Électronique programmable – communications

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

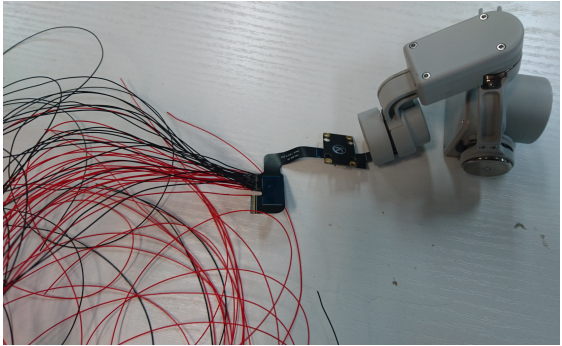
transparents à `jmfriedt.free.fr`

5 février 2024

Protocoles de communication

Protocole **parallèle** :

- un bit/fil : gourmand en ressources matérielles (bus parallèles)
- couplage entre signaux adjacents à haut débit ⇒ isolation (paires différentielles dans LVDS, masse entre les signaux dans SCSI)



← caméra drone DJI

parallèle-ATA →

`www.informit.com`

Table 7.3 shows the standard 40-pin PATA (IDE) interface connector pinout.

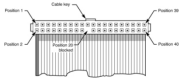


Figure 7.3. PATA (IDE) 40-pin interface connector detail.

Table 7.3. Pinout for the 40-Pin PATA Connector

Signal Name	Pin	Pin	Signal Name
-RESET	1	2	GROUND
Data Bit 7	3	4	Data Bit 8
Data Bit 6	5	6	Data Bit 9
Data Bit 5	7	8	Data Bit 10
Data Bit 4	9	10	Data Bit 11
Data Bit 3	11	12	Data Bit 12
Data Bit 2	13	14	Data Bit 13
Data Bit 1	15	16	Data Bit 14
Data Bit 0	17	18	Data Bit 15
GROUND	19	20	KEY (pin missing)

Protocoles **série** : bits transmis séquentiellement sur 1 ou 2 fil(s).

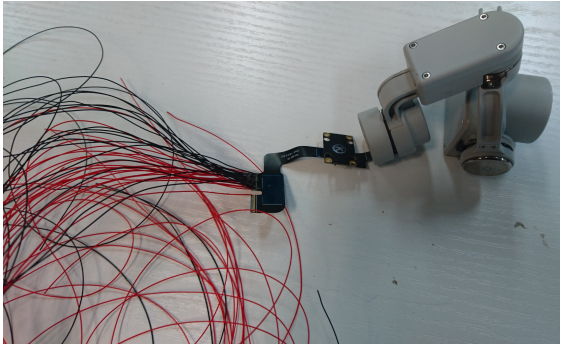
Asynchrone v.s synchrone :

- asynchrone : pas de partage d'horloge (historiquement, sur ligne téléphonique)
- synchrone : partage d'horloge ⇒ débit plus élevé

Protocoles de communication

Protocole **parallèle** :

- un bit/fil : gourmand en ressources matérielles (bus parallèles)
- couplage entre signaux adjacents à haut débit ⇒ isolation (paires différentielles dans LVDS, masse entre les signaux dans SCSI)



← caméra drone DJI

SCSI →

www.interfacebus.com

SCSI P Connector Single-Ended			
Pin	Signal Name	Pin	Signal Name
1	GND	35	D12-
2	GND	36	D13-
3	GND	37	D14-
4	GND	38	D15-
5	GND	39	DPARH-
6	GND	40	D0-
7	GND	41	D1-
8	GND	42	D2-
9	GND	43	D3-
10	GND	44	D4-
11	GND	45	D5-
12	GND	46	D6-
13	GND	47	D7-
14	GND	48	DPARITY-
15	GND	49	GND
16	GND	50	GND
17	TRMPWR	51	TRMPWR
18	TRMPWR	52	TRMPWR
19	RESERVED	53	RESERVED
20	GND	54	GND
21	GND	55	ATN-
22	GND	56	GND
23	GND	57	BSY-
24	GND	58	ACK-
25	GND	59	RST-
26	GND	60	MSG-
27	GND	61	SEL-
28	GND	62	C/D-
29	GND	63	REQ-
30	GND	64	I/O-
31	GND	65	D8-
32	GND	66	D9-
33	GND	67	D10-
34	GND	68	D11-

www.interfacebus.com

Protocoles **série** : bits transmis séquentiellement sur 1 ou 2 fil(s).

Asynchrone v.s synchrone :

- asynchrone : pas de partage d'horloge (historiquement, sur ligne téléphonique)
- synchrone : partage d'horloge ⇒ débit plus élevé

Quelques exemples de débits

Parallèle

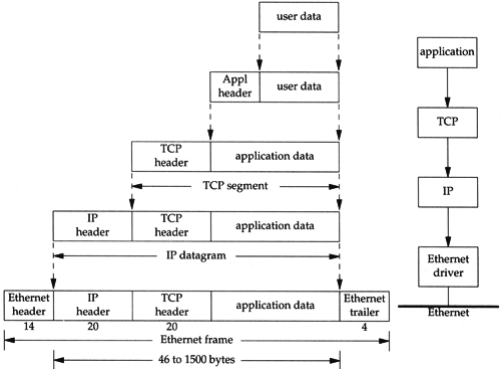
IDE/PATA (1986)	133 MB/s
SCSI (2003)	320 MB/s

Série

SATA (2000)	150–600 MB/s
USB-2	480 Mb/s=60 MB/s
USB-3	5 Gb/s=625 MB/s
Ethernet	100-1000-10000 Mb/s
	=12.5-125-1250 MB/s

1 byte (1 B)=8 bits (8 b)
1 Kb= 1024 b $\neq$ 1 kb
1 MB= $(1024)^2 = 1048576$ b $\neq 10^6$ b
1 GB= $(1024)^3 = 1.0737 \cdot 10^9$ b $\neq 10^9$ b

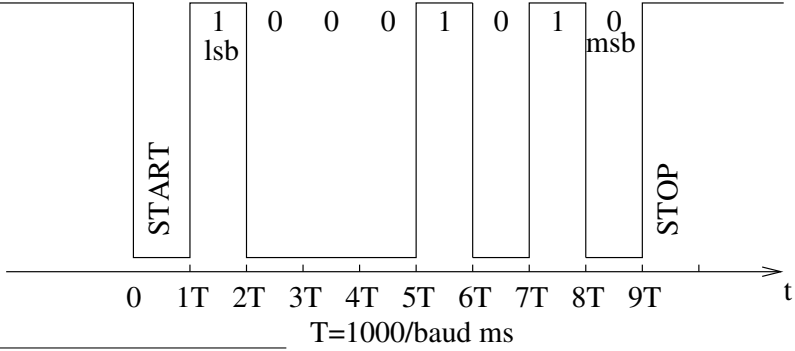
Attention aux entêtes, checksum... trame Ethernet :



https://benjamintoll.com/2018/04/29/on-tcp/ip-basics/

Protocole asynchrone (RS232) [1]

- **Pas** d'horloge commune ⇒ cadencement des données par convention (*baudrate*)
- Début : repos (1) → start bit (0)
- N bits de données, $N \in \{7, 8\}$, LSB first
- bit de parité optionnel (E, O, N)
- 1 ou 2 bit(s) de stop (1)
- **Exercice : quel est le temps de transmission d'un octet à 115200 bauds en 8N1 ?**



1. gtvhacker, *Hack All The Things : 20 Devices in 45 Minutes*, DEF CON 22 (2014) à `https://www.youtube.com/watch?v=h5PRvBpLuJs`

Protocole asynchrone (RS232) [1]

- **Pas** d'horloge commune ⇒ cadencement des données par convention (*baudrate*)
- Début : repos (1) → start bit (0)
- N bits de données, $N \in \{7, 8\}$, LSB first
- bit de parité optionnel (E, O, N)
- 1 ou 2 bit(s) de stop (1)

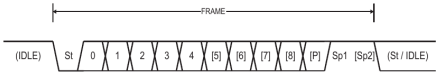
18.3 Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 18-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 18-4. Frame Formats

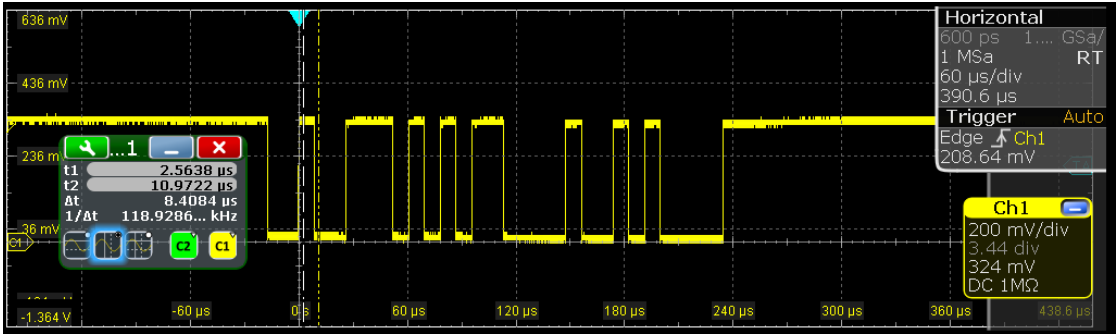


St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

1. gtvhacker, *Hack All The Things : 20 Devices in 45 Minutes*, DEF CON 22 (2014) à `https://www.youtube.com/watch?v=h5PRvBpLuJs`

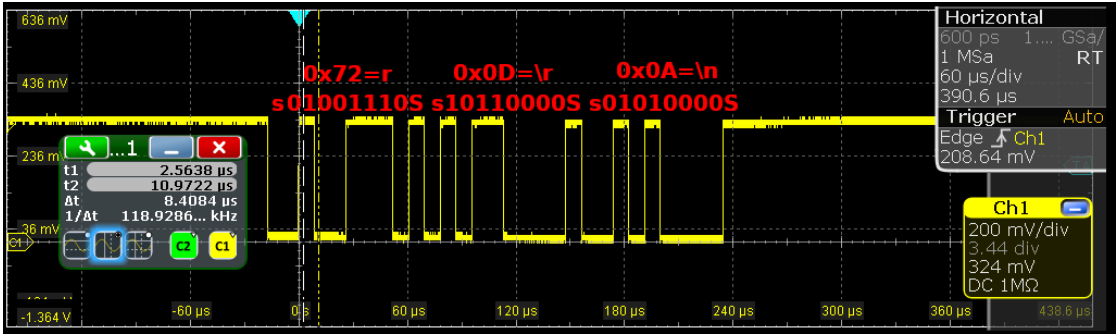
Analyse de trame RS232

Quels symboles sont transmis dans cette liaison RS232 ?



Analyse de trame

1. identifer le débit de communication (*baudrate*) comme la transition la plus courte
2. ne pas oublier les bits de début *start* ("s") et fin *stop* ("S")
3. resynchronisation sur le front descendant du bit de début
4. possibilité d'un bit de parité et de 2 bits de fin (peu utilisés)



Selection Table for Accel

https://www.analog.com/ 110% Search

Choose Parameters All | Reset Table | Maximize Filters | Sort by Newest

Save to myAnalog | Download to Excel | Share | Quick Tips | Send Feedback

Compare

Filter Parts

50 parts

Part Number	ometer Range	BW typ | Hz	Output Type	Sensing Axis
	AND	50 - 24k	OR AND	OR AND
	s Selected	HIDE	9 Values Selected	4 Values Selected
			HIDE	HIDE
ADXL316		1.6k	Analog	X, Y, Z
ADXL700		-	SPI	X, Y, Z
ADXL363	8 g	200	SPI	X, Y, Z
ADXL375		1.6k	I²C, SPI	X, Y, Z
ADXL313	g, 2 g, 4 g	1.6k	I²C, SPI	X, Y, Z
ADXL288		408	SPI	X, Y
ADXL295		408	SPI	X, Y

Synchronous protocols (I^2C, SPI)

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

- état au repos de l'horloge, front d'échantillonnage : imposés dans le protocole I^2C, libre dans SPI (CPHA, CPOL)
- ordre des bits : I^2C impose MSB en premier $\neq$ SPI : LSB ou MSB en premier
- SPI sélectionne un esclave par CS#, I^2C par l'*adresse* du destinataire (imposée par le matériel).

Exemple du XE1203F [2] : protocole proche de SPI

1 0 R/W# A4-A0 D7-D0

5 INTERFACE DEFINITION, PRINCIPLES OF OPERATION

5.1 SERIAL CONTROL INTERFACE

A 3-wire bi-directional bus (SCK, SI, SO) is used to communicate with XE1203. SCK and SI are input signals supplied externally, for example by the microcontroller. The XE1203 configures the SO signal as an output pin during read operation, and it is tri-stated in other modes. The falling edge of the SCK signal is used to sample the SI pin to write data into the internal shift register of the XE1203. The rising edge of the SCK signal is used to output data to SO pin by XE1203, so the microcontroller should sample data at the falling edge of SCK.

The signal EN must be low during the whole write and read sequences. In write mode the actual content of the configuration register is updated at the rising edge of the EN signal. Before this, the new data is stored in temporary registers whose content does not affect the transceiver settings.

The timing diagram of a write sequence is given in the figure below. The sequence is initiated when a Start condition is detected, that is when the SI signal is set to "0" during a period of SCK. The next bit is a read/write (R/W) bit which should be "0" to indicate a write operation. The next 5 bits are the address of the control register A[4:0] to be accessed, MSB first. Then, the next 8 bits are the data to be written in the register. The sequence ends with 2 stop bits set to "1". The data on SI should change at the rising edges of SCK, and is sampled at the falling edge of SCK. After the 2 stop bits, the data transfer is terminated. The SI line should be at "1" for at least one clock cycle on SCK before a new write or read sequence can start. In doing this, users can do multiple registers write without rising EN signal in between. The duty cycle of SCK must be between 40 % and 60 % and the maximum frequency of this signal is 1 MHz except when reading the RSSI output and FEI output where the maximum frequency of SCK is limited to 100 KHz. Over the operating supply and temperature range, set-up and hold time for SI on the falling edge of SCK are 200ns.

The time diagram of a read sequence is given in figure below. The sequence is initiated when a Start condition is detected, that is when the SI signal is set to "0" during a period of SCK. The next bit is a read/write (R/W) bit which should be "1" to indicate a read operation. The next 5 bits are the address of the control register A[4:0] to be accessed, MSB first. Then the data from the register are transmitted on the SO pin. The data become valid at the rising edges of SCK and should be sampled at the falling edge of SCK. After this, the data transfer is terminated. The SI line must stay high for at least one clock cycle on SCK to start a new write or read sequence. The maximum current drive on SO is 2mA @ 2.7V, the maximum load is CLop.

When the serial interface is not used for read or write operations, both SCK and SI should be set to "1". Except in read mode, SO is set to "HZ".

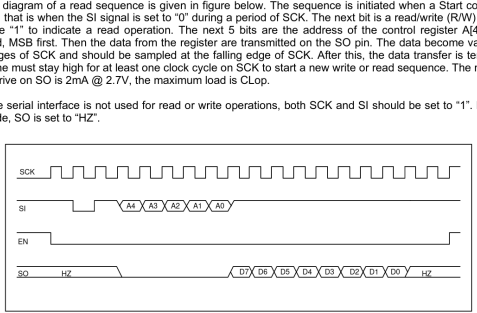
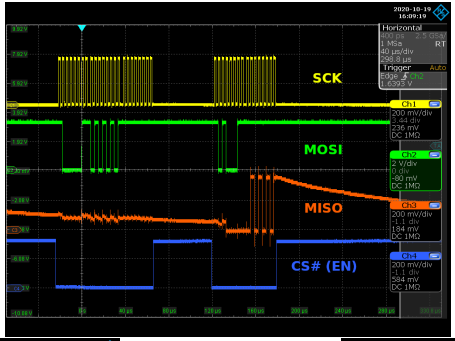
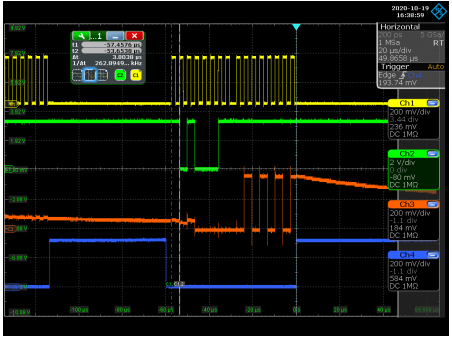


Figure 13 Read sequence of configuration register.

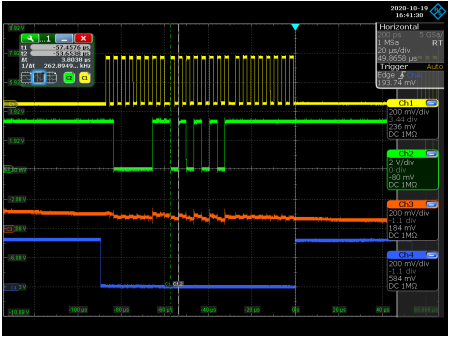
2. https://datasheet.digchip.com/531/531-00053-0-XE1203.pdf



Lecture



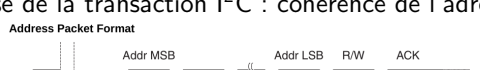
Écriture

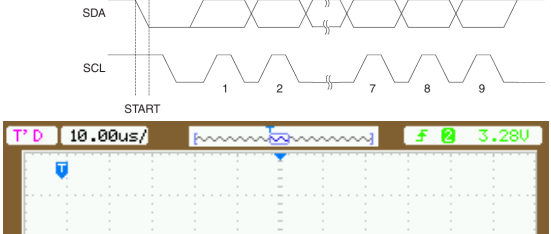


Protocoles synchrones (I^2C)

Analyse de la transaction I^2C : cohérence de l'adresse pour activer le périphérique

Figure 20-4. Address Packet Format





Penser aux résistances de tirage

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 I^2C-bus. Maximum bus speed is 400 kbit/s. The register address is incremented automatically after each written or read data byte.

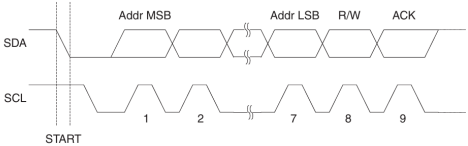
2. Features and benefits

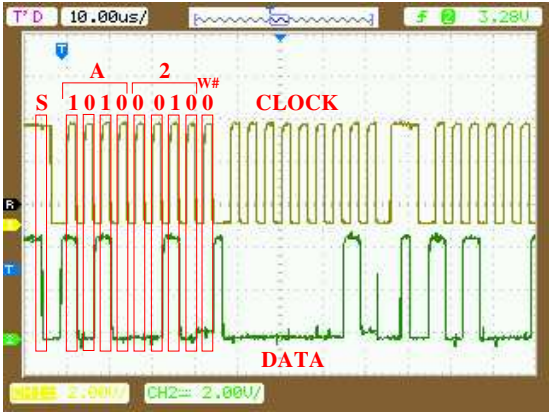
- Provides year, month, day, weekday, hours, minutes, and seconds based on a 32.768 kHz quartz crystal
- Century flag
- Clock operating voltage: 1.0 V to 5.5 V at room temperature
- Low backup current; typical 0.25 μA at V_{DD} = 3.0 V and T_{amb} = 25 °C
- 400 kHz two-wire I^2C-bus interface (at V_{DD} = 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)
- I^2C-bus slave address: read A3h and write A2h
- Open-drain interrupt pin

Protocoles synchrones (I^2C)

Analyse de la transaction I^2C : cohérence de l'adresse pour activer le périphérique

Figure 20-4. Address Packet Format





Penser aux résistances de tirage

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 I^2C-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 I^2C-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)
- I^2C-bus slave address: read A3h and write A2h
- Open-drain interrupt pin

Protocoles synchrones (SPI)

Vérification

- front d'horloge sur lequel la donnée est échantillonnée
- activation (manuelle ou automatique) du signal de sélection de l'esclave (attention aux transactions sur plusieurs octets consécutifs)

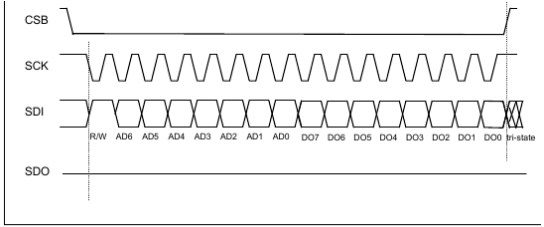
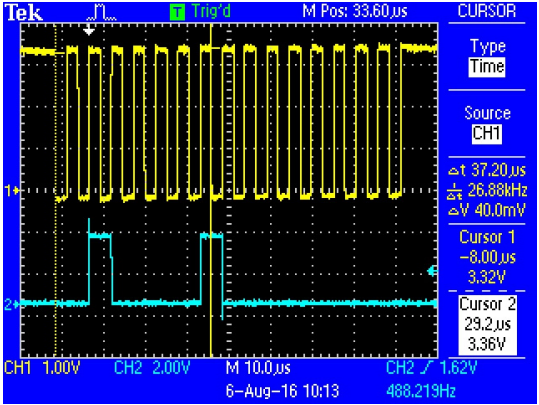


Figure 18: 3-wire SPI read protocol



Protocoles synchrones (SPI)

Vérification

- front d'horloge sur lequel la donnée est échantillonnée
- activation (manuelle ou automatique) du signal de sélection de l'esclave (attention aux transactions sur plusieurs octets consécutifs)

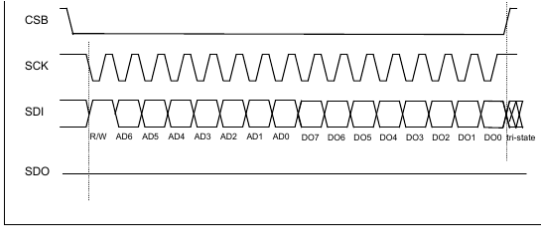
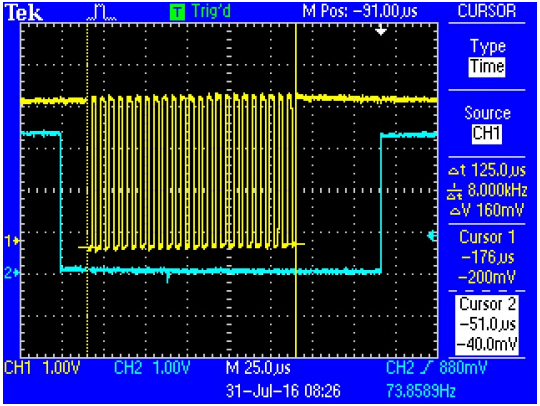


Figure 18: 3-wire SPI read protocol



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 I^2C-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 I^2C-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)
- I^2C-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()
{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 I^2C-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 I^2C-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)
- I^2C-bus slave address: read A3h and write A2h
- Open-drain interrupt pin

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]

Datasheet : adresses 0xA2, 0xA3

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

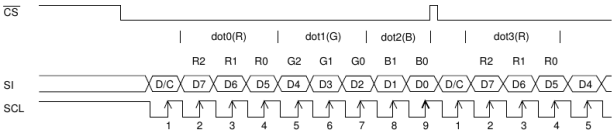
Émulation logicielle du protocole synchrone (SPI)

Pourquoi ?

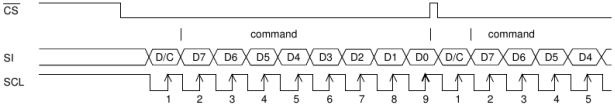
- implémenter un mode non-supporté par le matériel (e.g. 9 bits/message)
- manque de ressource matérielle

S1D15G10D08B000

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

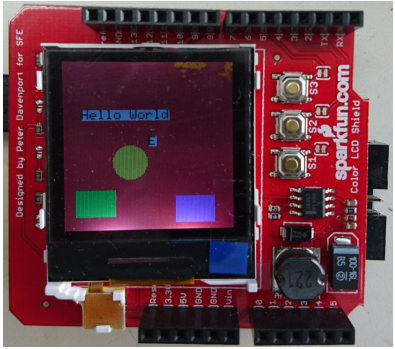


When entering commands: SI = LOW at the rising edge of the 1st SCL.



* 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

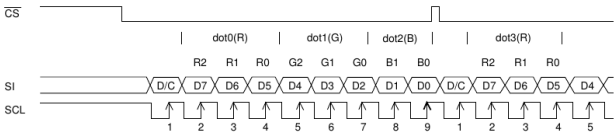


Émulation logicielle du protocole synchrone (SPI)

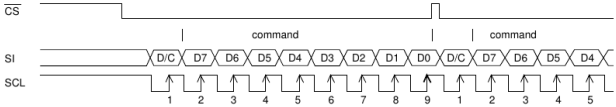
Envoi de commande/donnée à un écran LCD Nokia

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;
}
```

Initialisation du port asynchrone Atmega32U4

Table 18-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{2BAUD} - 1$

1. configurer le controle de flux
2. configurer le périphérique (8N1, 115200 bauds)

Initialisation :

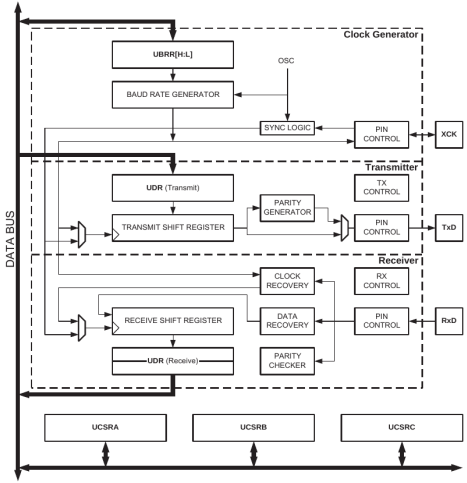
```
UCSR1A = 0;                                  // importantly U2X1 = 0
UCSR1B = 0;
UCSR1B = (1 << RXEN1)|(1 << TXEN1);  // enable receiver and transmitter
UCSR1C = _BV(UCSZ11) | _BV(UCSZ10);  // 8N1
// UCSR1D = 0;                               // no rtc/cts (probleme de version de libavr)
baud  = (((( F_CPU / ( USART_BAUDRATE * 16UL))) - 1));
UBRR1H = (unsigned char)(baud>>8);
UBRR1L = (unsigned char)baud;
```

18.5 Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the *Transmit Enable* (TXEN) bit in the UCSRnB Register. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USART and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up

Communication sur port asynchrone Atmega32U4

Figure 18-1. USART Block Diagram[1]



1. configurer le controle de flux
2. configurer le périphérique (8N1, 115200 bauds)

Communication :

```
void uart_transmit(unsigned char data )
{while (!(UCSR1A&(1<<UDRE1))) ;
 UDR1 = data;
}

unsigned char uart_receive(void)
{while (!(UCSR1A&(1<<RXC1))) ;
 return UDR1;
}
```

Note: 1. See “Pinout ATmega16U4/ATmega32U4” on page 3, Table 10-8 on page 71 and for USART pin placement.

Initialisation du port sychrone sur Atmega32U4

Table 17-5. CPOL Functionality

	Leading Edge	Trailing eDge	SPI Mode
CPOL=0, CPHA=0	Sample (Rising)	Setup (Falling)	0
CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)	1
CPOL=1, CPHA=0	Sample (Falling)	Setup (Rising)	2
CPOL=1, CPHA=1	Setup (Falling)	Sample (Rising)	3

Figure 17-3. SPI Transfer Format with CPHA = 0

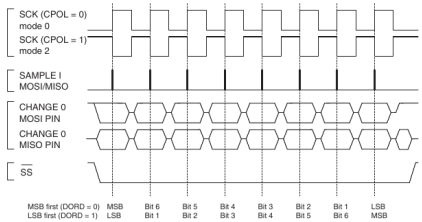
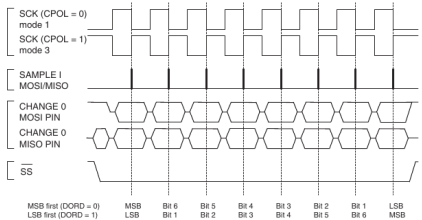


Figure 17-4. SPI Transfer Format with CPHA = 1



1. débit de communication (communication synchrone)
2. état au repos de l'horloge et front d'échantillonnage des données

SPI :

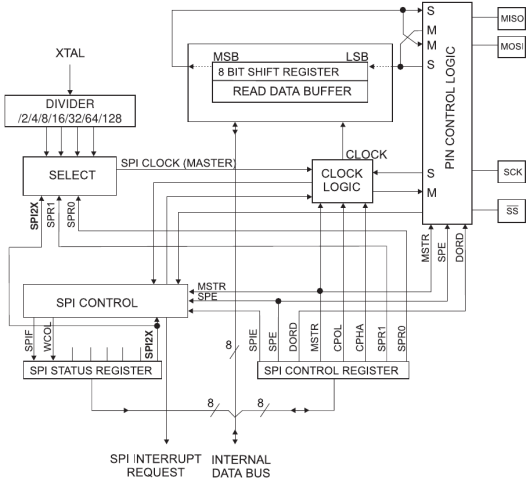
```
void init_SPI()
{// MOSI (B2), SCK (B1) sortie, MISO (B3) entree
 DDRB |= ((1<<DDB0)|(1<<DDB2)|(1<<DDB1));
 // ATTENTION : meme si CS manuel, DDB0 doit
 DDRB &= ~(1 << DDB3); // etre out (sinon bloque)
 DDRC |= (1 << DDC7);  // CS# p.17.2.1 datasheet

 SPCR=(0<<SPIE)|(1<<SPE)|(0<<DORD)|(1<<MSTR) | \
      (0<<CPOL)|(0<<CPHA)|(1<<SPR1)|(1<<SPR0);
//      Int Ena | SPI ENA|0=MSB 1st|  Master  | \
//      CK idle hi |sample trailing SCK| f_OSC/128
 SPSR &= ~(1 << SPI2X); // No doubled clock frequency
}
```

ou (I^2C) :

```
void I2C_init(uint8_t bitrate, uint8_t prescaler)
{ PRR0&=~1<<PRTWI; // internal pullups for twi
  TWBR = bitrate;
  TWSR = prescaler & 0x03;
  TWCR = 1<<TWEN ;
}
```

Communication sur port sychrone sur Atmega32U4

Figure 17-1. SPI Block Diagram[1]



Note: 1. Refer to “Pinout ATmega16U4/ATmega32U4” on page 3, and Table 10-3 on page 67 for SPI pin p

SPI :

```
char sd_raw_send_byte(char b)
{SPDR = b;          // pas de distinction TX/RX
 while(!(SPSR & (1 << SPIF))); // emet MOSI
 SPSR &= ~(1 << SPIF);
 return SPDR;                  // renvoi MISO
}
```

ou I^{2}C :

```
int8_t I2C_read(uint8_t add,uint8_t *data, →
    ↪uint8_t n)
{int ack,i;
 TWCR=(1<<TWSTA|(1<<TWINT)|(1<<TWEN));
 while(!(TWCR & (1<<TWINT)));   // init comm
 TWDR=(address|1); // read
 for(i=0; i<n; i++)
  {if (i==n-1) // ack
     {TWCR = (1<<TWEA|(1<<TWINT)|(1<<TWEN));
      while(!(TWCR & (1<<TWINT)));
     }
   else
     {TWCR = ((1<<TWINT)|(1<<TWEN)); // no ack
      while(!(TWCR & (1<<TWINT)));
     }
   *data++=TWDR;
  }                    // stop
 TWCR=((1<<TWINT)|(1<<TWSTO)|(1<<TWEN));
}
```

Émulation de la communication RS232[3] (32U4) : simavr

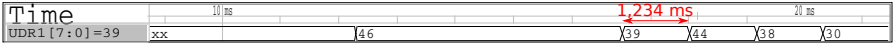
```
#include "avr_mcu_section.h"
AVR_MCU(F_CPU, "atmega32");
AVR_MCU_VCD_FILE("trace_file.vcd", 1000);
const struct avr_mmcu_vcd_trace_t _mytrace[]  _MMCU_ = {
    { AVR_MCU_VCD_SYMBOL("UDR1"), .what = (void*)&UDR1, }, };
```

pour déclarer les registres sondés (UDR1), puis

```
#define UART_BAUDRATE (57600)
//#define UART_BAUDRATE (9600)

void init_uart(void)
{unsigned short baud;
 UCSR1A = (1<<UDRE1);                    // importantly U2X1 = 0
 UCSR1B = (1 << RXEN1)|(1 << TXEN1);     // enable receiver and transmitter
 UCSR1C = (1<<UCSZ11)|(1<<UCSZ10);       // 8N1            // UCSR1D = 0; // no rtc/cts
 baud  = (((( F_CPU / ( UART_BAUDRATE * 16UL))) - 1));
 UBRR1H = (unsigned char)(baud>>8); UBRR1L = (unsigned char)baud;
}

void uart_transmit (unsigned char data)
{while (!( UCSR1A & (1<<UDRE1)));       // wait while register is free
 UDR1 = data;                            // load data in the register
}

void send_byte(unsigned char c)
{unsigned char tmp;
 tmp=c>>4;      if (tmp<10) uart_transmit(tmp+'0'); else uart_transmit(tmp+'A'-10);
 tmp=(c&0x0f); if (tmp<10) uart_transmit(tmp+'0'); else uart_transmit(tmp+'A'-10);
}
```

57600 bauds :

9600 bauds :



3. J.-M Friedt, *Développer sur microcontrôleur sans microcontrôleur : les émulateurs*, GNU/Linux Magazine France HS 103 (Juillet-Aout 2019)