Informatique embarquée 4/16

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

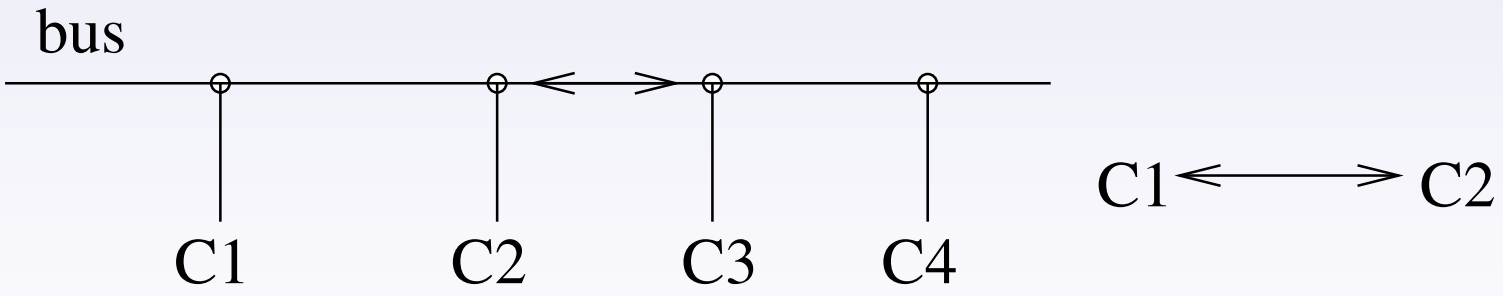
transparents à `jmfriedt.free.fr`

11 janvier 2018

Les bus de communications

Communication *sérielle* des données : plus rapide et moins coûteux (SATA v.s PATA)

- Notions de bus synchrones (partage de l'horloge) et asynchrone (chaque interlocuteur a sa propre horloge).
- synchrone est généralement plus rapide mais nécessite un signal additionnel pour l'horloge (sauf codage Manchester – ethernet)
- asynchrone nécessite l'accord préalable de tous les interlocuteurs sur le débit de communication (*baudrate*)
- très utilisé comme bus local entre un microcontrôleur et ses périphériques (peu de fils = réduction des coûts de câblage)



Les protocoles synchrones

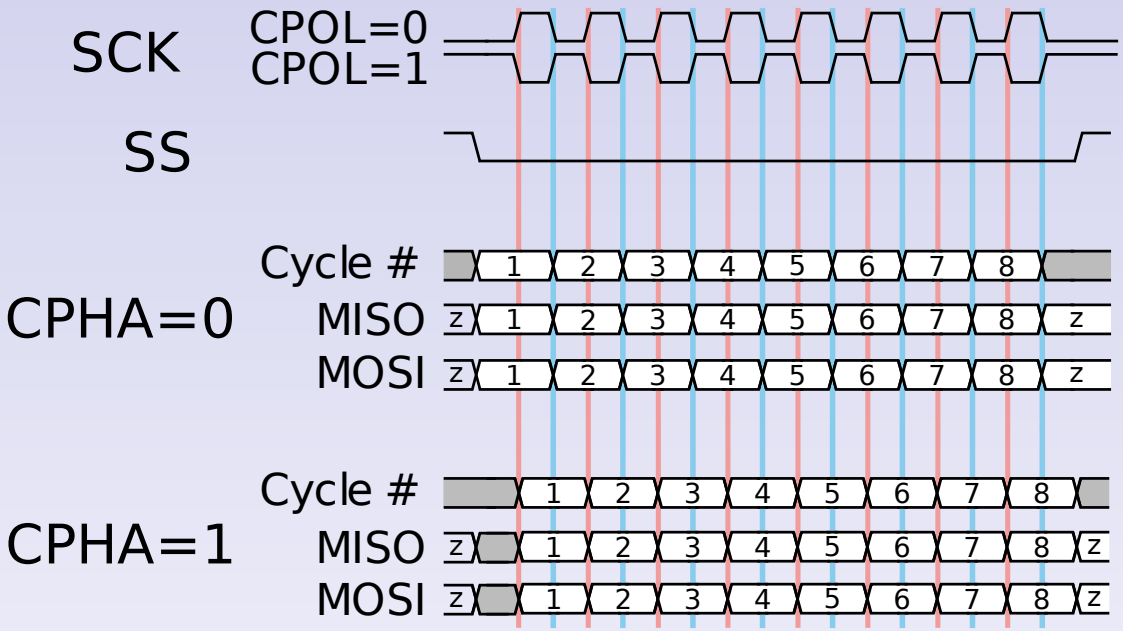


FIGURE – Chronogramme SPI (`http://en.wikipedia.org/wiki/Image:SPI_timing_diagram.svg`). Protocole synchrone : l'horloge est explicitement distribuée sur le bus.

- bus asymétrique : maître $\neq$ esclave, CS# désigne le(s) destinataire(s)
- version 2 fils d'un protocole synchrone : I^2C (bus de données est bidirectionnel).

Protocole synchrone (I^2C, SPI)

Accès matériel : degrés de liberté (état au repos de l'horloge, front d'échantillonnage, ordre des bits

Imposé dans I^2C, libre dans SPI (CPHA, CPOL)

SPI sélectionne par CS#, I^2C par l'*adresse* du destinataire (imposée par le matériel). I^2C : MSB first $\neq$ SPI : LSB ou MSB first. **I^2C @ ?**

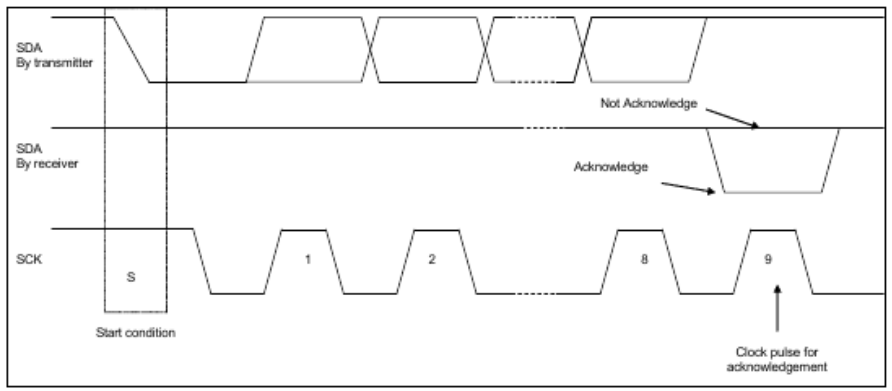


Figure 20: Waveform diagram for I²C acknowledgement on SDA

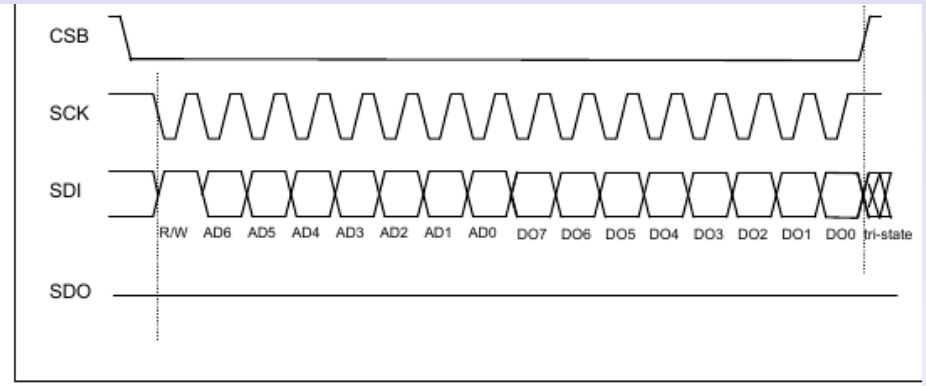
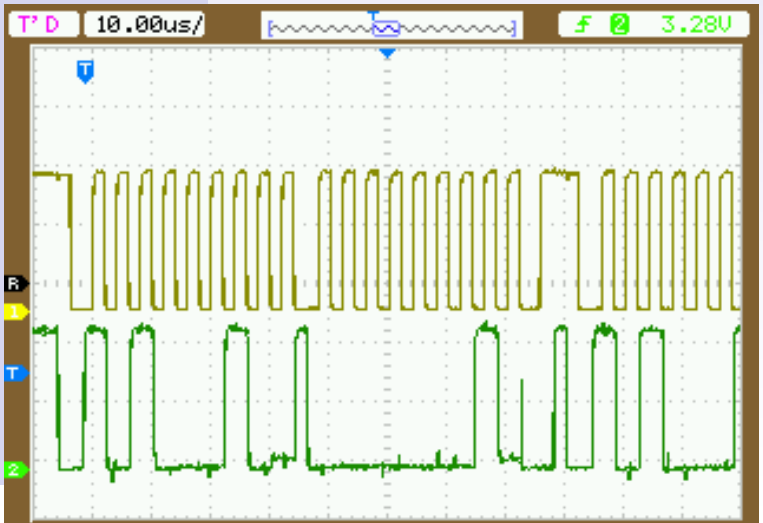
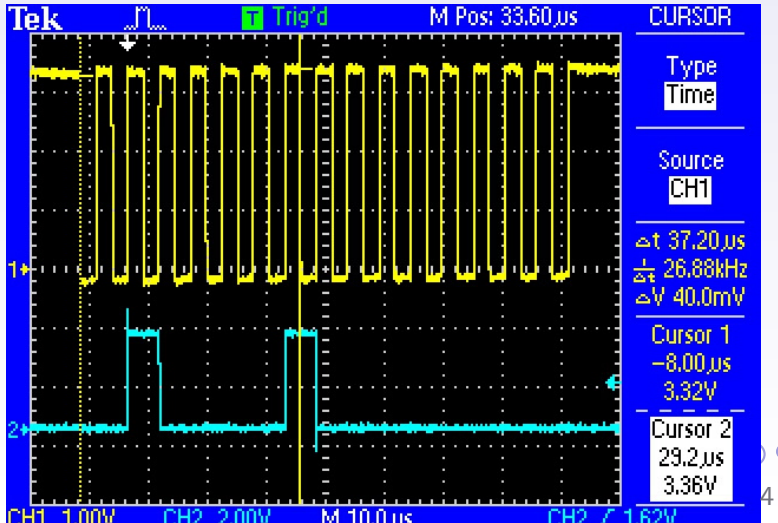


Figure 18: 3-wire SPI read protocol





Protocole synchrone (I^2C, SPI)

Accès matériel : degrés de liberté (état au repos de l'horloge, front d'échantillonnage, ordre des bits

Imposé dans I^2C, libre dans SPI (CPHA, CPOL)

SPI sélectionne par CS#, I^2C par l'*adresse* du destinataire (imposée par le matériel). I^2C : MSB first $\neq$ SPI : LSB ou MSB first. **I^2C @ ?**

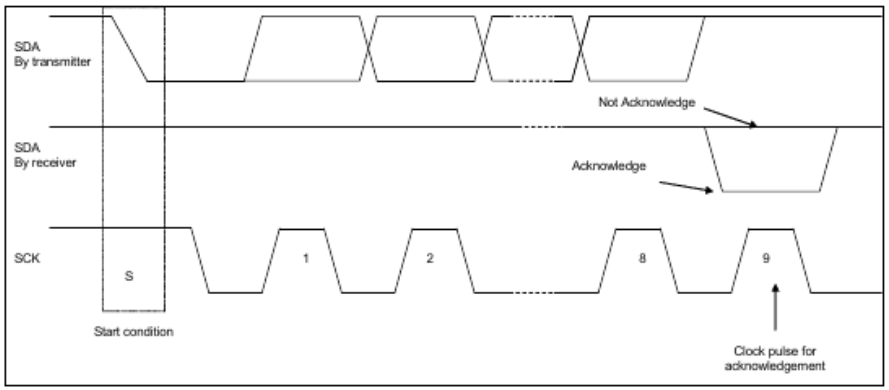


Figure 20: Waveform diagram for I²C acknowledgement on SDA

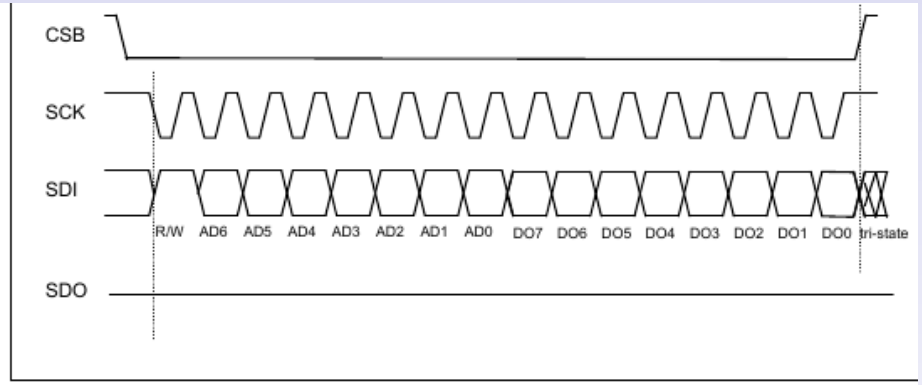
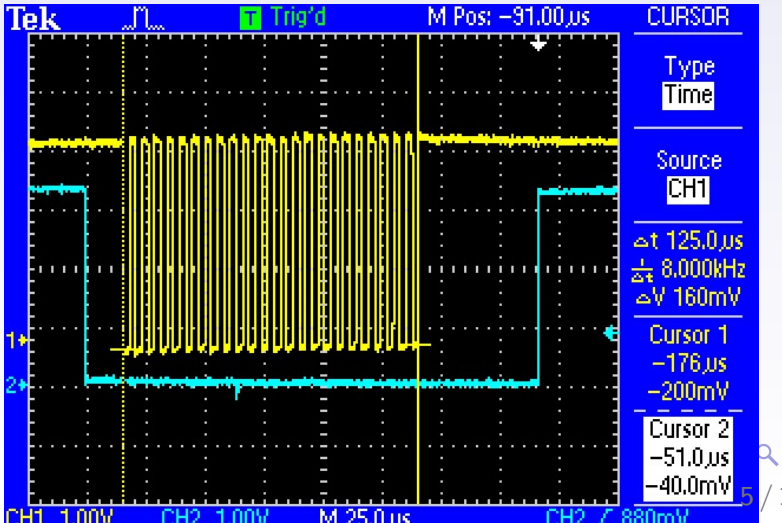


Figure 18: 3-wire SPI read protocol



Attention aux bibliothèques

PCF8563

Real-time clock/calendar

Rev. 10 — 3 April 2012 Product data sheet

1. General description

The PCF8563 is a CMOS[1] Real-Time Clock (RTC) and calendar optimized for low power consumption. A programmable clock output, interrupt output, and voltage-low detector are also provided. All addresses and data are transferred serially via a two-line bidirectional I²C-bus. Maximum bus speed is 400 kbit/s. The register address is incremented automatically after each written or read data byte.

2. Features and benefits

- Provides year, month, day, weekday, hours, minutes, and seconds based on a 32.768 kHz quartz crystal
- Century flag
- Clock operating voltage: 1.0 V to 5.5 V at room temperature
- Low backup current; typical 0.25 µA at V_{DD} = 3.0 V and T_{amb} = 25 °C
- 400 kHz two-wire I²C-bus interface (at V_{DD} = 1.8 V to 5.5 V)
- Programmable clock output for peripheral devices (32.768 kHz, 1.024 kHz, 32 Hz, and 1 Hz)
- Alarm and timer functions
- Integrated oscillator capacitor
- Internal Power-On Reset (POR)
- I²C-bus slave address: read A3h and write A2h
- Open-drain interrupt pin

Datasheet : adresses 0xA2, 0xA3 mais appel à

```
int8_t I2C_write(const uint8_t address, →
      ↪uint8_t *data, uint8_t n)
{int8_t d;
 int i;
 d = I2C_start();
 if((d != 0x08) && (d != 0x10)){
   I2C_stop();return d;
  }
  TWDR=address; // send the slave address
  if ((d=I2C_action(0)) != 0x18){
     I2C_stop(); return d;
  }
  for(i=0; i<n; i++){
    TWDR=(*data++);
    if ((d=I2C_action(0)) != 0x28){
      I2C_stop(); return d;
    }
  }
  I2C_stop();
  return 0;
}

[...]
#define PCF8563            0xA2

I2C_write(PCF8563,(uint8_t *)time,8);
```

pour communiquer (0xA2 ≠0x51 !)

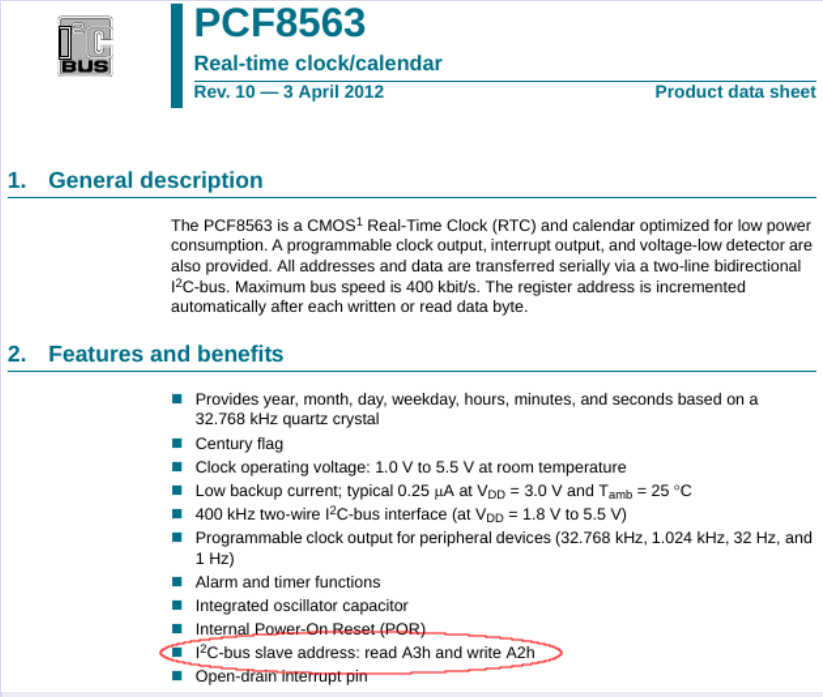
Exemple de code Arduino à [1]

```
#include "Wire.h"
#define PCF8563address 0x51
[...]

void setPCF8563()
// this sets the time and date to the →
      ↪PCF8563
{
  Wire.beginTransmission(PCF8563address);
  Wire.write(0x02);
[...]
```

[1] http://tronixstuff.com/2013/08/13/tutorial-arduino-and-pcf8563-real-time-clock-ic/

Attention aux bibliothèques



PCF8563

Real-time clock/calendar

Rev. 10 — 3 April 2012 Product data sheet

1. General description

The PCF8563 is a CMOS[1] Real-Time Clock (RTC) and calendar optimized for low power consumption. A programmable clock output, interrupt output, and voltage-low detector are also provided. All addresses and data are transferred serially via a two-line bidirectional I²C-bus. Maximum bus speed is 400 kbit/s. The register address is incremented automatically after each written or read data byte.

2. Features and benefits

- Provides year, month, day, weekday, hours, minutes, and seconds based on a 32.768 kHz quartz crystal
- Century flag
- Clock operating voltage: 1.0 V to 5.5 V at room temperature
- Low backup current; typical 0.25 μA at V_{DD} = 3.0 V and T_{amb} = 25 °C
- 400 kHz two-wire I²C-bus interface (at V_{DD} = 1.8 V to 5.5 V)
- Programmable clock output for peripheral devices (32.768 kHz, 1.024 kHz, 32 Hz, and 1 Hz)
- Alarm and timer functions
- Integrated oscillator capacitor
- Internal Power-On Reset (POR)
- I²C-bus slave address: read A3h and write A2h
- Open-drain interrupt pin

Datasheet : adresses 0xA2, 0xA3

1. 0x51=0xA2 >>1
2. méthode Wire de Arduino [1] appelle `beginTransmission` qui appelle `twi_writeTo(txAddress ...)`
3. `twi_writeTo` se trouve (par exemple) à [2] :

```
twi_slarw = TW_WRITE;
twi_slarw |= address << 1;
```

⇒ bibliothèque appelée par Arduino fait un décallage et ajoute 1 si lecture [3]

Opensource : capacité à suivre le cheminement des traitements

[1] `github.com/esp8266/Arduino/blob/master/libraries/Wire/Wire.cpp`

[2] `www.libelium.com/v11-files/api/waspmote/d5/d8c/twi_8c_source.html`

[3] `robotika.yweb.sk/skola/!Diplomovka/avr498/avr498/doc/twi__lib_8h.html`

dit que `TWI_READ` vaut 1 et `TWI_WRITE` vaut 0

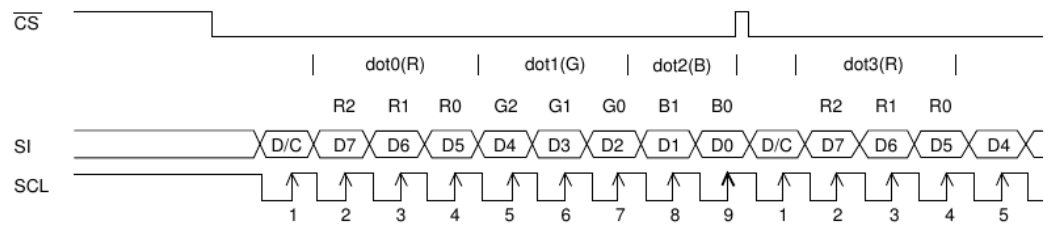
Émulation logicielle du protocole synchrone (SPI)

Pourquoi ?

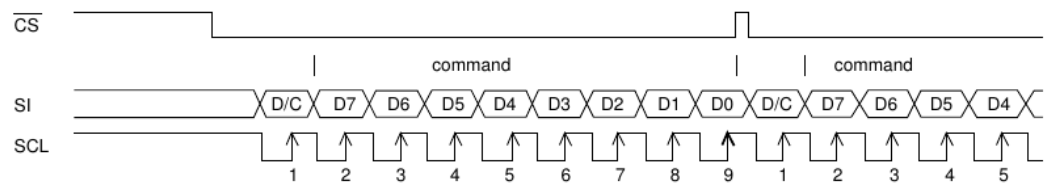
- implémenter un mode non-supporté par le matériel (e.g. 9 bits/message)
- manque de ressource matérielle

S1D15G10D08B000

(2) 9-bit serial interface
When entering data (parameters): SI = HIGH at the rising edge of the 1st SCL.

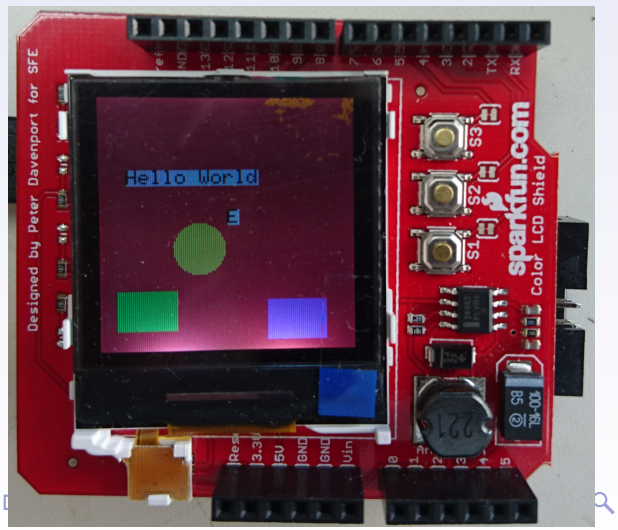


When entering commands: SI = LOW at the rising edge of the 1st SCL.



* If $\overline{CS}$ is caused to HIGH before 8 bits from D7 to D0 are entered, the data concerned is invalidated. Before entering succeeding sets of data, you must correctly input the data concerned again.
* In order to avoid data transfer error due to incoming noise, it is recommended to set $\overline{CS}$ at HIGH on byte basis to initialize the serial-to-parallel conversion counter and the register.

Exercice : implémenter l'envoi d'une commande et d'une donnée à un écran LCD Nokia

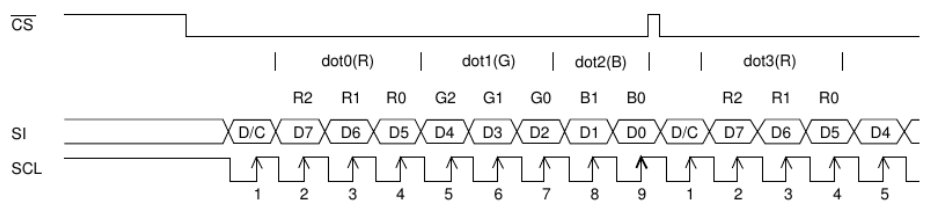


Émulation logicielle du protocole synchrone (SPI)

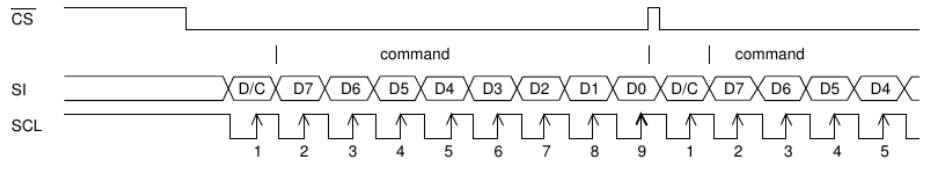
Envoi de commande/donnée à un écran LCD Nokia

S1D15G10D08B000

(2) 9-bit serial interface
When entering data (parameters): SI = HIGH at the rising edge of the 1st SCL.



When entering commands: SI = LOW at the rising edge of the 1st SCL.



* If $\overline{CS}$ is caused to HIGH before 8 bits from D7 to D0 are entered, the data concerned is invalidated. Before entering succeeding sets of data, you must correctly input the data concerned again.
* In order to avoid data transfer error due to incoming noise, it is recommended to set $\overline{CS}$ at HIGH on byte basis to initialize the serial-to-parallel conversion counter and the register.

```
#define cs_lo PORTB &= ~(1 << PORTB5)
#define cs_hi PORTB |= (1 << PORTB5)

#define mosi_hi PORTB |= (1 << PORTB7)
#define mosi_lo PORTB &= ~(1 << PORTB7)
#define ck_hi   PORTC |= (1 << PORTC7)
#define ck_lo   PORTC &= ~(1 << PORTC7)

void attend() {} //temps d'attente, sck

void sendByte(bool cmd, u8 data)
{int k;
 ck_up;
 cs_lo;
 if (cmd==0) mosi_lo; else mosi_hi;
 ck_down;attend();
 ck_up;attend();
 for (k=7;k>=0;k--)
   {if (((data>>k)&0x01) !=0) mosi_hi;
       else mosi_lo;
    ck_down;attend();
    ck_up;attend();
   }
 cs_hi;
}
```

Les protocoles asynchrones

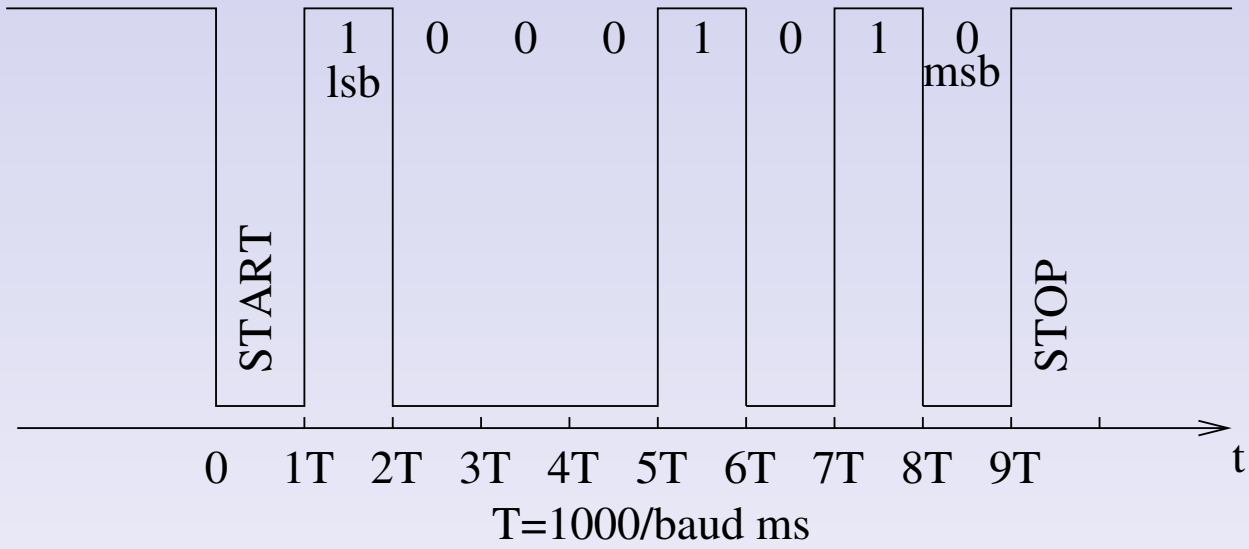


FIGURE – Chronogramme RS232. Pour 1200 bauds, $T = 833$ μs. En sortie du MAX232, les niveaux sont symétriques (± 12 V) *et inversés*. Protocole asynchrone : T doit être connu par les deux interlocuteurs.

RS232 : liaison point à point (2 interlocuteurs) [1]

1. RS232 reste le mode privilégié pour initier une connexion avec un système embarqué : *Hack All The Things : 20 Devices in 45 Minutes*, DefCon 22 (2014), à `https://www.youtube.com/watch?v=h5PRvBpLuJs`

Les protocoles asynchrones

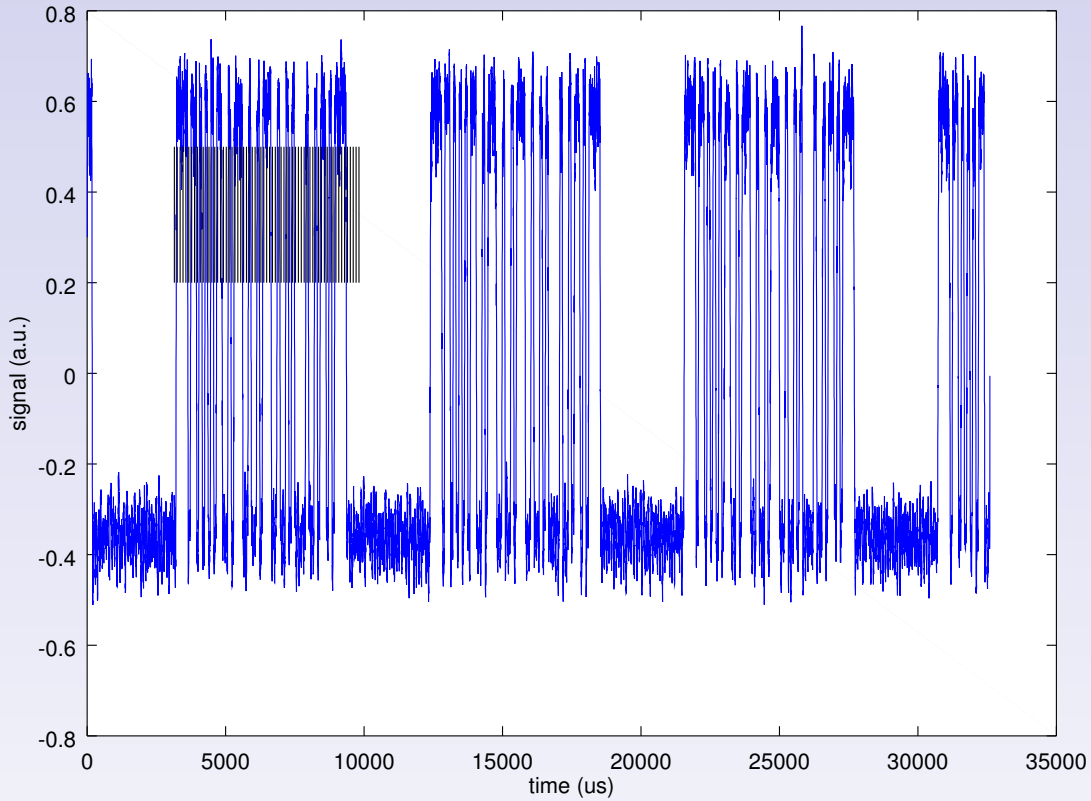


FIGURE – Chronogramme RS232. Pour 1200 bauds, $T = 833$ μs. En sortie du MAX232, les niveaux sont symétriques (± 12 V) *et inversés*. Protocole asynchrone : T doit être connu par les deux interlocuteurs.

RS232 : liaison point à point (2 interlocuteurs)

Les protocoles asynchrones

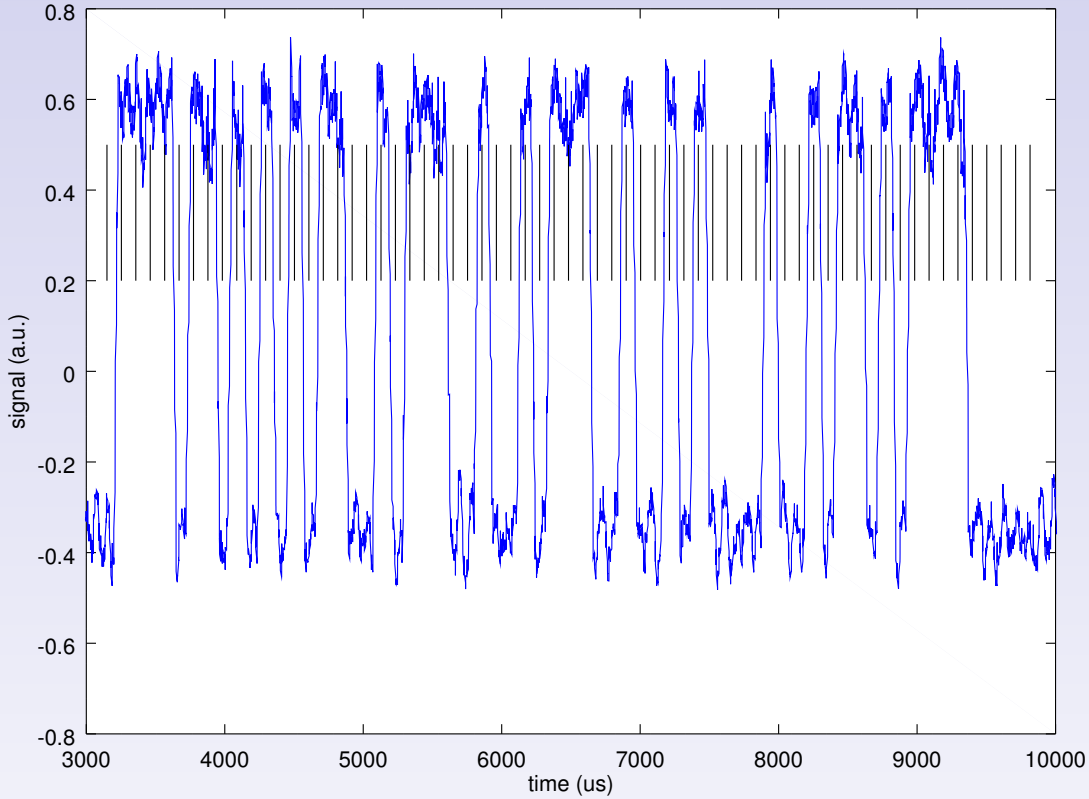


FIGURE – Chronogramme RS232. Pour 1200 bauds, $T = 833$ μs. En sortie du MAX232, les niveaux sont symétriques (± 12 V) *et inversés*. Protocole asynchrone : T doit être connu par les deux interlocuteurs.

RS232 : liaison point à point (2 interlocuteurs)

Les protocoles asynchrones

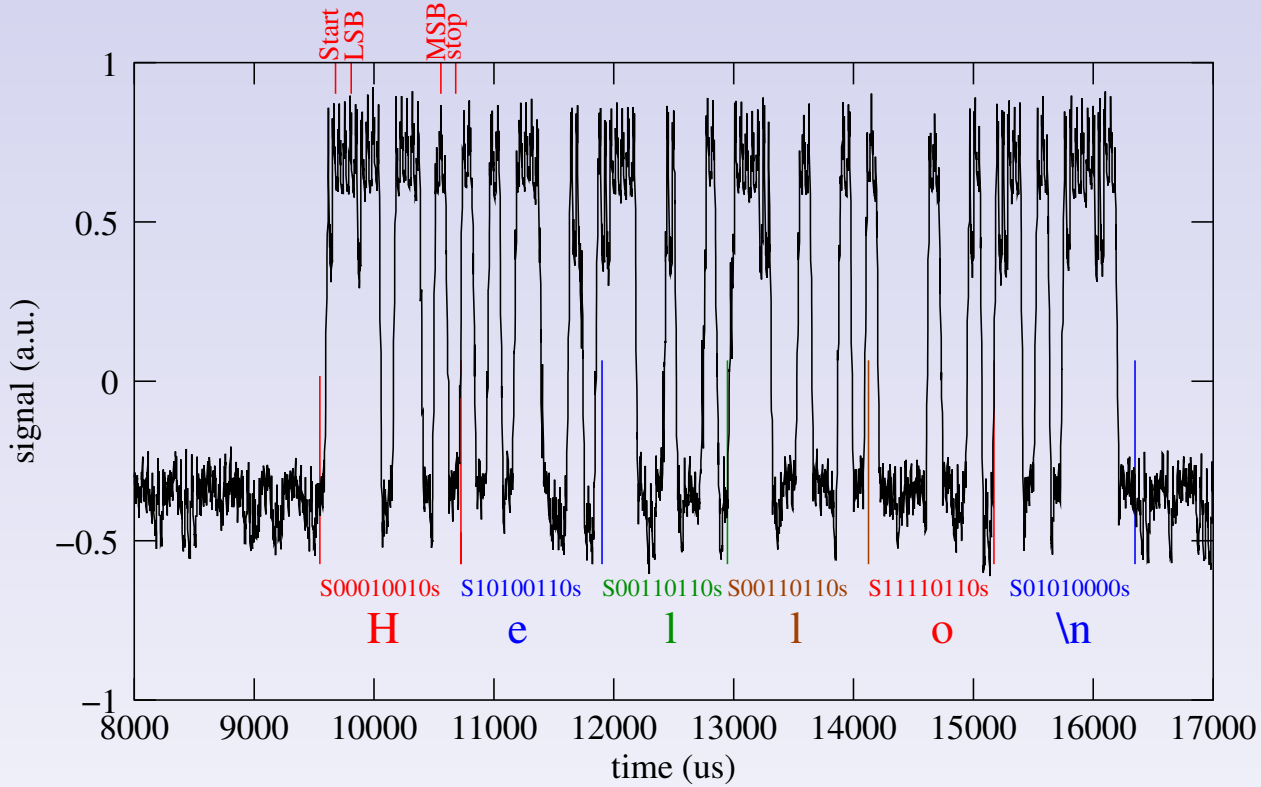


FIGURE – Chronogramme RS232. Pour 1200 bauds, $T = 833$ μs. En sortie du MAX232, les niveaux sont symétriques (± 12 V) *et inversés*. Protocole asynchrone : T doit être connu par les deux interlocuteurs.

RS232 : liaison point à point (2 interlocuteurs)

Bus locaux sous GNU/Linux

`lm-sensors` donne accès aux informations du processeur et de la carte mère

```
$ sensors
acpitz-virtual-0
Adapter: Virtual device
temp1:         +50.4C  (crit = +110.0C)
temp2:         +50.4C  (crit = +130.0C)

coretemp-isa-0000
Adapter: ISA adapter
Physical id 0:  +52.0C  (high = +87.0C, crit = +105.0C)
Core 0:         +49.0C  (high = +87.0C, crit = +105.0C)
Core 1:         +49.0C  (high = +87.0C, crit = +105.0C)
```

Démonstration avec le mise en œuvre du LM74 sur Atmega32U4.