TP introduction au microcontrôleur 8 bits

É. Carry, J.-M Friedt

18 janvier 2017

Questions sur ce TP :

1. Sur quelle broche de quel port d'entrée-sortie est connectée la LED1 ?
2. Quel bit placer à quelle valeur pour passer la broche 1 du PORTB en sortie ?
3. Quelle est la conséquence de l'optimisation d'un code C par l'option `-O2` de `gcc` sur une boucle vide utilisée pour introduire une lattence dans l'exécution du code ?
4. Quelle option de `gcc` permet de fournir le code assembleur issu du code source en C ?
5. Donner un nom du premier port série virtuel sous GNU/Linux ? comment lire les informations transmises sur ce port de communication ?
6. Quelle option de `gcc` permet d'introduire les symboles de déverminage (*debug*) dans l'assembleur et le binaire issus de la compilation ?
7. Quel est le facteur de division de l'horloge cadençant un *timer* si les bits CS1 et CS0 sont placés à la valeur 1 ? Justifier.

1 Compilation et GPIO

Le schéma [1] de la Fig. 1 fournit le brochage d'un certain nombre de périphériques dont les LEDs visibles à côté de l'embase d'alimentation de la carte.

Identifier sur le schéma les deux LEDs et les ports auxquelles elles sont connectées

Le comportement des GPIO est décrit dans l'extrait de la Fig. 2 de la feuille descriptive (*datasheet*) de l'Atmega32U4 disponible à `http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf` (p.67) [2].

Le programme ci-dessous configure deux ports GPIO en sortie, et manipule l'état de ces sorties pour faire clignoter les diodes.

Listing 1 – Diode clignotante

```
1  #include <avr/io.h> //E/S ex PORTB
2  #define F_CPU 16000000UL
3  #include <util/delay.h>
4
5  int main(void){
6    DDRB |=1<<PORTB5;
7    DDRE |=1<<PORTE6;
8    PORTB |= 1<<PORTB5;
9    PORTE &= ~(1<<PORTE6);
10
11   while (1){
12     /* Clignotement des LEDS */
13   PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
14   _delay_ms(500);
15   }
16   return 0;
17 }
```

Analyser ce programme compte tenu du descriptif de la *datasheet* fourni en Fig. 2.

La compilation du programme (passage du code C en binaire au format ELF) s'obtient par la version de `gcc` configurée pour produire du code à destination de l'architecture AVR 8 bits par

```
avr-gcc -mmcu=atmega32u4 -Os -Wall -o blink.out blink.c
```

1. https://www.olimex.com/Products/Duino/AVR/OLIMEXINO-32U4/open-source-hardware

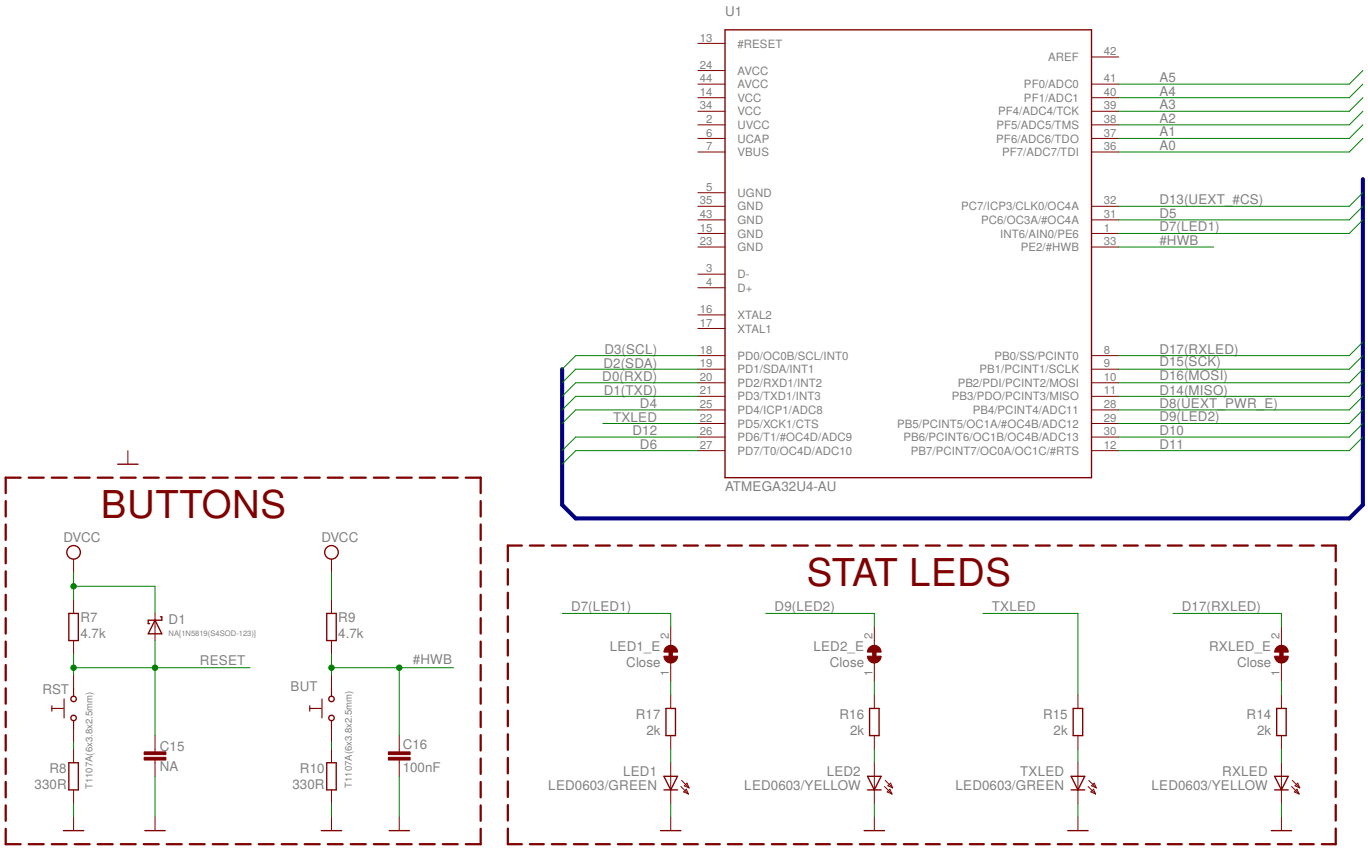


FIGURE 1 – Extrait du schéma de l'Olimexino-32U4

Table 10-1. Port Pin Configurations

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

FIGURE 2 – Extrait du schéma de l'Olimexino-32U4

1.1 Exécution du code

À peu près tous les microcontrôleurs modernes sont équipés d'un bout de code chargé du transfert des données et de leur écriture en mémoire non-volatile (*bootloader*). Un logiciel dédié est nécessaire pour communiquer avec le bootloader : dans le cas de l'AVR, il s'agit de `avrdude` qui s'exécute *après appui du bouton RST* par

```
avrdude -c avr109 -b57600 -D -p atmega32u4 -P /dev/ttyACM0 -e -U flash:w:blink.out
```

en prenant comme arguments la déclinaison du microcontrôleur, le débit de transfert, et le port de communication (le port `ttyACM` numéro zéro, soit le premier port série virtuel).

Observer le comportement des LEDs situées à droite de l'embase d'alimentation de la carte.

1.2 Analyse du code

Convertir un code C en langage compatible d'un microprocesseur est une opération complexe qui passe par de nombreuses étapes. `gcc` permet d'arrêter la compilation à chacune de ces étapes pour en voir le résultat.

`gcc -E` arrête au précompilateur qui a simplement effectué l'analyse syntaxique du code.

`gcc -S` arrête à la génération du code assembleur.

Une fois le binaire généré, il est possible de revenir au code assembleur correspondant aux opcodes par `objdump -dSt fichier.out`.

Si les symboles de déverminage ont été inclus dans le binaire (`gcc -g`), alors les lignes de C correspondantes aux instructions assembleur sont fournies dans le listing.

Par ailleurs, `objcopy` permet de convertir des fichiers entre de nombreux formats, et en particulier le binaire ELF en images de la mémoire du microcontrôleur dans les deux formats les plus courants, Intel Hex et Motorola S19.

iHex : `avr-objcopy -O ihex blink.out blink.hex`

S19 : `avr-objcopy -O srec blink.out blink.s19`

Il est ainsi possible d'utiliser un outil de programmation ne reconnaissant que ces formats (en particulier les outils de programmation propriétaires sont souvent peu flexibles sur le format de données compris).

1.3 Options d'optimisation

Un code C n'est pas bijectif avec un code assembleur : divers compilateurs vont générer diverses implémentations du même code C, et un même compilateur peut être configuré pour optimiser le code selon divers critères (taille, vitesse). Ces optimisations peuvent cependant avoir des effets indésirables s'ils ne sont pas maîtrisés.

Listing 2 – Diode clignotante avec retard manuel

```
1  #include <avr/io.h> //E/S ex PORTB
2  #define F_CPU 16000000UL
3
4  void mon_delai(long duree)
5  {long k=0;
6   for (k=0;k<duree;k++) {};
7  }
8
9  int main(void){
10   DDRB |=1<<PORTB5;
11   DDRE |=1<<PORTE6;
12   PORTB |= 1<<PORTB5;
13   PORTE &= ~(1<<PORTE6);
14
15   while (1){
16     PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
17     mon_delai(0xffff);
18   }
19   return 0;
20 }
```

L'exemple ci-dessus suit l'approche naïve d'implémenter un retard par une boucle vide.

Le code se compile par `avr-gcc -mmcu=atmega32u4 -Wall -o blink.out blink.c` et l'assembleur résultant s'observe par `avr-objdump -dS blink.out`

1. **compiler avec l'option `-O0` et observer la fonction `mon_delai()` résultante.**
2. **compiler avec l'option `-O2` et observer la fonction `mon_delai()` résultante.**
3. **compiler avec l'option `-O1` et observer la fonction `mon_delai()` résultante.**
4. **compiler avec l'option `-O2` après avoir préfixé la définition de la variable `k` par le terme `volatile` qui interdit toute hypothèse du compilateur sur la valeur de la variable, et observer la fonction `mon_delai()` résultante.**
5. **compiler avec l'option `-g -O2` et observer la fonction `mon_delay()` résultante.**

2 Programmation modulaire – compilation séparée

Un programme est efficacement séparé en modules indépendants, voir en bibliothèques. La compilation séparée consiste à compiler chaque programme en objets indépendants, qui sont ensuite liés (*linker*) en un exécutable unique.

Dans cet exemple, nous désirons pouvoir réutiliser la fonction d'attente en le compilant comme objet qui peut être rappelé par divers programmes. Cependant, nous devons déclarer la présence de la fonction dans l'objet : c'est le rôle du fichier d'entête `.h` (*header file*).

⚠Attention : un fichier d'entête ne contient *jamais* de code C.

Listing 3 – Fichier principal `separe_blink.c`

```
1  #include <avr/io.h> //E/S ex PORTB
```

```
2
3  #define F_CPU 16000000UL
4
5  #include "separe_delay.h"
6
7  int main(void){
8    DDRB |=1<<PORTB5;
9    DDRE |=1<<PORTE6;
10   PORTB |= 1<<PORTB5;
11   PORTE &= ~(1<<PORTE6);
12
13   while (1){
14     PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
15     mon_delai(0xffff);
16   }
17   return 0;
18 }
```

Listing 4 – Fonction de retard : `separe_delay.c`

```
1 void mon_delai(long duree)
2 {long k=0;
3  for (k=0;k<duree;k++) {};
4 }
```

Listing 5 – Fichier d'entête de définition des fonctions : `separe_delay.h`

```
1 void mon_delai(long);
```

Ces fichiers se compilent (pour d'abord générer les deux objets qui sont ensuite liés en un binaire) par

```
avr-gcc -mmcu=atmega32u4 -Os -Wall -c separe_delay.c
avr-gcc -mmcu=atmega32u4 -Os -Wall -c separe_blink.c
avr-gcc -mmcu=atmega32u4 -o blink_separe.out separe_blink.o separe_delay.o
```

Un outil spécifique permet de tenir compte des dépendances : `make`, qui prend un fichier de configuration nommé par défaut `Makefile`.

Par ailleurs, la compilation séparée nécessite de déclarer les fonctions ou variables définies dans un autre objet. Les fichiers d'entête (*header*) regroupent les définitions de fonctions, tandis que la définition de variable préfixée de `extern` annonce une définition dans un autre objet (avec l'allocation de mémoire associée).

Listing 6 – Exemple de Makefile

```
1  all:  separe_blink.out
2
3
4  separe_blink.out: separe_delay.o separe_blink.o
5     avr-gcc -mmcu=atmega32u4 -Os -Wall -o separe_blink.out separe_blink.o separe_delay.o
6
7  separe_delay.o: separe_delay.c separe_delay.h
8     avr-gcc -mmcu=atmega32u4 -Os -Wall -c separe_delay.c
9
10 separe_blink.o: separe_blink.c
11    avr-gcc -mmcu=atmega32u4 -Os -Wall -c separe_blink.c
12
13 clean:
14    rm *.o separe_blink.out
15
16 flash: separe_blink.out
17    avrdude -c avr109 -b57600 -D -p atmega32u4 -P /dev/ttyACM0 -e -U flash:w:separe_blink.out
```

Deux méthodes y sont classiquement définies : `all` indique la cible finale à atteindre, `clean` fournit les commandes de nettoyage du répertoire. Chaque ligne contient la cible, les dépendances puis la méthode à mettre en œuvre si les dépendances sont résolues.

⚠Attention : chaque méthode commence par une tabulation et *pas* par des espaces.

3 Communication – appel à une bibliothèque

LUFA [2] est une bibliothèque riche de gestion du protocole USB, complexe à mettre en œuvre [1] compte tenu de la variété des conditions d'utilisation et des transferts effectués lors de l'initialisation des transactions avec le PC. Ces initialisations ont en particulier vocation à informer le système d'exploitation du PC des capacités de la liaison USB (nombre de points de communication, débit, nature du pilote susceptible de prendre en charge ces transactions sur le PC).

Une bibliothèque acquise sous forme de code source doit être compilée. Rechercher l'archive compressée de la bibliothèque à `http://jmfriedt.free.fr/LUFA_light_EEA.tar.gz`. Désarchiver le contenu du fichier par `tar zxvf LUFA_light_EEA.tar.gz`. Entrer dans le répertoire ainsi créé `VirtualSerial_lib` et compiler la bibliothèque par `make lib` (noter la présence d'un fichier nommé `Makefile` dans ce répertoire). Lors de la compilation, les entêtes se trouvent dans `lufa-LUFA-140928` et l'archive complète des fonctions fournies dans la bibliothèque dans `libVirtualSerial.a`. On copiera donc ce dernier fichier dans le répertoire du projet nécessitant les fonctionnalités de communication sur bus USB. Par convention de `gcc`, l'édition de liens faisant appel à la bibliothèque `libtoto` fournie sous forme d'archive statique `libtoto.a` s'obtient par l'option `-ltoto`. Il peut être souhaitable de préciser l'emplacement de la bibliothèque par l'option `-L répertoire`.

Nous proposons une version allégée de la bibliothèque et un exemple simple de transactions sur port série virtuel, nommé sous GNU/Linux `/dev/ttyACM0`. Pour compiler un programme lié à cette bibliothèque :

1. aller dans le répertoire `VirtualSerial`, y copier la bibliothèque `libVirtualSerial.a`,
2. compiler l'exemple par `make`
3. On pourra éventuellement coupler compilation et transfert du programme au microcontrôleur sur la carte Olinuxino32U4 par `make flash_109`.
4. Constater le bon fonctionnement du logiciel par `cat < /dev/ttyACM0`

Ainsi, nous indiquons dans le `Makefile` le chemin vers cette bibliothèque au moment de l'édition de liens (*linker*) par `-L./lib` (chemin relatif par rapport au répertoire courant, qui pourrait être absolu). De la même façon, chaque objet a besoin des définitions des fonctions fournies par la bibliothèque telles que mentionnées dans les fichiers d'entête dont l'emplacement est défini par `-I./include`.

Noter la nécessité d'inclure le fichier d'entête `VirtualSerial.h` qui définit les fonctions nécessaires au fonctionnement de l'USB, ainsi que les structures de données

```
extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
extern FILE USBSerialStream;
```

Listing 7 – Communication au travers d'un port série (asynchrone) virtuel sur bus USB

```
1  #include <avr/io.h>
2  #include <avr/wdt.h>
3  #include <avr/power.h>
4  #include <avr/interrupt.h>
5  #include <string.h>
6  #include <stdio.h>
7
8  #include "VirtualSerial.h"
9  #include <util/delay.h>
10
11 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
12 extern FILE USBSerialStream;
13
14 int main(void)
15 {
16  char ReportString[20] = "World\r\n\0";
17
18  SetupHardware();
19 /* Create a regular character stream for the interface so that it can be used with the stdio.h functions */
20   CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
21   GlobalInterruptEnable();
22
23   for (;;)
24     {
25      fprintf(&USBSerialStream,"Hello ");
```

2. *Lightweight USB Framework for AVRs*, `www.fourwalledcubicle.com/LUFA.php`

```
26      fputs(ReportString, &USBSerialStream);
27
28 //   les 3 lignes ci-dessous pour accepter les signaux venant du PC
29      CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
30
31      CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
32      USB_USBTask();
33      _delay_ms(500);
34     }
35 }
```

À la compilation, `make` déroule alors la séquence

```
avr-gcc -mmcu=atmega32u4 -Os -Wall -c separe_delay.c
avr-gcc -c -mmcu=atmega32u4 -Wall -I. -I../lufa-LUFA-140928/ -DF_USB=16000000UL \
        -DF_CPU=16000000UL -Os -std=gnu99  VirtualSerial.c
avr-gcc -mmcu=atmega32u4 -L. separe_delay.o VirtualSerial.o -o VirtualSerial.elf -lVirtualSerial
```

L'interface USB est initialisée par `CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);` tandis que le chien de garde et la cadence d'horloge sont configurés par `SetupHardware()` (lire le code source de `VirtualSerial_lib/VirtualSerialConfiguration.c`. Enfin, les trois lignes

```
CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
USB_USBTask();
```

dans la boucle infinie gèrent les messages venant du PC vers le microcontrôleur. Omettre ces lignes n'empêche par le bon fonctionnement de la liaison microcontrôleur vers PC, mais trop de caractères venant du PC vont remplir la pile d'entrée de l'USB et rendre le logiciel inopérant.
Exercice : compléter l'affichage du message par défaut par la taille des trois types d'entiers gérés par le C.

4 GPIO en entrée – problème de rebonds

Compte tenu de l'extrait de la datasheet de la Fig. 2, le programme ci-dessous affiche l'état du port associé aux marquages A0-A5 sur la carte Olimexino-32U4.

Analyser le schéma de la Fig. 1 et expliquer le choix des registres. En particulier, tester le programme en retirant la ligne `PORTF |= 1<<PORTD0;` et observer le comportement lorsqu'un fil est alternativement connecté ou non entre GND et A5.

Listing 8 – Lecture d'un GPIO

```
1  #include <avr/io.h>        //E/S ex PORTB
2  #define F_CPU 16000000UL  //T=62.5ns
3  #include <util/delay.h>   // _delay_ms
4  #include "VirtualSerial.h"
5
6  extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
7  extern FILE USBSerialStream;
8
9  int main(void){
10   int val=0;
11   char aff[3];
12   SetupHardware();
13   CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
14   GlobalInterruptEnable();
15   DDRF  &=(~1<<PORTF0);// entree
16   PORTF |=  1<<PORTF0; // pull up
17   MCUCR &=~(1<<PUD);
18   while(1) {val=PINF&0x01;
19             aff[0]='0'+(val/16);aff[1]='0'+(val%10);aff[2]=0;
20             fputs("F=\0", &USBSerialStream);fputs(aff, &USBSerialStream);fputs("\r\n\0", &USBSerialStream);
21             delay(0xff);
22             fprintf(&USBSerialStream,"\r\n");
23             CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
24             CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
```

```
            USB_USBTask();
          }
}
```

Plutôt que simplement observer la transition en affichant l'état du port, nous désirons compter le nombre de transitions sur le GPIO au moyen du programme proposé à :

Listing 9 – Comptage du nombre de transitions sur une broche d'un GPIO

```
#include <avr/io.h> //E/S ex PORTB
#include <stdio.h>  //sprintf
#define F_CPU 16000000UL //T=62.5ns
#include <util/delay.h>  // _delay_ms
#include "VirtualSerial.h"



#define MAXCAR  25

extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
extern FILE USBSerialStream;


int main(void){
  int val=0,ancien=0,cpt=0;
  char aff[3];
  SetupHardware();
  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
  GlobalInterruptEnable();

  DDRF  &=(~1<<PORTF0);// entree
  PORTF |=  1<<PORTF0; // pull up
  MCUCR &=~(1<<PUD);
  while(1) {val=(PINF&0x01);
            if (val!=ancien)
               {cpt++;
                aff[0]='0'+(cpt/16);aff[1]='0'+(cpt%10);aff[2]=0;
                fputs("F=\0", &USBSerialStream);
                fputs(aff, &USBSerialStream);
                fputs("\r\n\0", &USBSerialStream);
               }
            ancien=val;
            CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
            CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
            USB_USBTask();
           }
}
```

Expliquer les observations lorsqu'un fil est alternativement connecté ou non entre GND et A5.

La multitude des décomptes est un problème classique des transitoires sur les entrées numériques qui se résout par le *debounce* consistant à filtrer (passe bas) la connexion entre le fil et la broche. La version logicielle d'un filtre passe bas est une temporisation.

5 Un périphérique complexe le *timer*

Parmi les périphériques disponibles autour du cœur de calcul du microcontrôleur, les fonctions de temporisation, ou *timer*, sont probablement les plus utiles. La fonction la plus simple est de fournir un signal au cœur lorsqu'une condition de délai a été atteinte : ceci permet d'une part de libérer le processeur pour des tâches plus intéressantes que l'attente dans une temporisation, voir de le placer en mode veille où la consommation est réduite, mais surtout de garantir de délai indépendamment de toute variabilité du logiciel (par exemple optimisations du code, voir la section 1.3).

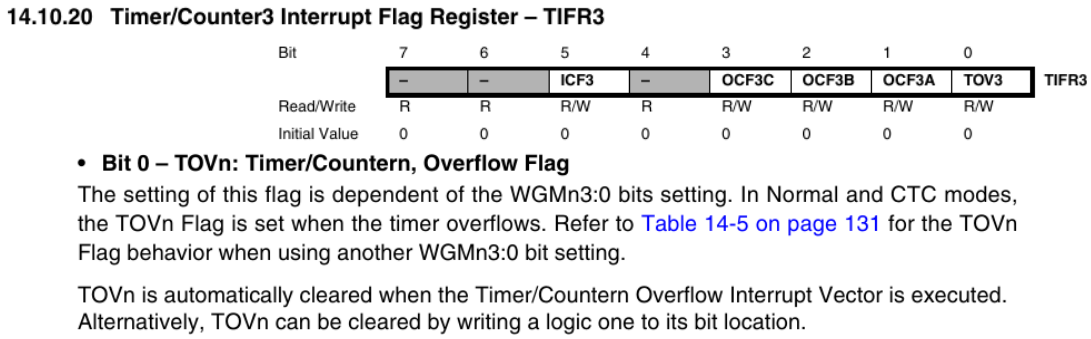
Les codes ci-dessous (10 et 11) s'affranchissent des aléas de l'optimisation du logiciel en exploitant les fonctions matérielles du micrcontrôleur.

Le *timer* est un périphérique un peu plus compliqué à maîtriser compte tenu du nombre de modes de fonctionnement : en entrée (*input capture*), en sortie (*output compare*) ou en saturation de comptage (*overflow*).

5.1 *Overflow*

En mode dépassement, un compteur tourne librement. Lorsque le seuil de comptage est atteint, un drapeau (*flag*) est commuté. Écrire sur ce flag le remet à 0, tel qu'illustré sur le code 10. Le timer0 est sur 8 bits (*datasheet* [2] chapitre 13), le timer 1 sur 16 bits (*datasheet* chapitre 14).

Consulter l'extrait de la *datasheet* de la Fig. 3 et analyser le comportement du *timer* en mode *overflow*.



14.10.20 Timer/Counter3 Interrupt Flag Register – TIFR3

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF3	–	OCF3C	OCF3B	OCF3A	TOV3	TIFR3
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 0 – TOVn: Timer/Countern, Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn Flag is set when the timer overflows. Refer to Table 14-5 on page 131 for the TOVn Flag behavior when using another WGMn3:0 bit setting.

TOVn is automatically cleared when the Timer/Countern Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.

FIGURE 3 – Extrait de la *datasheet* du Atmega32U4 décrivant le comportement en mode *overflow* du timer1.

Listing 10 – Timer pour temporiser le clignotement des LEDs

```
1  #include <avr/io.h>
2  #define F_CPU 16000000UL
3
4  void timer1_init()  // Timer1 avec prescaler=64 et mode normal
5  {TCCR1B |= (1 << CS11)|(1 << CS10);
6  }
7
8  int main(void)
9  {
10   DDRB |=1<<PORTB5;
11   DDRE |=1<<PORTE6;
12   PORTB |= 1<<PORTB5;
13   PORTE &= ~(1<<PORTE6);
14
15   timer1_init();
16   while(1)
17     {if (TIFR1 & (1 << TOV1))
18        {PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
19         TIFR1 |= (1 << TOV1); // clear flag
20        }
21     }
22 }
```

Dans ce mode, le seul degré de liberté pour ajuster la temporisation est le facteur de division de l'horloge qui cadence le compteur, un mode grossier d'ajustement qui ne répond pas toujours aux exigences de la définition d'une fréquence (ou période) précise.

5.2 *CTC*

Clear Timer on Compare Match (CTC) Mode permet de définir un seuil avec lequel le compteur est comparé : atteindre ce seuil induit une transition sur un drapeau (*flag* – Fig. 4) tel que illustré sur le code 10.

Listing 11 – Timer pour temporiser le clignotement des LEDs

```
1  #include <avr/io.h>
2  #define F_CPU 16000000UL
3
4  #define delai_leds (16000*2)
5
6  void timer1_init()  // Timer1 avec prescaler=64 et CTC (WGM=2)
7  {TCCR1B |= (1 << WGM12)|(1 << CS11)|(1 << CS10);
```

14.10.20 Timer/Counter3 Interrupt Flag Register – TIFR3

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF3	–	OCF3C	OCF3B	OCF3A	TOV3	TIFR3
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn value matches the Output Compare Register A (OCRnA).

Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA Flag.

OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

FIGURE 4 – Extrait de la *datasheet* du Atmega32U4 décrivant le comportement en mode *CTC* du timer1.

```
 OCR1A = delai_leds;  // valeur seuil <- delai
}

int main(void)
{
  DDRB |=1<<PORTB5;
  DDRE |=1<<PORTE6;
  PORTB |= 1<<PORTB5;
  PORTE &= ~(1<<PORTE6);

  timer1_init();
  while(1)
    {if (TIFR1 & (1 << OCF1A))
       {PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
        TIFR1 |= (1 << OCF1A); // clear flag
       }
    }
}
```

Modifier la temporisation et observer le résultat

Références

[1] J.-M Friedt, S. Guinot, *Programmation et interfaçage d'un microcontroleur par USB sous linux : le 68HC908JB8*, GNU/Linux Magazine France, Hors Série **23** (November/Décembre 2005)

[2] Atmel, 8-bit Microcontroller with 16/32K Bytes of ISP Flash and USB Controller – ATmega16U4 & ATmega32U4, disponible à `http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf` (version 776F–AVR–11/10)