TP ports de communication synchrones (2/2)

É. Carry, J.-M Friedt

13 février 2025

Questions sur ce TP :

1. Combien de fils faut-il pour relier un microcontrôleur (maître) à un périphérique (esclave) communiquant par protocole I^{2}C?
2. Combien de fils faut-il pour relier un microcontrôleur (maître) à trois périphériques (esclaves) communiquant par protocole I^{2}C?
3. Combien de fils faut-il pour relier un microcontrôleur (maître) à trente périphériques (esclaves) communiquant par protocole I^{2}C? Est-ce possible avec l'Atmega32U4?
4. Est-il possible de connecter 30 capteurs de température LM75 sur un même bus I^{2}C? justifier
5. Quel est l'ordre des bits transmis sur une liaison I^{2}C?
6. Identifier l'adresse I^{2}C que nous transmettrons pour communiquer avec un capteur de température LM75.
7. Identifier l'adresse I^{2}C que nous transmettrons pour communiquer avec l'horloge temps-réel PCF8563 : où trouver cette information? Cette valeur est-elle cohérente avec la documentation technique? Justifier.
8. un code source Arduino, tel que celui fourni par exemple à l'adresse `http://tronixstuff.com/2013/08/13/tutorial-arduino-and-pcf8563-real-time-clock-ic/`, pour accéder à ce périphérique utilise une autre adresse : laquelle?
9. Justifier cette différence, en suivant les appels de fonctions et en identifiant l'origine de la différence.
10. Quelle spécificité sur l'impédance du bus I^{2}C le différencie du bus SPI : quel composant ajouter sur le signal de données dans le contexte de son utilisation bidirectionnelle?

1 Généralités sur les ports de communication synchrones SPI et I^{2}C

Les ports de communication synchrones permettent d'échanger des informations entre circuits physiquement distincts et fournissant des fonctionnalités complémentaires. Le mode le plus simple de communication est la liaison parallèle, dans laquelle N fils (un bus de N bits de largeur) portent les signaux pour échanger des informations sur N bits. Cependant, cette approche est gourmande en surface de circuit imprimé, en broches de circuits intégrés, et en cuivre de par la multiplicité des fils pour une liaison à longue distance. Par ailleurs, son débit est limité par le couplage capacitif entre pistes adjacentes. Ce mode de communication est donc souvent remplacé par une liaison série, dans laquelle les bits ne circulent pas en parallèle sur N fils mais sont séquencés dans le temps. Ces N intervalles de temps doivent donc être connus entre l'émetteur et le récepteur : dans le cas de l'UART il s'agissait d'un accord *a priori* (*baudrate*) entre les deux interlocuteurs dans le contexte d'une liaison asynchrone. Ici, nous nous intéresserons à deux protocoles largement déployés qui partagent l'horloge entre interlocuteurs (protocole synchrone) : SPI et I^{2}C. Le premier sépare les fils portant les signaux allant du maître à l'esclave et informe l'interlocuteur concerné par le message par un signal de *Chip Select*, le second ne fournit qu 'un fil bidirectionnel, faisant circuler adresse puis données.

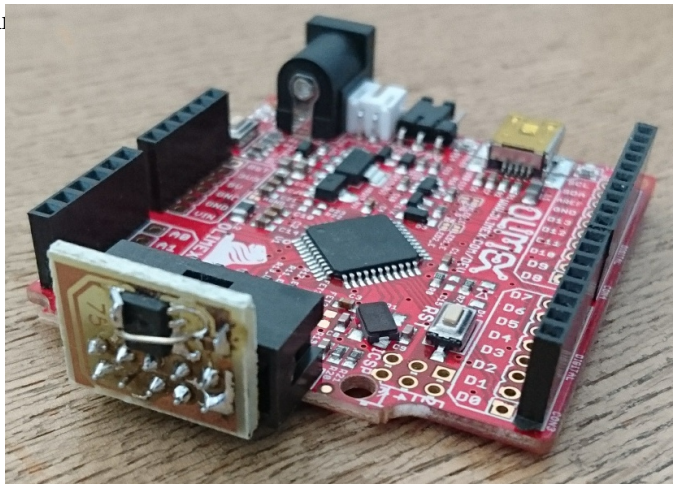


FIGURE 1: Sens de connexion de la carte munie du capteur de température LM75 communiquant par I^{2}C sur le port UEXT.

2 Le connecteur UEXT

Les signaux qui nous intéressent sont tous routés vers le connecteur HE10 en bout de carte, nommé UEXT (Fig. 1). Le schéma ci-dessous en fournit le brochage. Noter en particulier la possibilité de commuter l'alimentation du connecteur au travers d'un transistor, en vue de réduire la consommation globale du circuit en désactivant le périphérique qui y est connecté.

Analyser la figure 2, identifier le signal de commande de la grille du transistor, ainsi que les signaux disponibles pour la communication sur bus synchrone et asynchrone séries.

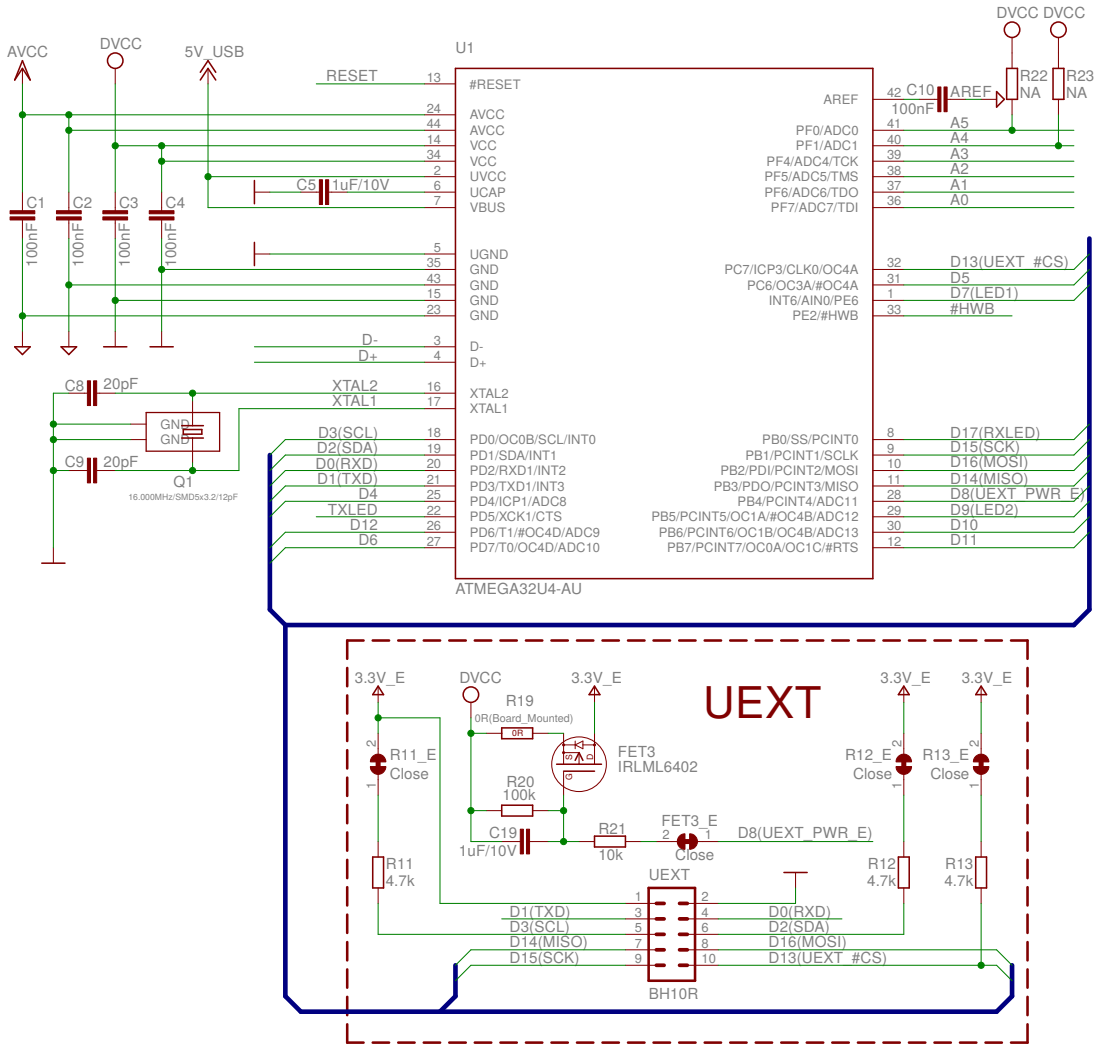


FIGURE 2 – Schéma du connecteur UEXT

On notera que le FET est à canal P [1] et qu'il nécessite un potentiel négatif sur sa grille relativement à la source pour devenir passant.

3 Communication I²C

La mise en pratique de la communication I²C se fait sur un capteur de température et sur une horloge temps-réel (*Real Time Clock* – RTC), un périphérique de faible consommation qui permet de réveiller un microcontrôleur de son mode veille profonde dans lequel il consomme le moins d'énergie. Nous utiliserons, pour communiquer sur le bus I²C, les fonctionnalités à cet effet fournies dans `LUFA`.

Après avoir téléchargé la bibliothèque de communication I²C à `http://jmfriedt.free.fr/lib_atmega.tar.gz`, consulter `include/libi2c.h` pour identifier les fonctions disponibles, et `src/libi2c.c` pour en appréhender les implémentations.

On notera quelques subtiles différences entre SPI et I²C. Dans la documentation de l'accéléromètre Bosch BMA220 qui supporte les deux modes de communication (Fig. 3), nous constatons que l'ordre des bits n'est pas le même, un point à mémoriser lors de l'observation des signaux de données à l'oscilloscope. Par ailleurs, I²C adresse les périphériques. contrairement à SPI qui sélectionne l'esclave actif en abaissant un signal de sélection (*Chip Select#*). Par ailleurs, le 8ème bit de la transaction I²C indique la nature de l'échange : 1 pour une lecture, 0 pour une écriture.

Exercice : en observant l'extrait de la documentation de la Fig. 3, quelle adresse faut-il fournir pour lire un registre de l'accéléromètre BMA220 si CSB est connecté à la tension d'alimentation ?

3.1 Capteur de température LM75

Le capteur de température LM75 propose une interface I²C pour transmettre une information numérique issue d'un capteur codant sur 9 bits des températures comprises dans l'intervalle ±125°C incluant un bit de signe. Nous avons câblé ce

1. `http://www.irf.com/product-info/datasheets/data/irlml6402.pdf`

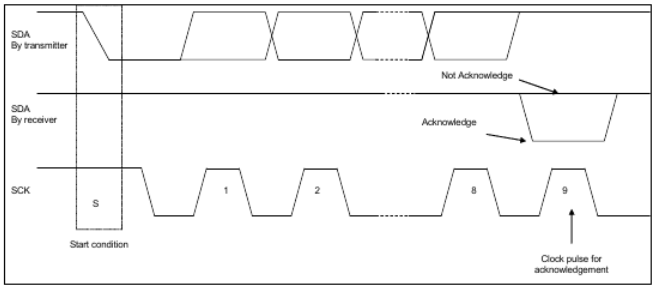


Figure 20: Waveform diagram for I²C acknowledgement on SDA

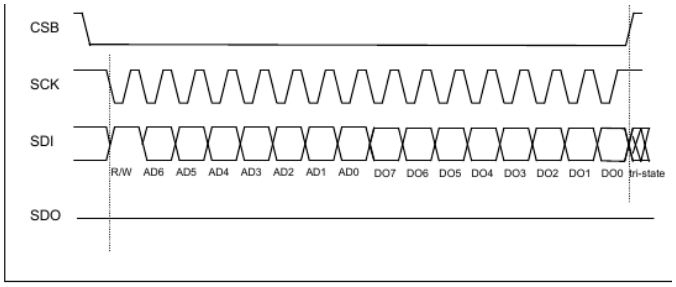


Figure 18: 3-wire SPI read protocol

FIGURE 3 – Extrait de la documentation de l'accéléromètre Bosch BMA220 illustrant les chronogrammes des communications synchrones SPI et I²C. Noter la différence dans l'organisation des données, à prendre en compte lors de l'observation sur oscilloscope des signaux.

Important:
The default slave address assigned to the BMA220 is *000 1011*. When in I²C mode, the LSB can be inverted by tying the CSB pin to '1'. This allows resolving conflicts with existing devices. Also, the 4 LSB can be configured via OTP.

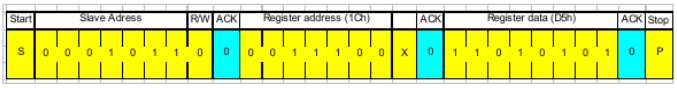
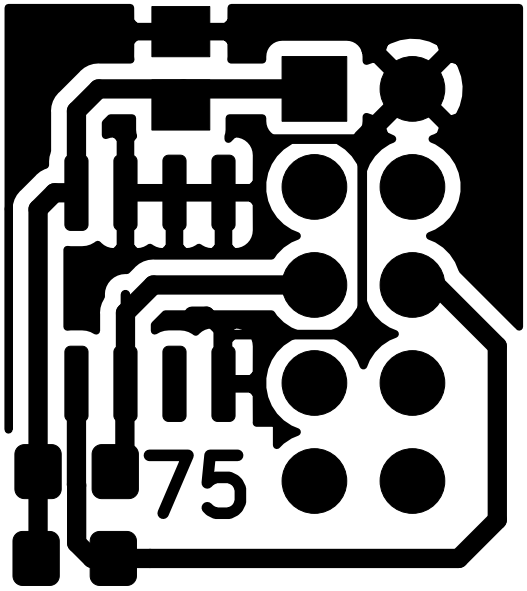
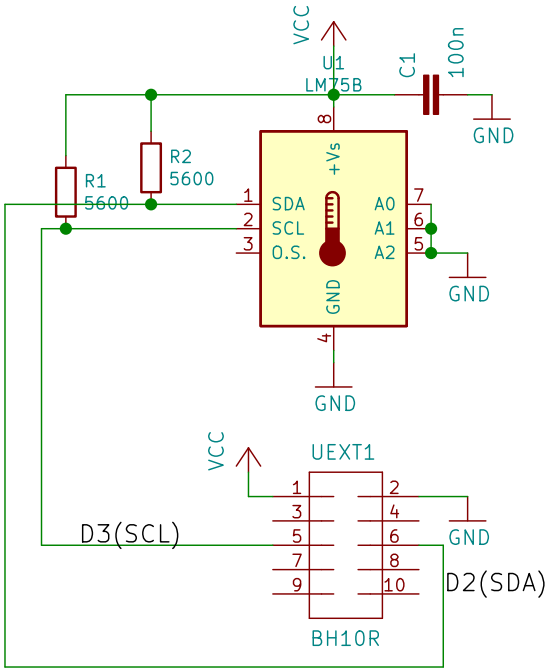


Figure 22: I²C one byte write protocol

composant suivant le schéma suivant :





Identifer les résistances de tirage (*pull up*) sur le bus de données et l'horloge afin de définir à tout moment l'état des bus même si les deux interlocuteurs se placent en haute impédance. **Nous avions initialement omis de placer ces deux résistances de tirage, et pourtant le circuit fonctionne : pourquoi ?**

L'adresse I²C est décrite dans l'extrait suivant de la datasheet[2] :

Table 1. Slave Address

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	0	0	1	A2	A1	A0	R/$\overline{W}$

Table 2. Register Functions

REGISTER NAME	ADDRESS (hex)	POR STATE (hex)	POR STATE (binary)	POR STATE (°C)	READ/ WRITE
Temperature	00	000X	0000 0000 0XXX XXXX	—	Read only
Configuration	01	00	0000 0000	—	R/W
T_{HYST}	02	4B0X	0100 1011 0XXX XXXX	75	R/W
T_{OS}	03	500X	0101 0000 0XXX XXXX	80	R/W

X = Don't care.

2. par exemple chez Maxim IC à `https://datasheets.maximintegrated.com/en/ds/LM75.pdf`

qui indique donc qu'un maximum de 8 capteurs peuvent être connectés à un même bus I^2C puisque le quartet de poids faible de l'adresse se décline sur 3 bits (A0 à A2).

La mesure quand à elle s'obtient par une lecture sur 16 bits du contenu du registre d'adresse 0 :

Table 3. Temperature, T_{HYST}, and T_{OS} Register Definition

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64˚C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

X = Don't care.

Table 4. Temperature Data Output Format

TEMPERATURE (°C)	DIGITAL OUTPUT	
	BINARY	HEX
+125	0111 1101 0XXX XXXX	7D0X
+25	0001 1001 0XXX XXXX	190X
+0.5	0000 0000 1XXX XXXX	008X
0	0000 0000 0XXX XXXX	000X
-0.5	1111 1111 1XXX XXXX	FF8X
-25	1110 0111 0XXX XXXX	E70X
-55	1100 1001 0XXX XXXX	C90X

X = Don't care.

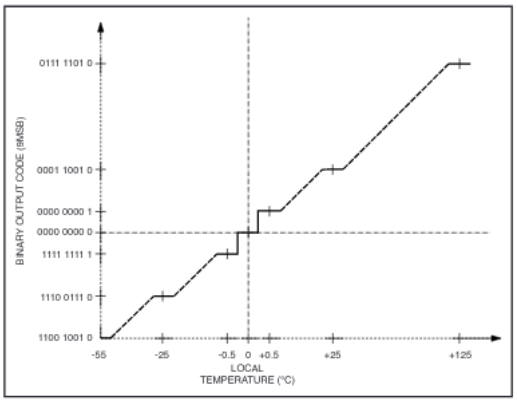
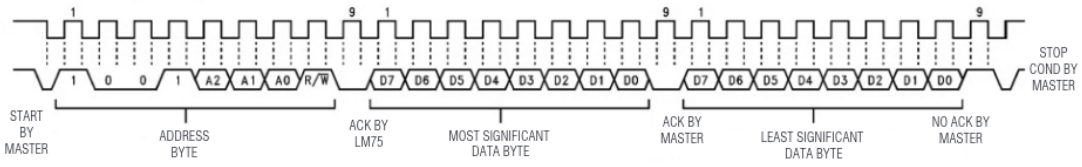


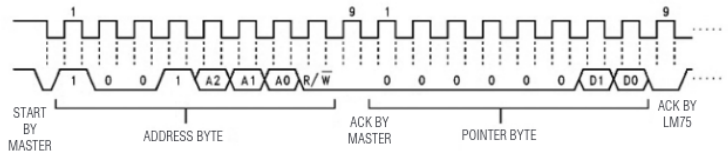
Figure 3. Temperature-to-Digital Transfer Function

Connaissant la gamme de mesure et le nombre de bits pour encoder l'information, quelle est la résolution de mesure? est-ce cohérent avec la résolution fournie par la datasheet?

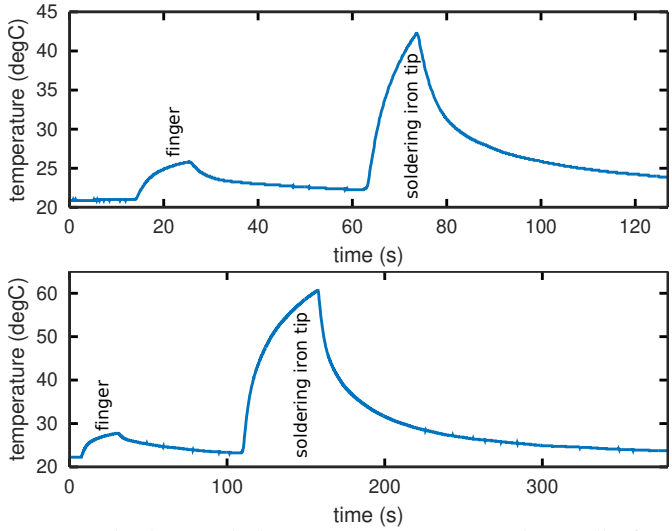
Le chronogramme de la transaction indique que nous commençons par adresser le périphérique, puis indiquer quel registre est lu, avant de réceptionner les 16 bits de données :

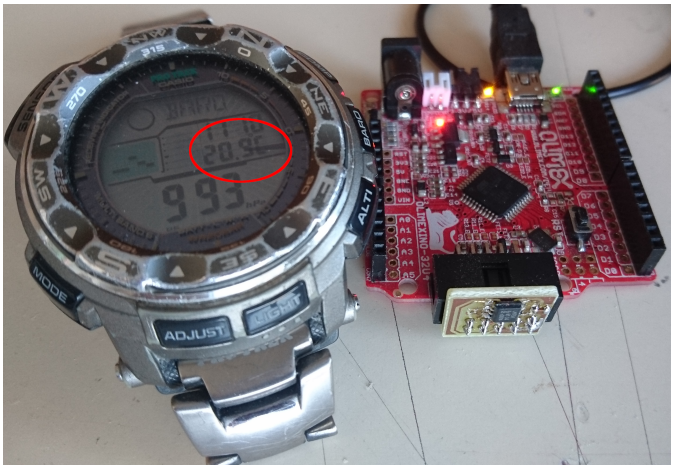


(a) TYPICAL 2-BYTE READ FROM PRESET POINTER LOCATION SUCH AS TEMP, T_{HIGH}, T_{LOW}.



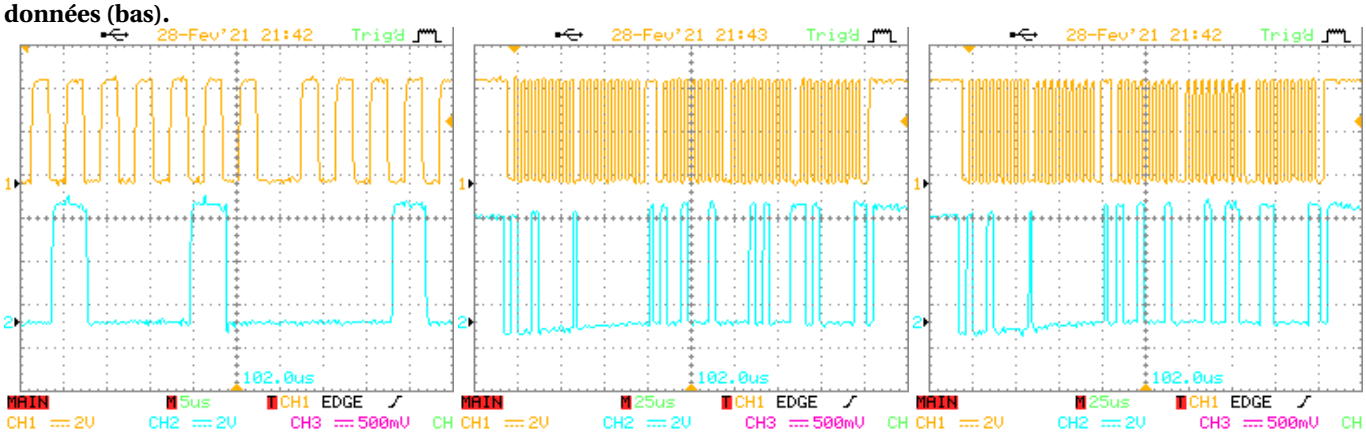
À titre d'illustration de mesures que nous avons obtenues avec ce capteurs, les graphiques ci-dessous démontrent une mesure de température lorsque nous plaçons le doigt sur le capteur (première moitié de la courbe, température maximale en deça de 30°C) puis la pointe de la panne d'une fer à souder :





Le point de départ de la mesure est en accord avec l'information fournie par le capteur d'une montre possédant un capteur de température (droite).

Les graphiques ci-dessous ont été capturés au cours de mesures de température par le LM75 cadencé par l'Atmega32U4. **En analysant le graphique de gauche, vérifier la cohérence des fronts lors des transactions I^{2}C entre horloge (haut) et données (bas).**



En comparant les mesures fournies par le graphique du milieu (données en bas) et de droite (idem), quel graphique a été acquis lorsque la température était la plus élevée? Justifier

En s'inspirant du listing de la section 3.2 sur la lecture de l'horloge temps-réel PCF8563 et en exploitant les fonctions de la bibliothèque disponible à `http://jmfriedt.free.fr/lib_atmega.tar.gz`, **démontrer la capacité à retranscrire sur le port série virtuel accessible par USB une grandeur représentative de la température mesurée par le capteur LM75. Comment s'appelle la bibliothèque après compilation par `make` dans le répertoire `src`? Où se trouve-t-elle?**

3.2 Horloge PCF8563

Nous allons nous intéresser à la RTC PCF8563 qui communique avec le microcontrôleur sur un bus I^{2}C. Son adresse est fournie dans la documentation technique. En mode normal, l'horloge est incrémentée toute les secondes. Dans ce mode, les registres de contrôle sont à zéro (par défaut, tous les registres sont initialisés à zéro à la mise sous tension – voir appendice A). Nous nous proposons donc d'initialiser la RTC, et de vérifier que les divers registres évoluent comme prévu dans une horloge, notamment avec les secondes qui s'incrémentent toutes les secondes et les minutes toutes les 60 secondes.

Le programme ci-dessous propose les fonctionnalités minimales pour accéder à la RTC sur bus I^{2}C. Il permet en particulier de vérifier que les divers registres (Fig. 4) contiennent bien les informations annoncées, et que nous avons compris le protocole de communication avec l'horloge (annexe A). Son affichage est cependant fort peu satisfaisant.

```
1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3  #define F_CPU 16000000UL
4  #include <util/delay.h>
5  #include "libi2c.h"
6
7  #include <string.h>
8  #include <stdio.h>
9
10 #include "VirtualSerial.h"
11
```

Table 4. Formatted registers overview

Bit positions labelled as x are not relevant. Bit positions labelled with N should always be written with logic 0; if read they could be either logic 0 or logic 1. After reset, all registers are set according to Table 27.

Address	Register name	Bit							
		7	6	5	4	3	2	1	0
Control and status registers									
00h	Control_status_1	TEST1	N	STOP	N	TESTC	N	N	N
01h	Control_status_2	N	N	N	TI_TP	AF	TF	AIE	TIE
Time and date registers									
02h	VL_seconds	VL	SECONDS (0 to 59)						
03h	Minutes	x	MINUTES (0 to 59)						
04h	Hours	x	x	HOURS (0 to 23)					
05h	Days	x	x	DAYS (1 to 31)					
06h	Weekdays	x	x	x	x	x	WEEKDAYS (0 to 6)		
07h	Century_months	C	x	x	MONTHS (1 to 12)				
08h	Years	YEARS (0 to 99)							
Alarm registers									
09h	Minute_alarm	AE_M	MINUTE_ALARM (0 to 59)						
0Ah	Hour_alarm	AE_H	x	HOUR_ALARM (0 to 23)					
0Bh	Day_alarm	AE_D	x	DAY_ALARM (1 to 31)					
0Ch	Weekday_alarm	AE_W	x	x	x	x	WEEKDAY_ALARM (0 to 6)		
CLKOUT control register									
0Dh	CLKOUT_control	FE	x	x	x	x	x	FD[1:0]	
Timer registers									
0Eh	Timer_control	TE	x	x	x	x	x	TD[1:0]	
0Fh	Timer	TIMER[7:0]							

FIGURE 4 – Registres de la RTC – vérifier la cohérence du programme avec les informations fournies dans ce tableau.

```
12 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
13 extern FILE USBSerialStream;
14
15 int main(void){
16   uint8_t i;
17   uint8_t mask[]={0x7F, 0x7F, 0x3F, 0x3F, 0x07, 0x1F, 0xFF};
18   uint8_t time[16];               // Le PCF8563 possede 16 registres
19   char buffer[80]="C'est␣parti\r\n\0";
20   SetupHardware();
21   CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
22   GlobalInterruptEnable();
23   fputs(buffer, &USBSerialStream);
24   I2C_UEXT(ON);                   // Alimentation du connecteur UEXT on (D8=PB4)
25   I2C_init(24, 0);                // Initialisation du bus I2C : 24 pour 400kHz
26   I2C_stop();
27
28   time[0]=2;
29   time[1]=0x00;time[2]=0x12;time[3]=5;time[4]=6;time[5]=7;time[6]=8; time[7]=9;
30   // second            min         hour      day      weekday   month     year
31
32   // initialisation RTC
33   for (i=0;i<10;i++)  // repeat: "all accesses must be completed within 1 second" (p.12 DS)
34     if (I2C_write(PCF8563,(uint8_t *)time,8)!=0)   // tester l'echec
35       fputs("Echec␣ecriture\r\n\0", &USBSerialStream);
36   while (1){
37     time[0]=2;                          // registre qu'on veut lire
38     I2C_rw(PCF8563, time, 1, time+1, 7);  // lecture de la RTC
39     for(i=0; i<7; i++)
40        {time[i+1]=time[i+1]&mask[i];
41         buffer[2*i+1]=(time[i+1]&0x0f)+'0';
42         buffer[2*i]=((time[i+1]&0xf0)>>4)+'0';
43        }
44     buffer[14]='\r';buffer[15]='\n';buffer[16]=0; // affichage
45     fputs(buffer, &USBSerialStream);
```

```
46      CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
47      CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
48      USB_USBTask();
49      _delay_ms(1000);                        // attente pour laisser RTC tourner
50    }
51    return 1;
52  }
```

1. **Expliquer l'utilisation du tableau `mask` dans l'affichage.**
2. **Proposer une fonction qui convertit l'affichage du contenu des registres proposé ici (`valeur + '0'`) par une fonction qui affiche le contenu de l'horloge par un utilisateur humain. On prendra en particulier soin de noter que la RTC stocke les informations en BCD.**
3. **Proposer une fonction qui remplace l'initialisation statique proposée dans cet exemple par une demande à l'utilisateur de fournir date et heure, convertisse cette information (ASCII) en format exploitable par la RTC (BCD), et initialise la RTC en conséquent.**

3.3 Alarme de la RTC pour générer une interruption sur Atmega32U4

La RTC propose un mode alarme dans lequel le statut courant de l'horloge (minutes, heures ...) et comparé avec une valeur pré-définie (date de l'alarme). S'il y a coïncidence, un signal actif au niveau bas se déclenche sur la broche INT#, connectée à la broche 4 du connecteur UEXT (D0 (RXD)) qui est liée à D0 de la carte Olimex (INT2).

Pour utiliser l'alarme, il est nécessaire de positionner les bits suivants :

— `AEI` dans le registre 1 à 1 autorise l'interruption sur l'alarme,

— les bits `AE_x` (`x= M, H D` et `W`) à "1" *désactivent* la comparaison de l'alarme avec l'horloge,

— si le bit `AF` est à 1, ce drapeau indique l'égalité de l'alarme et de l'horloge. Il doit être remis à zéro après interruption.

Déclencher une interruption sur l'ATMega 32U4 en notant que le signal – actif au niveau bas – issu de l'alarme est connecté à la broche D0 (INT2) du microprocesseur. On pourra tester manuellement le bon fonctionnement de l'interruption matérielle en reliant par un fil D0 et la masse. Afin de ne pas attendre une minute le déclenchement de l'interruption, on pensera à initialiser l'horloge avec une valeur des secondes proche de 60.

3.4 Mode interruption de la RTC

Le schéma du module d'horloge temps-réelle est disponible à `https://www.olimex.com/Products/Modules/Time/MOD-RTC/resources/MOD-RTC_Rev_B.pdf`

Ce schéma met en évidence la **nécessité d'imposer le potentiel** sur le bus de données du bus I^2C en tirant, par convention vers la tension d'alimentation (pull-up), au moyen de la résistance de tirage R2. En effet dans bus I^2C où SDA (données) est bi-directionnel, il se peut que tous les interlocuteurs se placent en écoute (haute-impédance) et que personne n'impose le courant circulant sur ce fil, rendant son potentiel flottant et inconnu. Cette résistance de tirage a vocation à empêcher cette situation incertaine en imposant le potentiel par défaut. Nous découvrons par ailleurs sur ce schéma que ...

⚠... les résistances R4 (330 Ω) et R8 (2200 Ω) pour le tirage à la tension d'alimentation ne sont pas peuplées sur la carte commercialisée par Olimex. L'activation de l'interruption pour réveiller le microcontrôleur nécessite de souder ces deux composants passifs (Fig. 5).

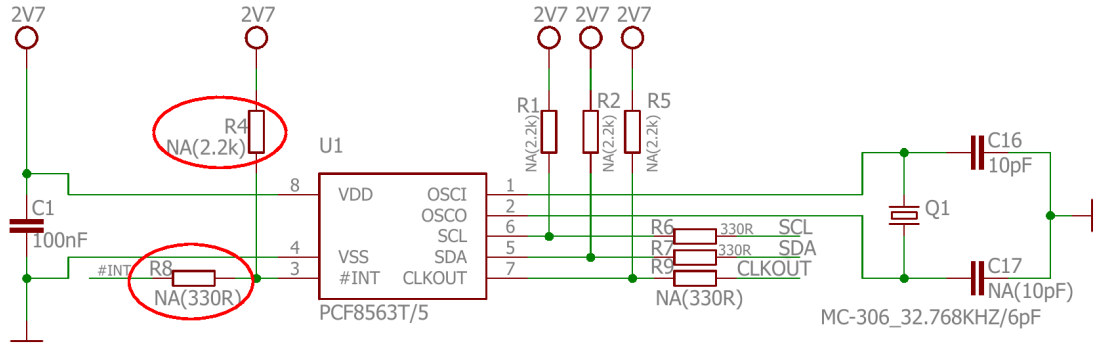


FIGURE 5 – Composants manquant pour faire fonctionner le réveil du microcontrôleur sur déclenchement d'interruption par la RTC.

Une fois la résistance reliant INT# du module RTC à INT0 du microcontrôleur mise en place – on notera que cette broche sert aussi à la liaison du PC vers le microcontrôleur en RS232, qu'on prendra soin de désactiver – le programme ci-dessous met en veille le microcontrôleur et le réveille chaque minute. On notera en particulier l'acquittement de l'interruption, en communiquant sur le bus I2C dans le gestionnaire d'interruption, afin d'abaisser la broche INT# et éviter de boucler dans

le gestionnaire d'interruption. Une fois l'interruption acquittée, un drapeau est mis en place et la prochaine alarme est programmée dans le programme principal.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#define f_CPU 16000000UL
#include <util/delay.h>
#include "libttycom.h"
#include "libi2c.h"

#include <string.h>
#include <stdio.h>

#define USART_BAUDRATE 9600

volatile int alarme=0;

// https://github.com/tomvdb/avr_arduino_leonardo/blob/master/examples/uart/main.c
void uart_transmit(char data )
{while (!(UCSR1A&(1<<UDRE1))) ;
 UDR1 = data;
}

void uart_puts(char *c)
{int k=0;
 while (c[k]!=0) {uart_transmit(c[k]);k++;}
}

void uart_init()
{unsigned short baud;
 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;
}

void enterSleep(void)
{set_sleep_mode(SLEEP_MODE_IDLE);
 sleep_enable();
 sleep_mode(); //enter sleep mode ... good night
 sleep_disable(); //wake up from interrupt
}

ISR(INT2_vect) //front descendant
{alarme=1;
 PORTB^=1<<PORTB5;
 _delay_ms(100);
 }

int main (void)
{int8_t dummy;
 uint8_t i;                                                  //            m_alm          week_alm
 uint8_t mask[13]={0xC8,0x1F,0x7F, 0x7F, 0x3F, 0x3F, 0x07, 0x1F, 0xFF, 0xFF,0xBF,0xBF,0x87};
 uint8_t time[16]; //le PCF8563 possede 16 registres

 wdt_disable();
 uart_init();

 MCUCR &=~1<<PUD;
 DDRD&=~1<<PORTD2;
 PORTD|=1<<PORTD2;

```

```
 I2C_UEXT(ON); //alimentation du connecteur UEXT on (D8=PB4)
 I2C_init(24,0); //initialisation du bus I2C : 24 pour 400kHz
 I2C_stop();

 time[0]=1; //adresse du premier registre (ici seconde)
 time[1]=0x02;
 time[2]=0x55; time[3]=01; time[4]=05; time[5]=06; time[6]=7; time[7]=8; time[8]=9;
//valeur des registres a partir duquel on veut imposer la valeur
//   second      min         hour         day       weekday      month      year

 I2C_write(PCF8563, (uint8_t *) time, 9); // initialisation RTC
 // @ PCF 8563, puis @ premier registre (time[0]) puis val des 8 premiers registres (time[1] a time[7])

 // initialisation alarme
 time[0]=9; //adresse du premier registre (ici seconde)
 time[1]=0x02; time[2]=0x82; time[3]=0x83; time[4]=0x84;

 I2C_write(PCF8563, (uint8_t *) time, 5); //tester l'echec

 EIMSK = 1<<INT2; //enable int2
 EICRA=1<<ISC21; //front descendant
 USBCON=0;
 sei();

 while(1)
  {time[0]=1;
   I2C_rw(PCF8563, time, 1, time+1, 12); //lecteur de la RTC //8

   for(i=1; i<=12; i++)
     {time[i]=time[i]&mask[i];
      buffer[2*i-1]=(time[i]&0x0f)+'0';     if (buffer[2*i-1]>'9') buffer[2*i-1]=buffer[2*i-1]-'9'+'A';
      buffer[2*i-2]=((time[i]&0xf0)>>4)+'0';if (buffer[2*i-2]>'9') buffer[2*i-2]=buffer[2*i-2]-'9'+'A';
     }
    buffer[2*12]=0;
    uart_puts(buffer);

    if (alarme==1)
      {sprintf(buffer,"␣ALARME");uart_puts(buffer);
       time[0]=9;time[1]=time[3]+1; // status=1 seconde=2 minute=3
       I2C_write(PCF8563, (uint8_t *) time, 2);
       time[0]=1;time[1]=0x02;
       I2C_write(PCF8563, (uint8_t *) time, 2);
       alarme=0;
       PORTE ^=1<<PORTE6; //toggle led
       enterSleep();
     }
   uart_puts("\r\n\0");
   _delay_ms(1000); //attente pour laisser RTC tourner
  }
  return 0;
}
```

A Annexe

Extrait de la *datasheet* de la RTC qui indique que le premier octet transmis sur le bus I^{2}C contient l'adresse du premier registre à remplir.

9.5.1 Addressing

Before any data is transmitted on the I^{2}C-bus, the device which should respond is addressed first. The addressing is always carried out with the first byte transmitted after the start procedure.

The PCF8563 acts as a slave receiver or slave transmitter. Therefore the clock signal SCL is only an input signal, but the data signal SDA is a bidirectional line.

Two slave addresses are reserved for the PCF8563:

Read: A3h (10100011)

Write: A2h (10100010)

The PCF8563 slave address is illustrated in Figure 18.

Fig 18. Slave address

9.5.2 Clock and calendar READ or WRITE cycles

The I^{2}C-bus configuration for the different PCF8563 READ and WRITE cycles is shown in Figure 19, Figure 20 and Figure 21. The register address is a 4-bit value that defines which register is to be accessed next. The upper four bits of the register address are not used.

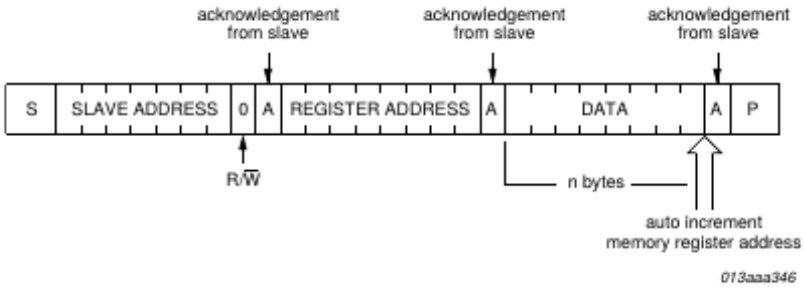


Fig 19. Master transmits to slave receiver (WRITE mode)