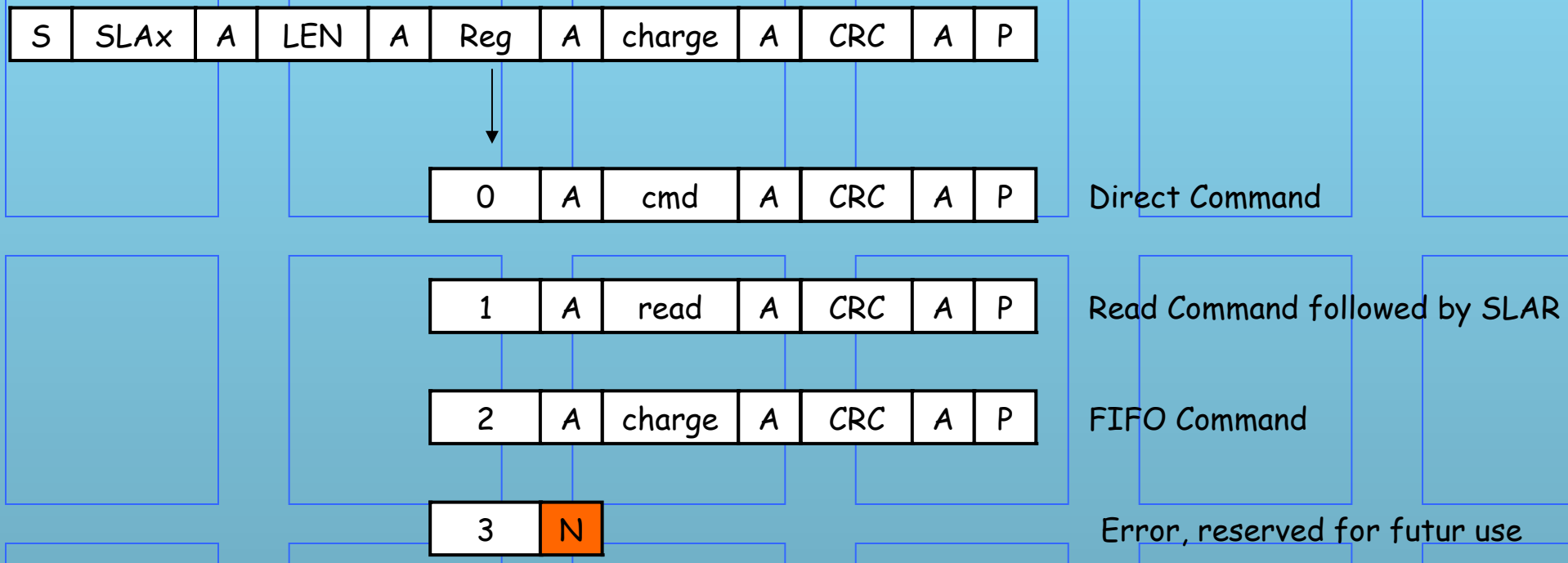

Dès qu'un NACK est détecté, un STOP est envoyé. Le MASTER va traiter ces erreurs ainsi:

1. Mauvaise adresse SLAx, ou SLAVE ne fonctionne pas (pas branché, en cours de RESET, pas d'alimentation, etc). Vérifier l'adresse du SLAVE puis relancer la trame.
2. Le codage de LEN est incohérent. Vérifier le codage puis ré-envoyer la trame.
3. Adresse des registres I2C faux. On essaye d'accéder un registre de SLAVE hors plage. Corriger REG et relancer la trame.
4. CRC error: la trame reçue (avec SLAW) est polluée. Un/des bits ont été modifiés. Corriger la trame et la renvoyer.

Dans la CHARGE il peut encore y avoir des erreurs spécifiques.

# 6. Extension du protocole

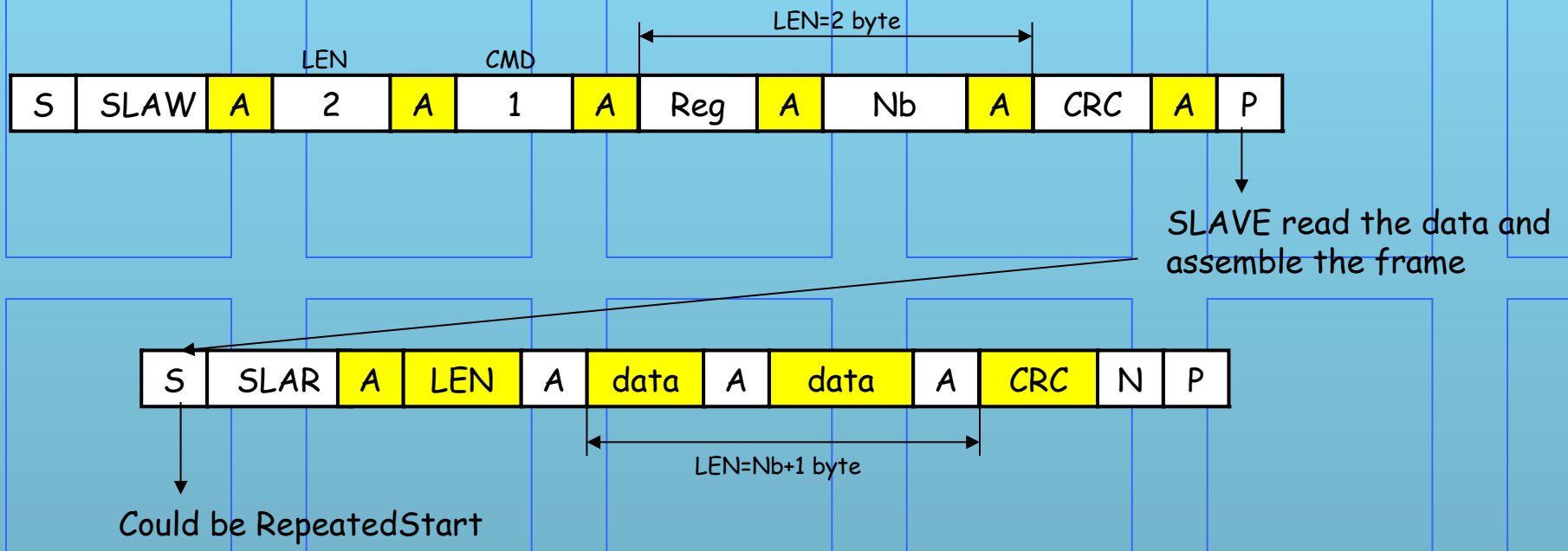
Si la trame est une commande, on peut avoir:




REG= 0 correspond à une commande directe, comme le RESET de la carte.
REG= 1 correspond à une demande de lecture.
REG= 2 correspond à l'écriture dans un FIFO de DATA
REG= 3 est pour l'instant réservé.

# 6. Extension du protocole

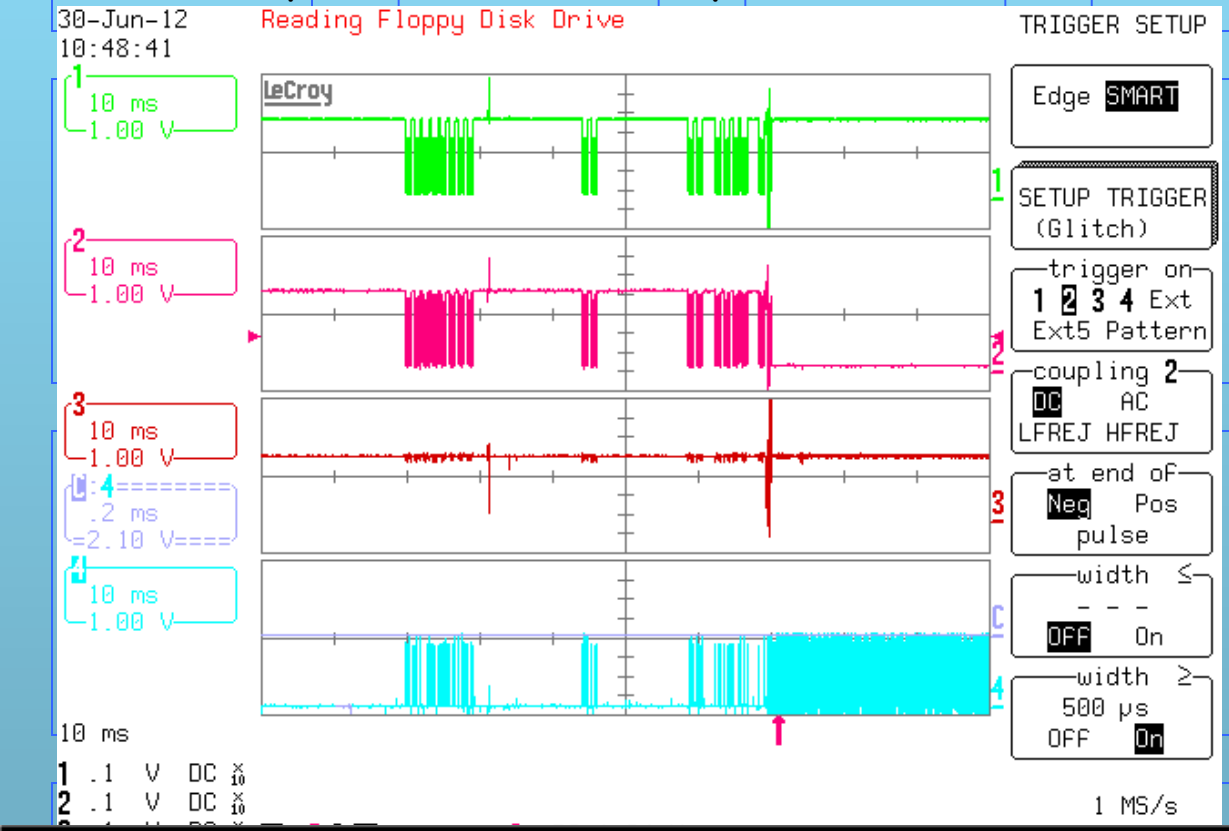
La commande READ est un peu spéciale:

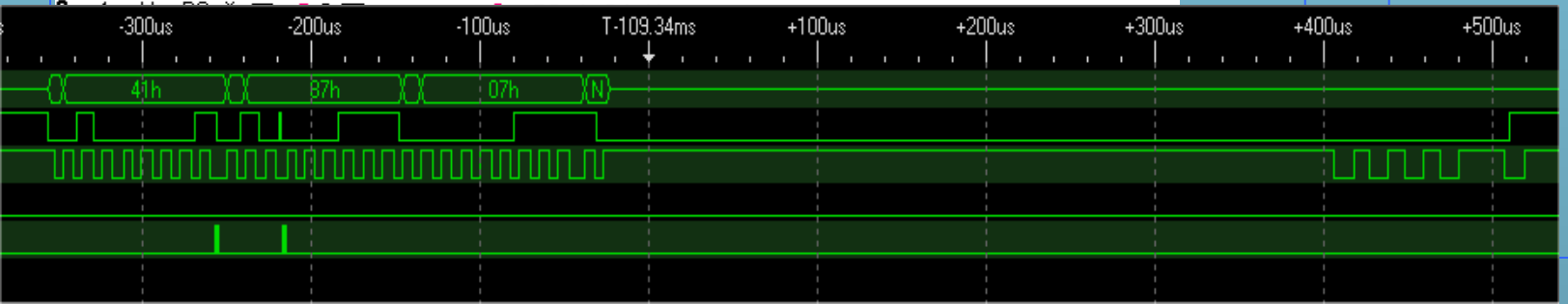



MASTER peut ainsi être sûr de ce qu'il reçoit

# 7. Cauchemars

Un classique: SDA reste bloqué à 0

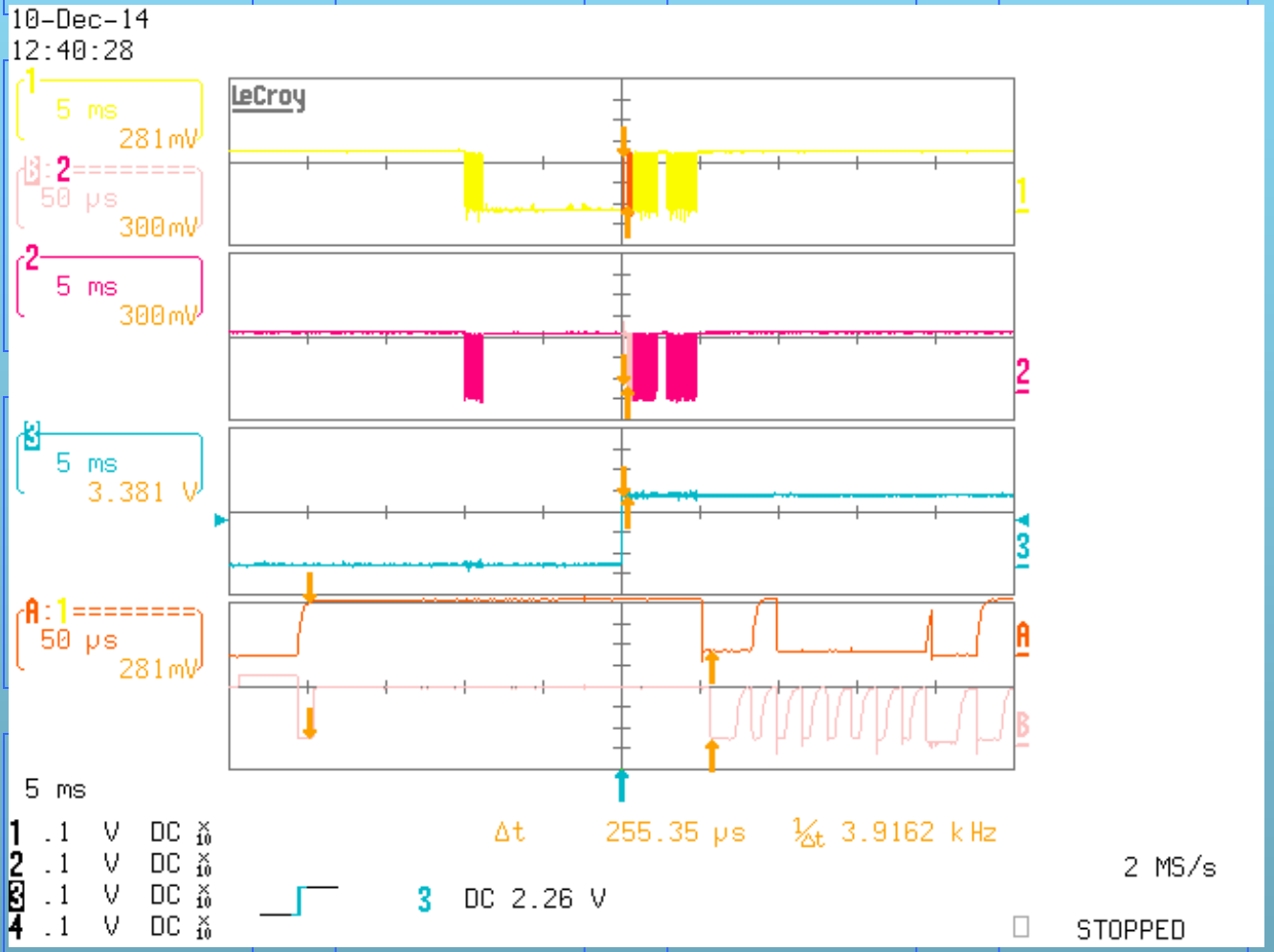



Pour détecter ce cas dans un système multi-master, j'utilise un timeout sur l'attente de la condition « free bus ». Si le bus ne se libère pas au bout d'un certain temps, j'essaye d'envoyer des clocks -> SDA=1. S'il reste bloqué au bout de 256 clocks, je suis très mal!

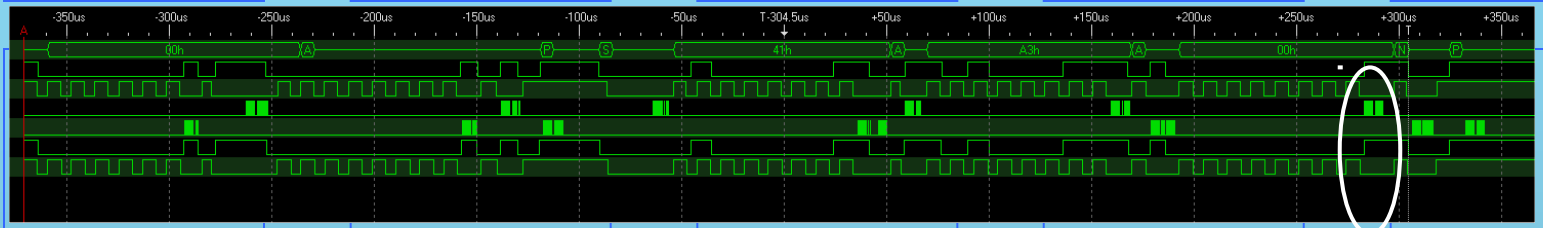
# 7. Cauchemars

Un classique: SDA reste bloqué à 0



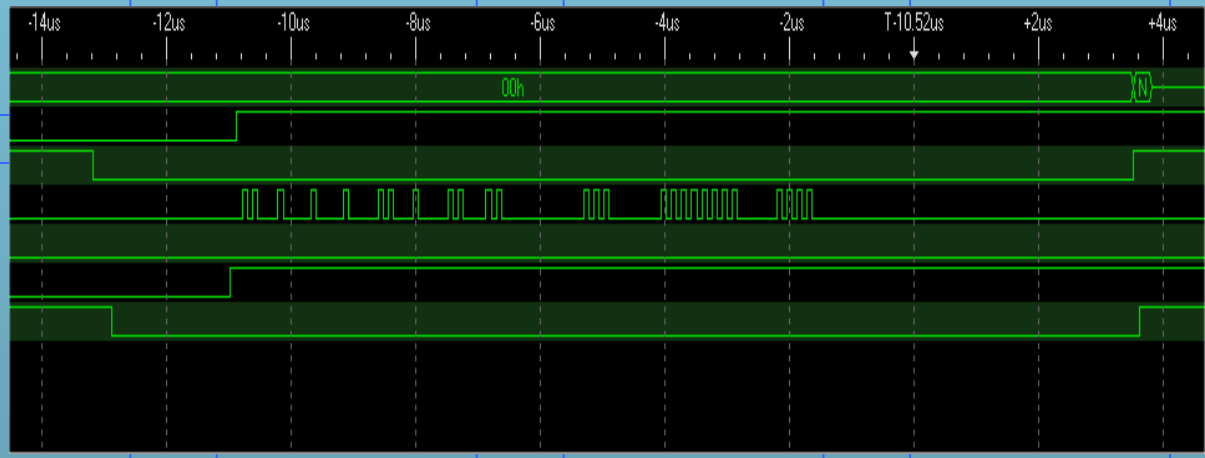
# 7. Cauchemars

Un plus drôle: SCL reste bloqué à 0




SCL peut être bloqué par MASTER ou SLAVE. La norme impose à celui qui met SCL=0 de programmer un timer qui force au bout d'un certain temps SCL=1. Dans le soft de gestion de l'I2C, ON DOIT prévoir un mécanisme qui remet automatiquement SCL à 1 (interruption NMI par exemple).

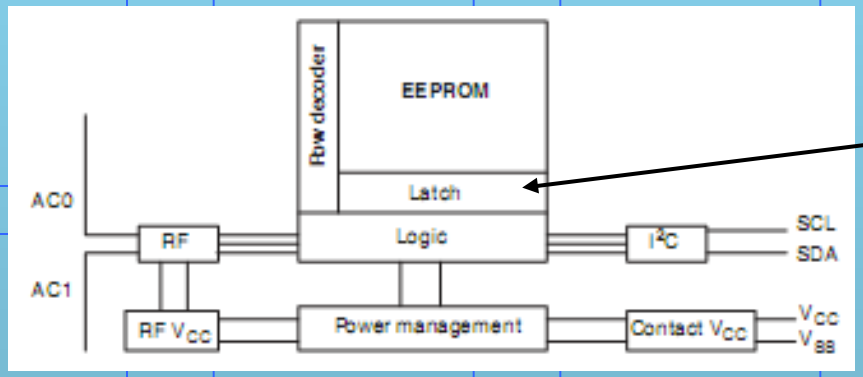
Ma méthode de debug: utiliser un port pour le debug et je code une information sur ce fils pour m'indiquer où je me trouve dans le soft.
Ici on a un byte qui code 1000$1011b
Et trois valeurs, 3, 8 et 4. Merci au C8051F380 qui est assez rapide pour faire cela!



# 7. Cauchemars

Un classique: les valeurs écrites en flash sont de temps en temps fausses.

Il faut aller au data sheet et vérifier comment on accède à la flash. Par exemple, pour le M24LR64-R de STMicroelectronics:




C'est ici que cela se passe. Quel est la taille de ce latch?

The Page Write mode allows up to 4 bytes to be written in a single Write cycle, provided that they are all located in the same "row" in the memory: that is, the most significant memory address bits (b12-b2) are the same. If more bytes are sent than will fit up to the end of the row, a condition known as 'roll-over' occurs. This should be avoided, as data starts to become overwritten in an implementation dependent way.

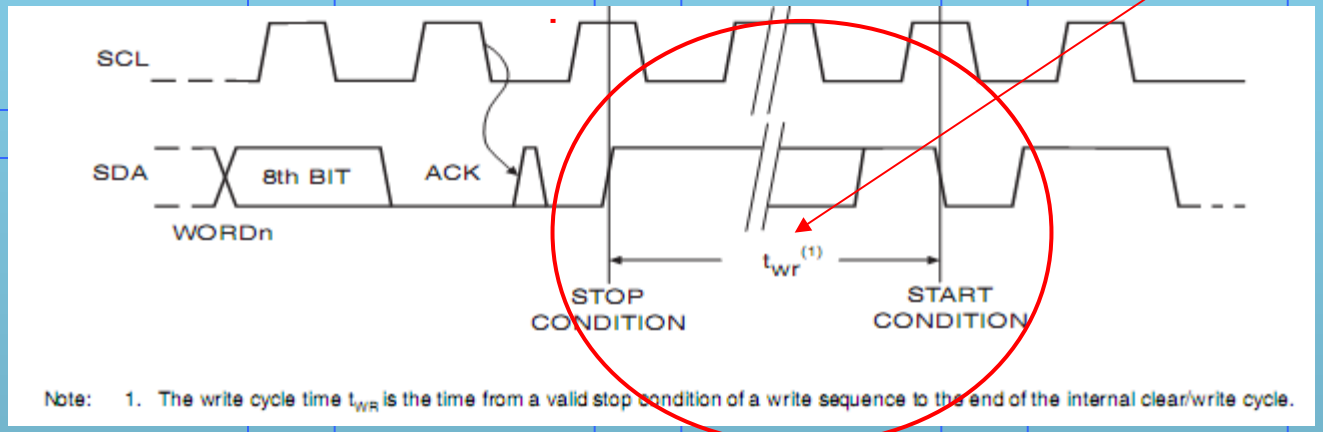
Bien entendu, ce paramètre est différent pour chaque flash!

Ref: STMicroelectronics, M24LR64-R

# 7. Cauchemars

Un classique: les valeurs écrites en flash sont toujours fausses, même en faisant attention au buffer.

Surtout pour les flash (AT24C128 d'Atmel), il faut « un certain temps » pour écrire du data, et change en fonction du fabricant de mémoire.




Note: 1. The write cycle time $t_{WR}$ is the time from a valid stop condition of a write sequence to the end of the internal clear/write cycle.

| Symbol | Parameter | 2.7-volt | | 5.0-volt | | Units |
|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | |
| $t_{WR}$ | Write Cycle Time | | 10 or 5[3] | | 10 or 5[3] | ms |

C'est long! Tous les constructeurs ne répondent pas forcement NACK pendant ce délais.

Ref: Atmel, Two-wire Serial EEPROMs

# 7. Cauchemars

Un classique: les valeurs écrites en flash sont toujours fausses, même en faisant attention au buffer et au temps Twr.

Surtout pour les flash, il existe une valeur appelée « endurance » qui donne le nombre de cycles de WR que l'on peut faire. Pour la 24AA256 de Microchip, on a:

| All parameters apply across the specified operating ranges unless otherwise noted. | Commercial (C): Vcc = +1.8V to 5.5V Tamb = 0°C to +70°C<br>Industrial (I): Vcc = +2.5V to 5.5V Tamb = -40°C to +85°C<br>Automotive (E): Vcc = +4.5V to 5.5V Tamb = -40°C to 125°C | | | | |
|---|---|---|---|---|---|
| | Symbol | Min | Max | Units | Conditions |
| Endurance | | 100K | — | cycles | 25°C, Vcc = 5.0V, Block Mode (Note 4) |

**Note 1:** Not 100% tested. $C_B$ = total capacitance of one bus line in pF.

**2:** As a transmitter, the device must provide an internal minimum delay time to bridge the undefined region (minimum 300 ns) of the falling edge of SCL to avoid unintended generation of START or STOP conditions.

**3:** The combined TSP and VHYS specifications are due to new Schmitt trigger inputs which provide improved noise spike suppression. This eliminates the need for a TI specification for standard operation.

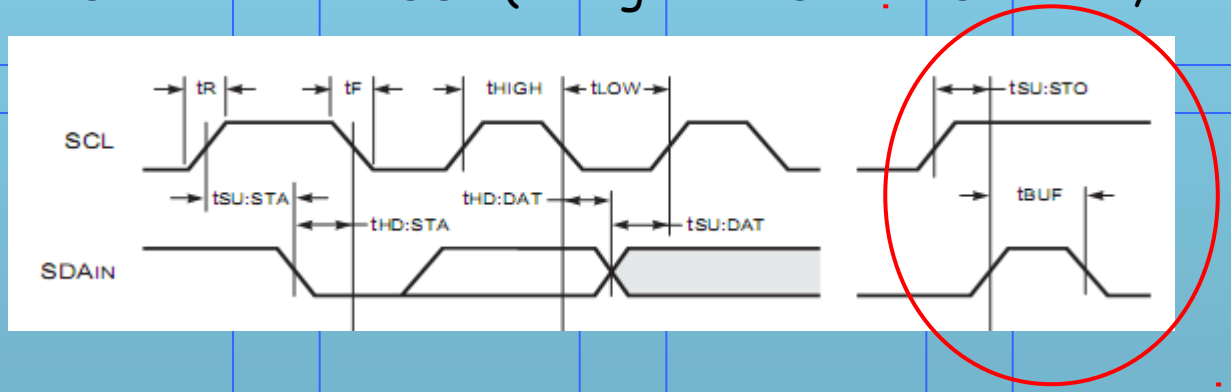
**4:** This parameter is not tested but guaranteed by characterization. For endurance estimates in a specific application, please consult the Total Endurance Model which can be obtained on Microchip's website @www.microchip.com.

Ca peut paraître beaucoup, mais pour des disques SSD (Solid-State Drive), ça limite méchamment la durée de vie! Il faut mettre en place une stratégie pour « répartir » régulièrement les écritures l'espace de la flash.
Et en plus, c'est une variable « échantillonnée »!

# 7. Cauchemars

Un classique très difficile: les valeurs relues en flash n'ont pas toujours la même valeur.

$t_{BUF}$ est un paramètre souvent négligé car les processeurs sont « lents ». Mais avec un C8051F380 tournant à 48[MHz] et un CLK processeur à 21[ns] et 50% du jeu d'instructions faisant 1 CLK, il faut IMPERATIVEMENT tenir compte de ce temps! Pour le IS24C256 de ISSI (Integrated Silicon Solution, Inc), on a:



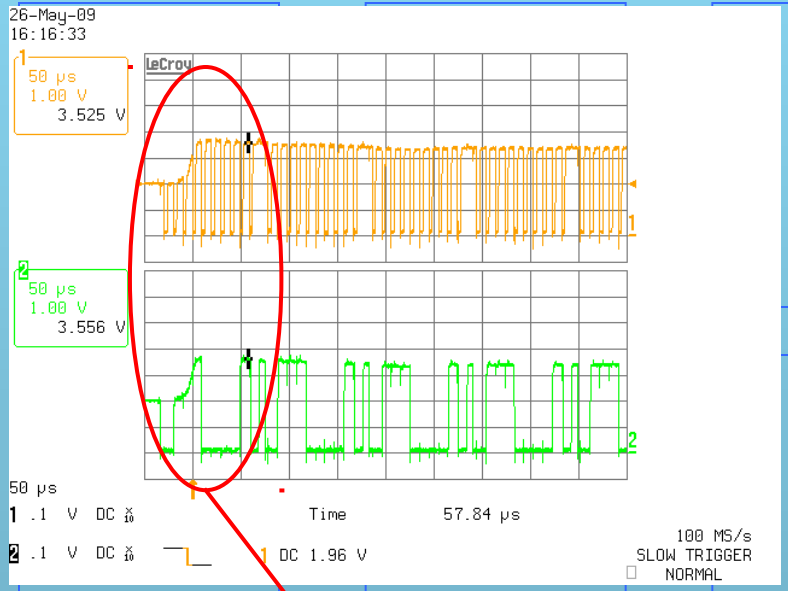

| Symbol | Parameter | 1.8V-5.5V Min. | 1.8V-5.5V Max. | 2.5V-5.5V Min. | 2.5V-5.5V Max. | 4.5V-5.5V Min. | 4.5V-5.5V Max. | Unit |
|---|---|---|---|---|---|---|---|---|
| fSCL | SCL Clock Frequency | 0 | 100 | 0 | 400 | 0 | 1000 | KHz |
| tBUF | Bus Free Time Before New Transmission | 4.7 | — | 1.2 | — | 0.5 | — | µs |

Cela fait quand même entre 100 et 150 instructions! Et ce temps dépend de la tension d'alimentation du circuit!

Ref: Integrated Silicon Solution Inc, IS24C256

# 7. Cauchemars

Un facile: temps de montée de l'alimentation non respecté.




Est-ce correctement interprété?

# 7. Cauchemars

Un particulièrement ardu avec des FIFO implantés dans SLAVE.

Dans un SLAVE on implante un FIFO avec par exemple 3 bytes par commande. Chaque trame aura dans sa « charge » exactement 3 bytes qui doivent être insérés dans ce FIFO.


Que se passe-t-il si le FIFO est plein? Généralement une désynchronisation de MASTER-SLAVE apparait avec un arrêt complet de la communication, sans possibilité de la faire redémarrer de manière software.

Pour ne JAMAIS tomber dans ce cas, je programme dans la routine d'interruption de SLAVE une fonction qui va vérifier AVANT de répondre au MASTER si le FIFO a assez de place pour accepter la commande au complet. Si c'est le cas, je réponds ACK, sinon NACK. MASTER devra renvoyer sa trame un certain temps plus tard.

# 7. Cauchemars

Si les grands fabricants de circuits se foutent de la norme …
Par exemple, Texas avec son TMS320x280x

**Table 5. I2C Mode Register (I2CMDR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 15 | NACKMOD | | NACK mode bit. This bit is only applicable when the I2C module is acting as a receiver. |
| | | 0 | In the slave-receiver mode: The I2C module sends an acknowledge (ACK) bit to the transmitter during each acknowledge cycle on the bus. The I2C module only sends a no-acknowledge (NACK) bit if you set the NACKMOD bit.<br>In the master-receiver mode: The I2C module sends an ACK bit during each acknowledge cycle until the internal data counter counts down to 0. At that point, the I2C module sends a NACK bit to the transmitter. To have a NACK bit sent earlier, you must set the NACKMOD bit. |
| | | 1 | In either slave-receiver or master-receiver mode: The I2C module sends a NACK bit to the transmitter during the next acknowledge cycle on the bus. Once the NACK bit has been sent, NACKMOD is cleared.<br>Important: To send a NACK bit in the next acknowledge cycle, you must set NACKMOD before the rising edge of the last data bit. |

Oups!!!!

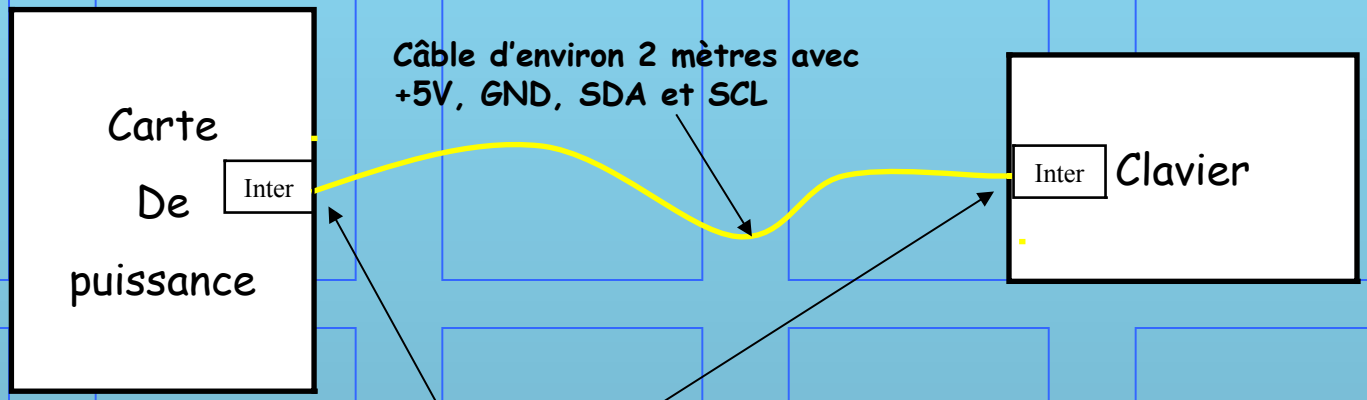
On comprend enfin la signification de ce paragraphe qu'après plusieurs dizaines d'heures de debug. Traduction: le bit ACK/NACK doit être généré AVANT le dernier data bit (!!!). Dans mon cas, avec le CRC suivi d'un STOP, il a fallu rajouter un byte « vide » après le CRC pour pouvoir détecter l'éventuel NACK. N'importe quoi…

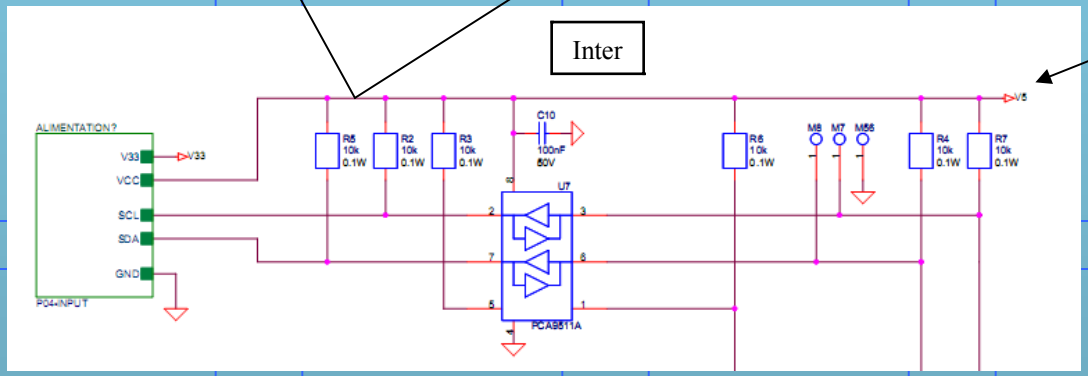
Texas, Freescale, Samsung et d'autres ont aussi d'autres petites surprises de ce genre, noyées dans plusieurs centaines de pages de documentation, souvent écrit en petits caractères.

Ref: Texas Instruments, SPRU721C

# 8. Mesures

Cas réel: un clavier avec un microcontrôleur, un afficheur OLED et des LED, est déporté d'une carte d'électronique de puissance.



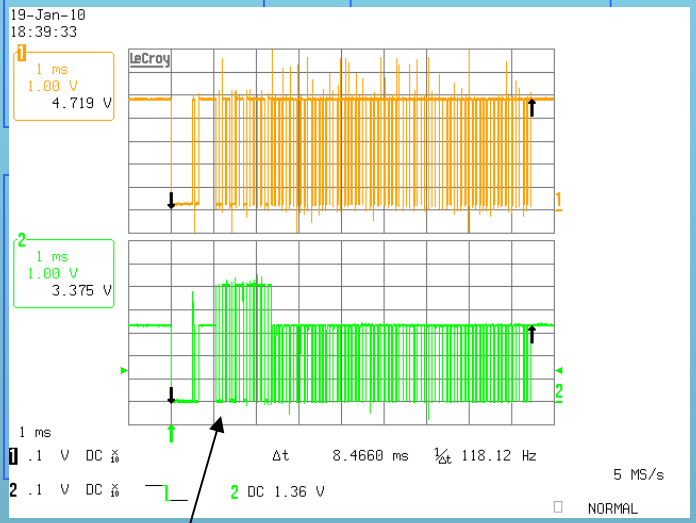

De chaque coté du câble on trouve un driver, le PCA8511A, alimenté sous 5[V]

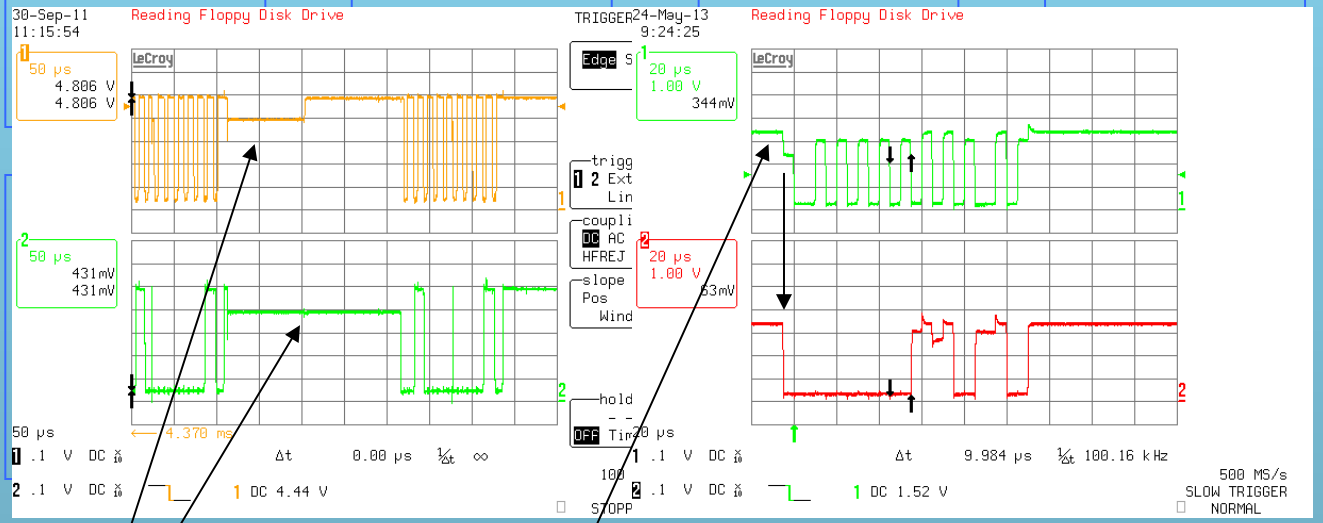
A cause de tout ce que vous allez voir, j'ai remplacé le driver PCA8511A par un P82B715, qui fonctionne beaucoup mieux, et j'ai ramené la tension d'alimentation du driver à 3.3[V].

# 8. Mesures

Un ingénieur est un peu comme un toubib: tous les gens qu'il côtoie sont malades. Les cartes qui reviennent sur mon bureau sont des « monstres » où tout ce qui peut « foirer » foire!

**Problèmes des tensions d'alimentation différentes, +5[V] et +3.3[V]**



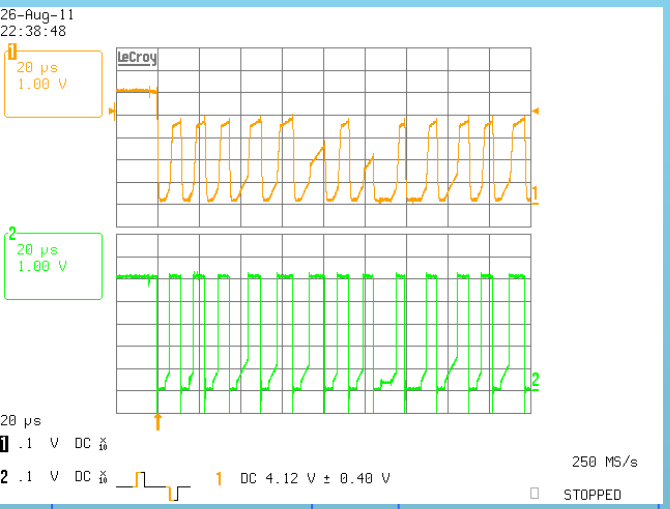

**Ici on voit quel est l'émetteur sur SDA**

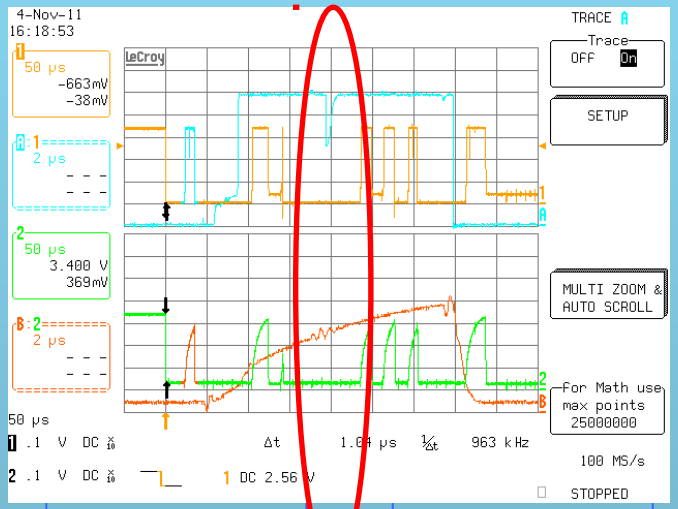
**Que s'est-il passé ici?**

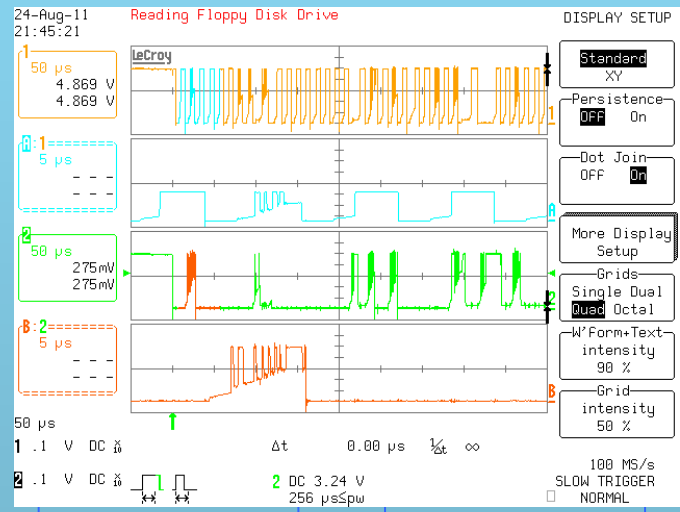
**Selon le niveau de bruit, les accélérateurs de tension du PCA9511A génère un START au mauvais moment**

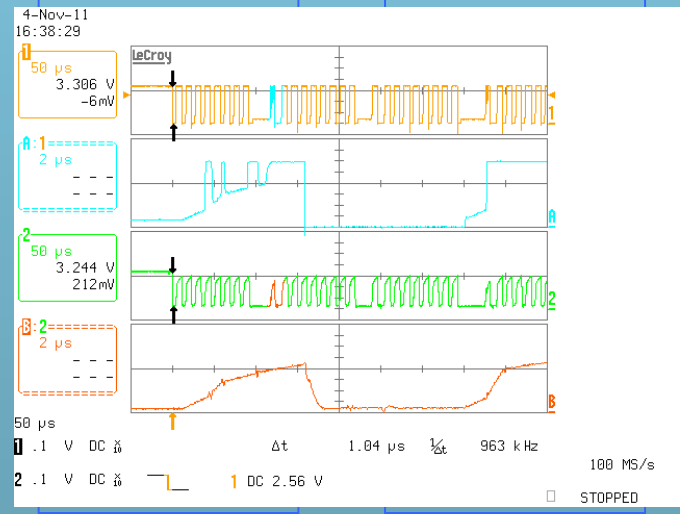
# 8. Mesures

Ici, la ligne (=câble) est mal adaptée, ou sa capacité est trop grande




**Différents cas de figure pour un signal qui ne respecte pas le temps $t_r$ rise time maximum du PCA9511A**

# 8. Mesures

Maintenant, quelques mesures réelles …