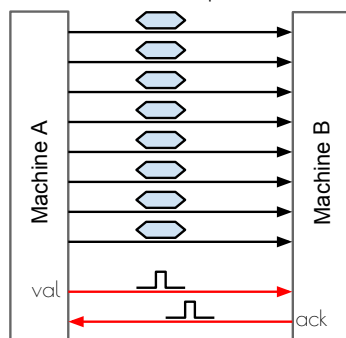


# Ports série RS232 et I2C

## Différence : Série / Parallèle

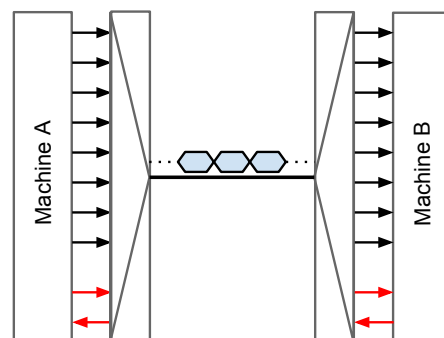
Transmission parallèle



Les bits sont envoyés en parallèle  
les caractères sont envoyés en série.

a priori plus simple, mais tous les signaux  
doivent arriver en même temps, c'est donc  
cher et difficile pour les grandes distances  
à haute fréquence.

Transmission série



Les bits de chaque caractères sont  
envoyés en série.

nécessite un sérialiseur/désérialiseur,  
mais tous les bits arrivent dans l'ordre  
cela semble plus long, mais on peut  
augmenter la fréquence.

# Différences technologiques

## ● RS232

- full duplex.
- pas de signal d'horloge
- 2 data (3 fils minimum : RX, TX, GND).
- signal non différentiel
- point à point
- de 75 bits/s à 115 kb/s

## ● SPI

- full duplex.
- horloge et data séparés (4 fils minimum : SCLK, MISO, MOSI, SS)
- signal non différentiel
- point à point
- adhoc jusqu'à 100Mb/s

## ● I2C ls / hs

- half duplex
- horloge et data séparé (3 fils : SDA, SCL, GND).
- signal non différentiel.
- bussé
- 100 kb/s à 3.4 Mb/s

## ● USB 1 / 2 / 3

- half duplex
- horloge et data mélangé (4 fils : VBUS, D+, D-, GND).
- signal différentiel
- point-à-point
- de 1.5 Mb/s à 5 Gb/s

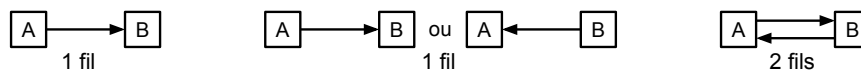
La vitesse est gagnée au prix de la complexité des protocoles et du matériel

3

# Différences technologiques

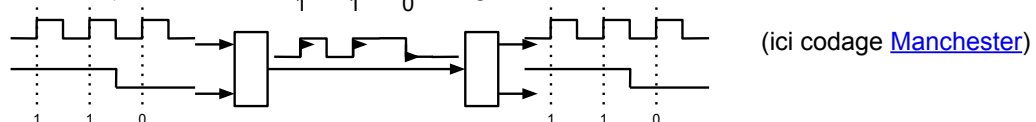
## ● Half duplex ou Full duplex

o transit dans un sens, les deux sens séparément ou en même temps.



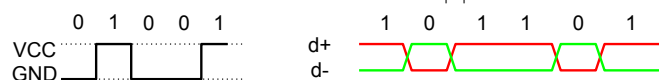
## ● Horloge et Data mélangés ou séparés

o 1 même fil pour les données et l'horloge, ou 2 fils.



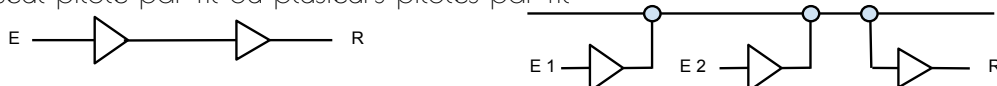
## ● Signal différentiel ou simple

o une donnée utilise 2 fils de valeurs opposées ou 1 seul valant 0 ou 1.



## ● Signal point-à-point ou bussé

o 1 seul pilote par fil ou plusieurs pilotes par fil

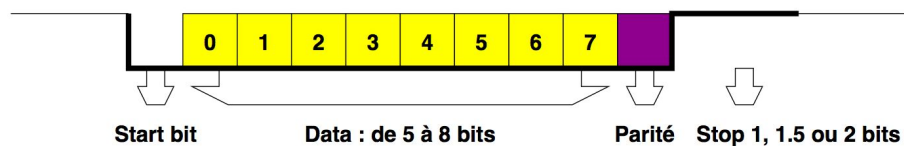
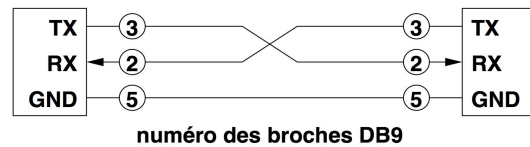


# RS232

- Protocole faible débit, simple et très diffusé, datant des années 60
- Pas d'horloge: l'émetteur et le récepteur s'entendent avant.
- Protocole *handshake* optionnel : CTS, RTS, ...
- Liaison point-à-point, pas de notion d'adresse.
- Trame de données de 5 à 8 bits avec parité.
- La parité est optionnelle:
  - parité paire: le nombre de 1 de la donnée et du bit de parité doit être pair
  - parité impaire: c'est le contraire

- RS232 prévoit plusieurs types de cablages:
  - le cablage null-modem définit la communication entre 2 terminals

- au minimum 3 fils : TX, RX et GND
- on peut avoir besoin d'un convertisseur de niveaux électriques :
  - 0 logique : +8 à +12V
  - 1 logique : -8 à -12V

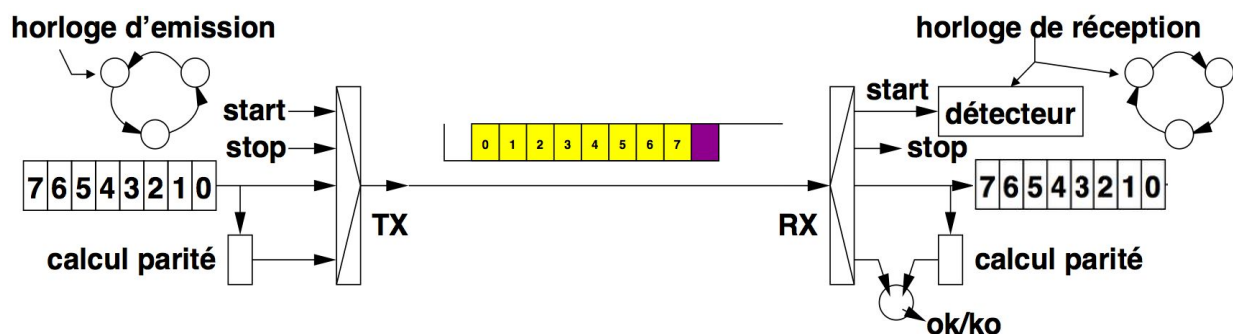


## RS232 Schéma de principe

**Émetteur**      La trame est produite par un automate qui vide un registre à décalage

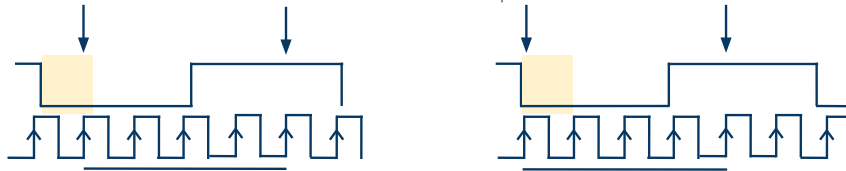
**Récepteur**      La trame est lue par un automate qui remplit un registre à décalage

**Parité**      Un bit supplémentaire qui signe la donnée  
 parité paire : le nombre de bit à 1 de la donnée est rendu pair grâce au bit de parité.  
 parité impaire : c'est le contraire

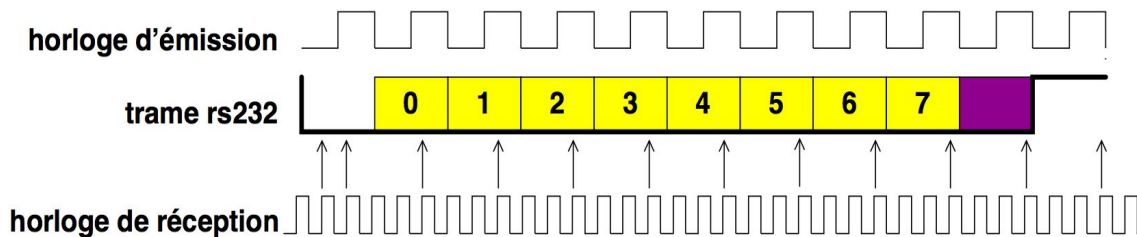


# RS232 Synchro émetteur/récepteur

- L'émetteur transmet à une fréquence standardisée (1200, 2400, 4800...)
- Le récepteur connaît cette fréquence et sur-échantillonne pour repérer start
  - Si la fréquence d'échantillonnage est 3 fois la fréquence d'émission → 1 bits = 3 périodes
  - Lorsqu'on lit 0, on est au dans le premier tiers du start
  - L'échantillon suivant est le bit 0, il est pris "1.5 bit" plus tard → 4 périodes
  - Les bits de la trame sont alors lu toutes les 3 périodes



- Le récepteur a une petite marge d'erreur possible sur la fréquence.



# I2C caractéristiques principales

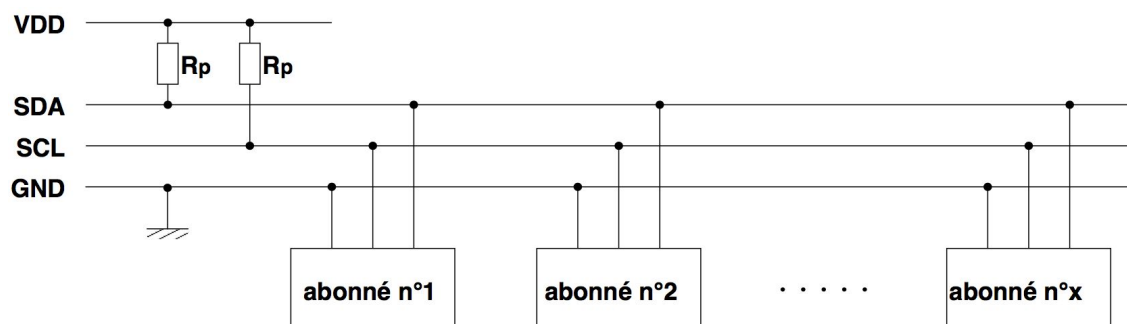
- I2C = IIC = Inter Integrated Circuit
- Protocole défini dans les années 80 par Philips.
- Protocole simple et très diffusé.
- Jusqu'à 128 abonnés (version de base) communiquent sur 3 fils :
  - SCL (horloge) .
  - SDA (data) .
  - GND (tension de référence).
- Abonnés Plug and Play acceptant le Hot Plug (branchement à chaud).
- Bus multi-maitres, tout abonné peut devenir maître du bus.
- Arbitrage décentralisé.
- débit de 100Kbauds à 3.4Mbauds.
- Adaptation du débit en fonction de l'abonné.
- Permet la communication entre différentes technologies (5 et 3.3V).

# I2C Glossaire

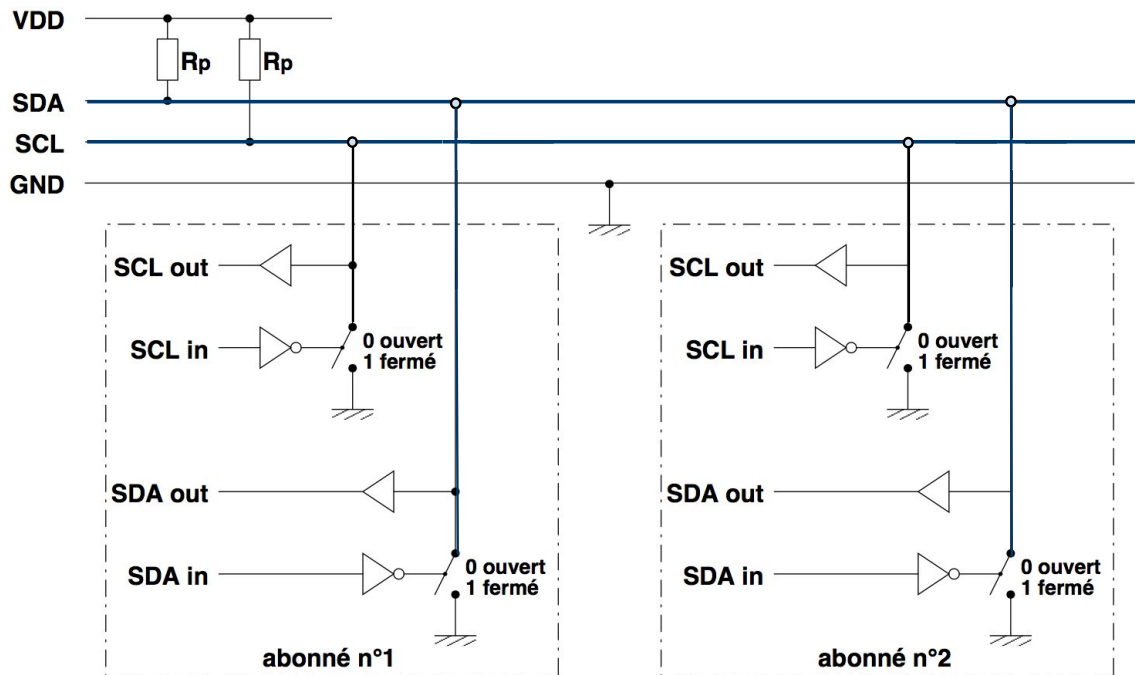
- abonné tout élément connecté sur le bus.
- émetteur tout abonné qui envoie des données sur SDA.
- récepteur tout abonné qui reçoit des données de SDA.
- maître tout abonné qui démarre et termine un échange.  
Le maître place l'horloge sur SCL.
- esclave tout abonné adressé par un maître.  
Un esclave à la possibilité de ralentir l'horloge du maître.
- adresse numéro attribué à un esclave.  
Sur le bus tous les esclaves ont une adresse unique.
- échange dialogue entre un maître et un esclave.  
Il commence par une adresse émise par le maître, suivie d'une ou plusieurs données émises par le maître ou l'esclave.  
Un maître peut chaîner plusieurs échanges d'affilé.
- arbitrage résolution du conflit d'un accès simultané par 2 maîtres.

# I2C cablage

- Les lignes SCL et SDA sont à VDD si aucun abonné ne parle.
- Pour mettre 1 sur SCL ou SDA, un abonné programme le port en entrée, la résistance  $R_p$  se charge de tirer la ligne à 1
- Pour mettre 0 sur SCL ou SDA, un abonné doit écrire un 0, c.-à-d. relier la ligne à la masse.
- Il ne peut jamais y avoir de conflit électrique (court-circuit VDD-GND).



## I2C schéma de principe



## I2C principe d'un échange

Le maître :

- émet une condition de démarrage
- envoie une adresse sur 7 bits
- envoie la commande r/w
- lit l'accusé et stoppe si NACK
- pour une écriture, il boucle sur
  - envoie les 8 bits de donnée
  - lit l'accusé et stoppe si NACK
- pour une lecture, il boucle sur
  - lit les 8 bits de donnée
  - émet ACK, ou NACK pour stopper
- émet une condition de stop

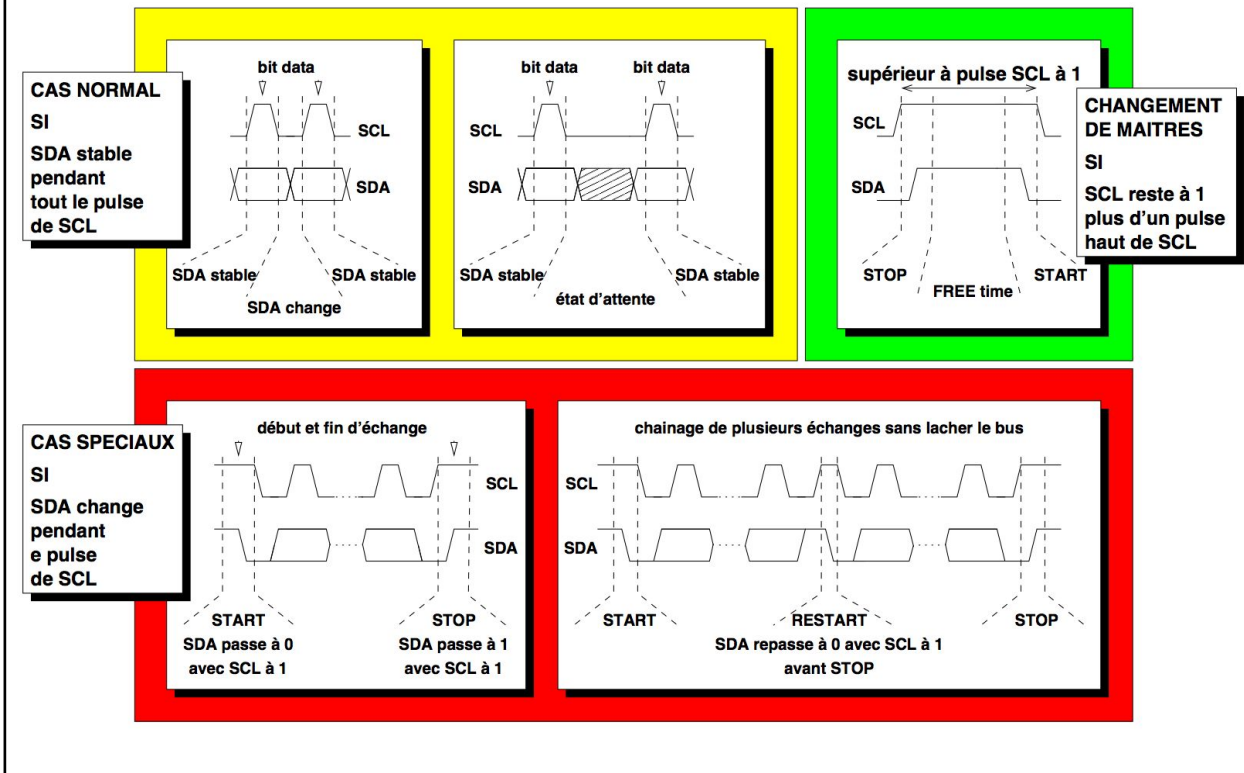
L'esclave :

- attend une condition de démarrage
- lit l'adresse sur 7 bits
- lit la commande r/w
- émet ACK si concerné
- pour une écriture, il boucle sur
  - lit les 8 bits de donnée
  - met ACK ou NACK pour arrêter
- pour une lecture, il boucle sur
  - écrit les 8 bits de donnée
  - lit l'accusé et stoppe si NACK

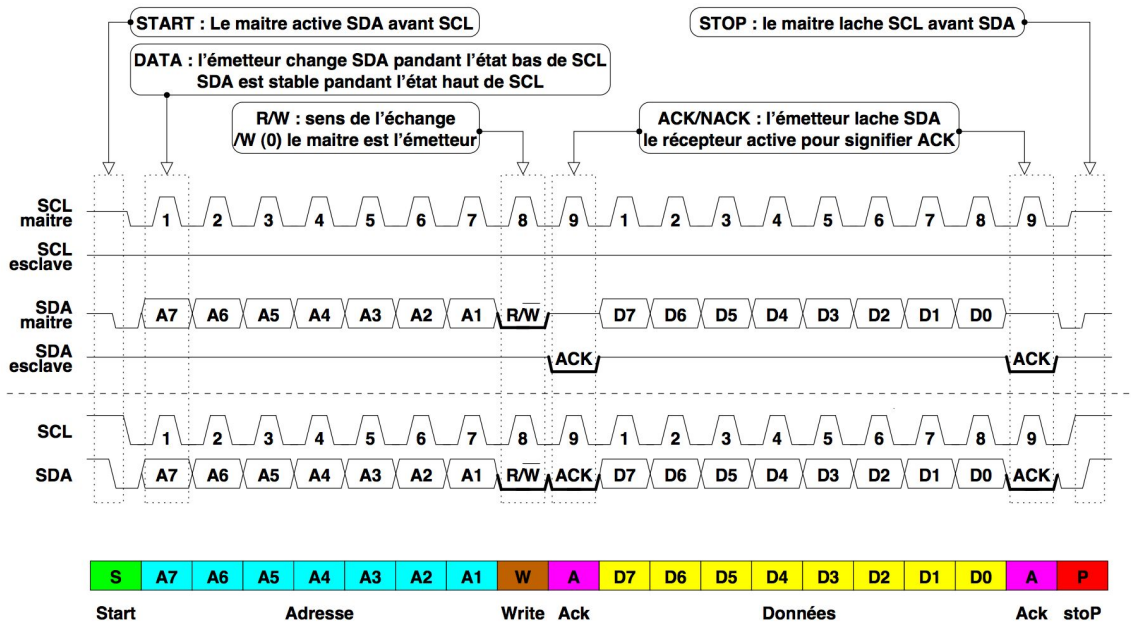
Le maître et l'esclave peuvent ralentir l'échange en jouant sur SCL

- C'est le maître qui pilote l'horloge SCL en la mettant à 0 et à 1
- Si l'esclave force l'horloge à 0, le maître ne peut plus la mettre à 1  
Le maître comprend qu'il doit attendre que l'esclave "relâche" l'horloge

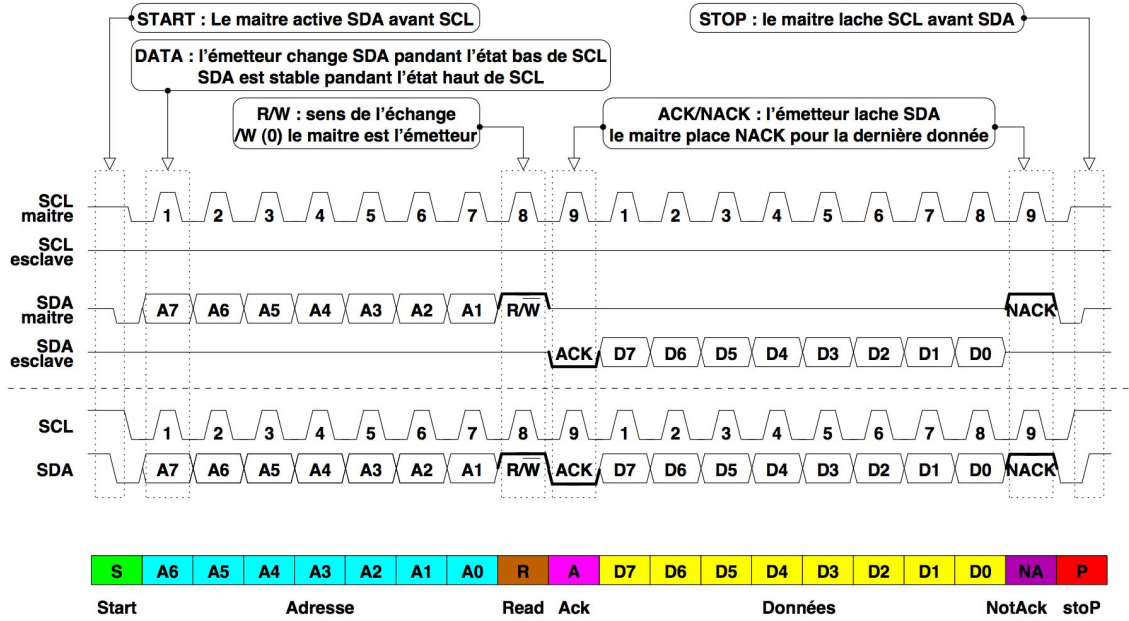
# Etat des lignes SDA et SCL



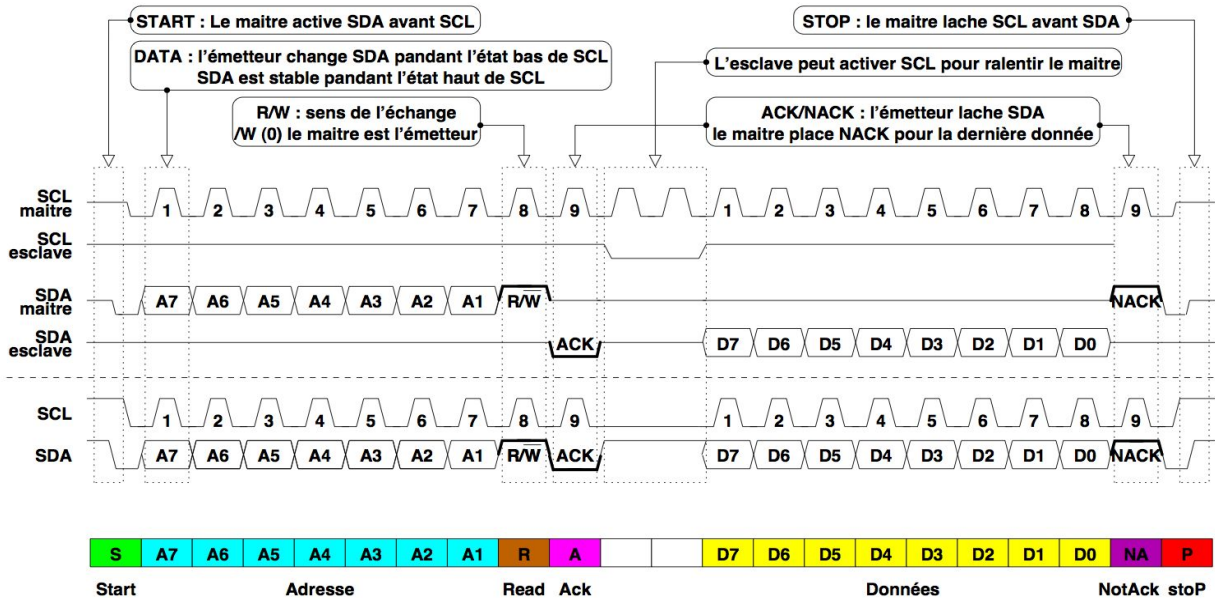
# I2C trame de base : écriture d'un octet



# I2C trame de base : lecture d'un octet



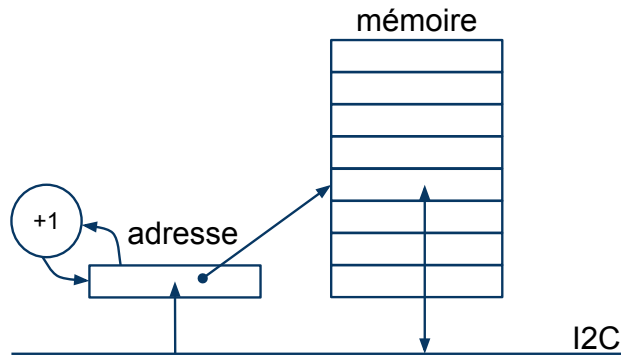
# I2C trame de base : état d'attente





# I2C écriture d'une mémoire

- Un abonné I2C dispose d'une adresse sur le bus (numéro d'abonné)
- Dans le cas général un abonné contient de la mémoire adressable.
- La manière de lire ou d'écrire la mémoire interne d'un abonné est propre à l'abonné
- Le principe général est le suivant



## Pour une écriture

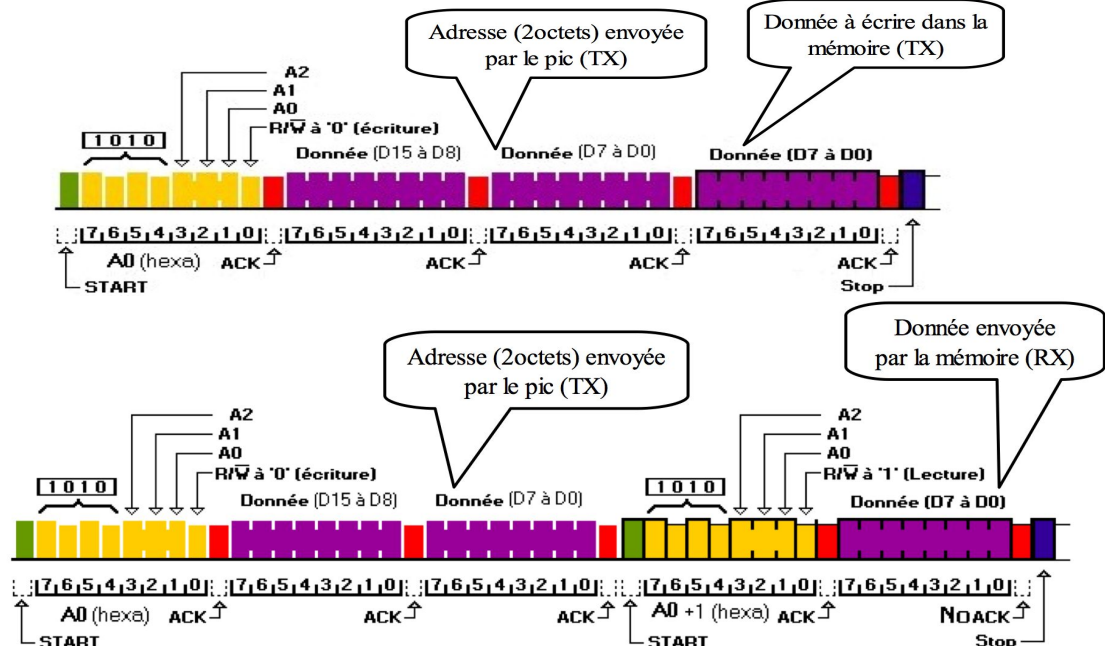
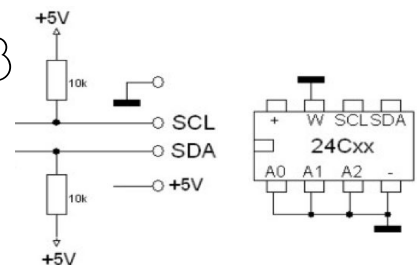
- la première écriture est faite dans un registre d'adresse
- Les écritures suivantes sont faites dans la mémoire aux adresses pointées par le registre d'adresse avec auto incrément ou pas

## Pour une lecture

- on commence par faire une écriture ... forcément l'adresse
- on fait ensuite des lectures qui lisent forcément la mémoire, et donc pas le registre d'adresse, avec auto incrément ou pas

# I2C exemple 24C3

Mémoire flash de 32kb 4ko  
 A[2:0] = 000 => répond à l'adresse 10100000 (A0) pour une écriture  
 10100001 (A1) pour une lecture

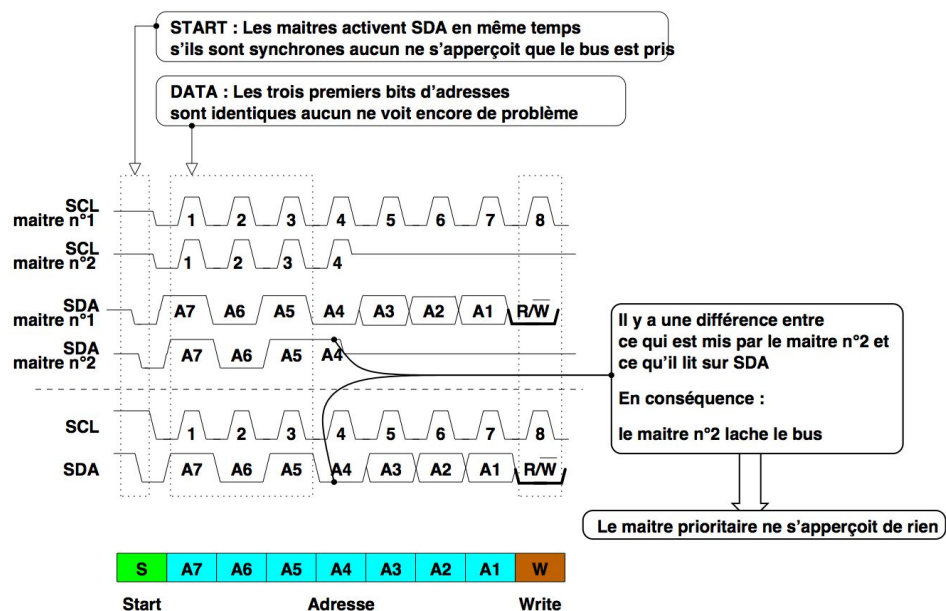


# Arbitrage entre maîtres

- Quand un maître adresse un esclave il place l'adresse de celui-ci sur SDA.
  - Pour mettre un 0 sur SDA, donc il active le transistor de pull-down
  - Pour mettre un 1 sur SDA, donc il utilise le pull-up de la ligne.
- Quand un maître écrit sur SDA, il vérifie en la relisant SDA
- Si SDA vaut 0 alors que le maître n'a pas activé son pull-down alors c'est qu'un autre maître communique aussi
- Le perdant se retire aussitôt
- Comme les adresses sont données avec les bits de poids fort d'abord, les adresses d'esclaves les plus petites sont prioritaires par rapport aux grandes

# Arbitrage entre maîtres

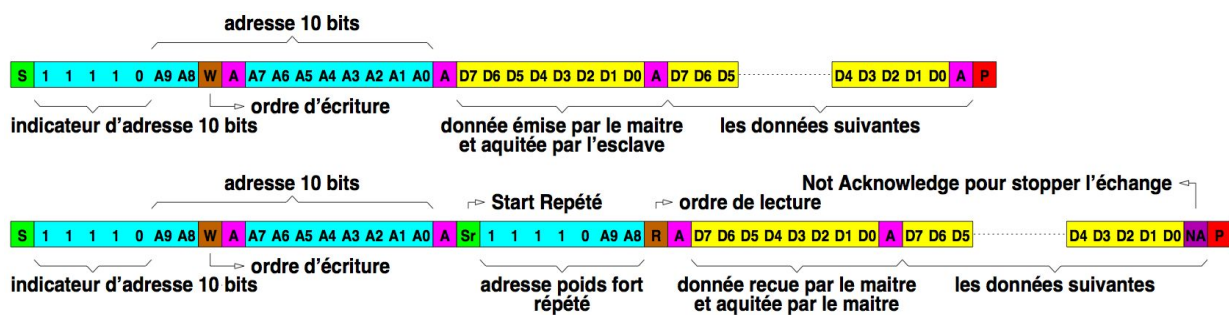
Si deux maîtres tentent de démarrer un échange simultanément :  
le premier qui dit 1 sur SDA a perdu.



# I2C adressage 10 bits

Passage à 10 bits d'adresse, il y a deux cas de figure :

1. Le maître veut faire une écriture :  
il envoie l'adresse de 10 bits dans les deux premiers octets puis il envoie les données normalement.
2. Le maître veut faire une lecture :  
il envoie l'adresse de 10 bits en indiquant qu'il s'agit d'une écriture puis il envoie à nouveau le poids fort de l'adresse en indiquant qu'il s'agit d'une lecture, seul l'esclave qui s'était reconnu répond.



# I2C Arduino : Wire Library

## Pins

- UNO A4 (SDA), A5 (SCL)
  - Mega2560 20 (SDA), 21 (SCL)
- } Le choix est imposé par le microcontrôleur

## API

### INITIALISATION

- `begin()` initialise communication avec Arduino "maître"
- `begin(address)` initialise communication avec Arduino "esclave"

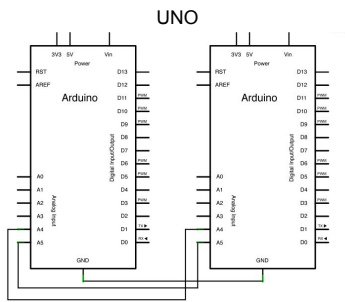
### MODE MAÎTRE

- `beginTransaction(address)` débute communication avec un esclave
- `endTransmission()` envoie des données vers esclave, rend 0 si succès
- `write(<value|string|pointer, size>)` écrit les données à envoyer vers esclave
- `requestFrom(address, size)` demande de données à un esclave
- `available()` rend le nombre d'octets disponibles après un `requestFrom`
- `read()` lit l'octet reçu de l'esclave après un `requestFrom`

### MODE ESCLAVE

- `onReceive(function)` définit la fonction à appeler sur réception de données du maître
- `onRequest(function)` définit la fonction à appeler sur requête du maître
- `write()` envoie les données vers le maître après requête
- `available()` test si données dispo. en provenance du maître (cf `onReceive`)
- `read()` lit données en provenance maître

## Exemple 1 : <http://arduino.cc/en/Tutorial/MasterWriter>



Master Writer Code - Program for Arduino 1

```
#include <Wire.h>

void setup()
{
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop()
{
  Wire.beginTransmission(4); // transmit to device #4
  Wire.write("x is ");      // sends five bytes
  Wire.write(x);            // sends one byte
  Wire.endTransmission();   // stop transmitting

  x++;
  delay(500);
}
```

```
#include <Wire.h>

void setup()
{
  Wire.begin(4); // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600); // start serial for output
}

void loop()
{
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  while(1 < Wire.available()) // loop through all but the last
  {
    char c = Wire.read(); // receive byte as a character
    Serial.print(c);      // print the character
  }
  int x = Wire.read();    // receive byte as an integer
  Serial.println(x);     // print the integer
}
```

Slave Receiver Code - Program for Arduino 2

// by Nicholas Zambetti <<http://www.zambetti.com>>

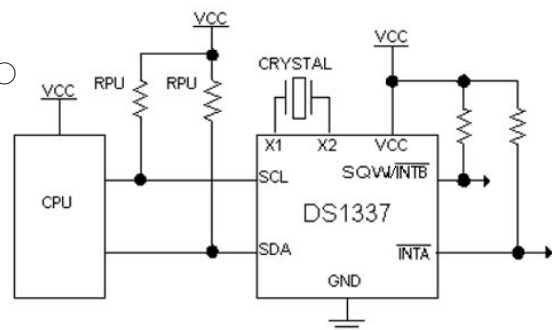
## Exemple 2 : Horloge temps réel [ds1337](http://www.st.com/~/media/DS1337)

The DS1337 serial real-time clock is a low-power clock/calendar with two programmable time-of-day alarms and a programmable square-wave output. Address and data are transferred serially through an I<sup>2</sup>C bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator.

The device is fully accessible through the serial interface while VCC is between 1.8V and 5.5V. Timekeeping operation is maintained with VCC as low as 1.3V.

### Key Features

- Real-Time Clock (RTC) Counts Seconds, Minutes, Hours, Day, Date, Month, and Year with Leap-Year Compensation Valid Up to 2100
- I<sup>2</sup>C Serial Interface
- Two Time-of-Day Alarms
- Oscillator Stop Flag
- Programmable Square-Wave Output Defaults to 32kHz on Power-Up
- Available in 8-Pin DIP, SO, or  $\mu$ SOP
- -40°C to +85°C Operating Temperature Range



# Exemple 2 : Horloge temps réel [ds1337](#)

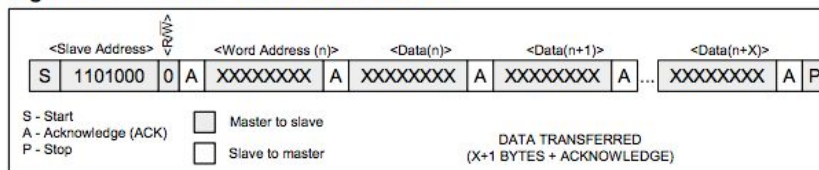
**Table 2. Timekeeper Registers**

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00H	0	10 Seconds		Seconds			Seconds	Seconds	00-59	
01H	0	10 Minutes		Minutes			Minutes	Minutes	00-59	
02H	0	12/24	AM/PM	10 Hour	Hour			Hours	1-12 + AM/PM 00-23	
03H	0	0	0	0	0	Day		Day	1-7	
04H	0	0	10 Date		Date			Date	01-31	
05H	Century	0	0	10 Month	Month			Month/ Century	01-12 + Century	
06H	10 Year				Year			Year	00-99	
07H	A1M1	10 Seconds		Seconds			Alarm 1 Seconds	Alarm 1 Seconds	00-59	
08H	A1M2	10 Minutes		Minutes			Alarm 1 Minutes	Alarm 1 Minutes	00-59	
09H	A1M3	12/24	AM/PM	10 Hour	Hour			Alarm 1 Hours	1-12 + AM/PM 00-23	
0AH	A1M4	DY/DT	10 Date		Day			Alarm 1 Day	1-7	
					Date			Alarm 1 Date	01-31	
0BH	A2M2	10 Minutes		Minutes			Alarm 2 Minutes	Alarm 2 Minutes	00-59	
0CH	A2M3	12/24	AM/PM	10 Hour	Hour			Alarm 2 Hours	1-12 + AM/PM 00-23	
0DH	A2M4	DY/DT	10 Date		Day			Alarm 2 Day	1-7	
					Date			Alarm 2 Date	01-31	
0EH	EOSC	0	0	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0FH	OSF	0	0	0	0	0	A2F	A1F	Status	—

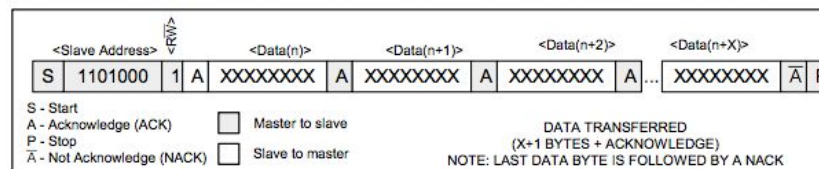
**Note:** Unless otherwise specified, the state of the registers is not defined when power is first applied or V<sub>CC</sub> falls below the V<sub>OSC</sub>.

# Exemple 2 : Horloge temps réel [ds1337](#)

**Figure 3. Data Write—Slave Receiver Mode**



**Figure 4. Data Read (from Current Pointer Location)—Slave Transmitter Mode**



**Figure 5. Data Read (Write Pointer, Then Read)—Slave Receive and Transmit**

