# TP introduction au microcontrôleur 8 bits

É. Carry, J.-M Friedt

1er novembre 2016

## 1 Arborescence

L'arborescence de GNU/Linux et comment s'y retrouver, dans l'ordre de priorité :

— / est la racine de l'arborescence. Le séparateur entre répertoire sera toujours le /.
— /home contient les répertoires des utilisateurs. Unix, et Linux en particulier, impose un confinement strict des fichiers de chaque utilisateur dans son répertoire. Seul l'administrateur (root) peut accéder aux fichiers du système en écriture : personne ne travaille comme administrateur.
— /usr contient les programmes locaux à un ordinateur : applications, entêtes de compilation (/usr/include), sources du système (/usr/src)
— /lib contient les bibliothÃÍques de fonctions du système d'exploitation
— /proc et /sys contient des pseudo fichiers fournissant des informations sur la configuration et l'état du matériel et du système d'exploitation. Par exemple, cat /proc/cpuinfo.
— /etc contient les fichiers de configuration du système ou la configuration par défaut des logiciels utilisés par les utilisateurs en l'absence de fichier de configuration locale (par exemple cat /etc/profile).
— /bin contient les outils du système accessibles aux utilisateurs et /sbin contient les outils du systèmes qu'*a priori* seul l'administrateur a à utiliser.
— . (un point) est le répertoire courant
— .. (deux points) est le répertoire au-dessus du répertoire courant et est utilisé pour remonter dans l'arborescence

Les commandes de base :

— ls : list, afficher le contenu d'un répertoire
— mv : move, déplacer ou renommer un fichier. (mv source dest)
— cd : change directory, se déplacer dans l'arborescence. Pour remonter dans l'aroborescence : (cd ..)
— rm : remove, effacer un fichier
— cp : copy, copier un fichier. (cp source dest) Afin de copier un répertoire et son contenu, (cp -r repertoire destination).
— cat : afficher le contenu d'un fichier. (cat fichier)
— mkdir : créer un répertoire (mkdir toto)
— man, manuel, est la commnde la plus importante sous Unix, qui donne la liste des options et le mode d'emploi de chaque commande.

**Créez un répertoire nommé** TP1 **et déplacez-vous dedans.**

Afin de remonter dans l'historique, utilisez la flèche vers le haut. Toute commande est lancée en tâche de fond lorsqu'elle est terminée par &.

## 2 Commandes simples

### 2.1 man

La commande la plus importante, et au pire la seule à connaître, est man pour accéder aux pages de documentations. man commande informe de l'utilisation d'une commande, ses arguments et ses options. Les pages de manuel sont organisées par sections, avec l'organisation décrite dans ... man man, et reproduite ci-dessous

```
1   Executable programs or shell commands
2   System calls (functions provided by the kernel)
3   Library calls (functions within program libraries)
4   Special files (usually found in /dev)
5   File formats and conventions eg /etc/passwd
6   Games
7   Miscellaneous  (including  macro  packages and conven-
tions), e.g. man(7), groff(7)
8   System administration commands (usually only for root)
```

**Observer la différence entre** `man 2 open` **et** `man 1 open`**. Commenter**

Si une commande n'est pas connue, `man -k mot_cle` recherche l'occurence du mot clé dans la desription des pages de manuel et facilite l'identification de la commande associée.

**Imaginons que nous voulions trouver la commande en C qui nous permet de changer l'ordre des octets dans un mot de plus de 8 bits (et régler l'endianness : tapez la commande** `man -k order` **et identifiez la commande appropriée.**

### 2.2 grep

`grep` recherche l'occurence d'un motif dans une chaîne de caractères. Elle est très utile pour *filtrer* la masse d'informations issues des autres commandes que nous verrons plus bas, et s'utilise donc quotidiennement pour rendre le résultat d'une recherche digeste.

**Nous désirons connaîtres les utilisateurs ayant accès aux ports série du PC, et donc appartenant au group *dialout*.**

```
grep dialout /etc/group
```

`grep` recherche la présence d'une chaîne de caractères dans l'argument qui lui est fourni, par exemple un fichier. Il sait aussi recherche récursivement dans une arborescence avec l'option `-r`

**Nous avons vu plus haut que la commande** `htons` **inverse les octets d'un mot de 16 bits. Nous désirons connaître l'implémentation de cette commande : trouver l'occurence de cette commande dans les sous-répertoires de le répertoire des fichiers d'entête et ses sous-répertoires** `/usr/include/linux`

`grep` propose une multitude d'options qui s'apprennent avec le temps : `-i` pour ne pas faire dépendre la recherche de majuscule/minuscule, `-v` pour rejeter au lieu d'accepter la correspondance du motif, `-A` et `-B` pour afficher les lignes avant et après celle correspondant au motif.

### 2.3 find

`find` est une commande à la syntaxe un peu complexe mais fort utile pour trouver un ou des fichier(s) dans une arborescence.

Recherche d'un fichier : `find` (locate) qui prend comme argument le répertoire de départ de la recherche, la nature de la recherche (nom du fichier, date de création, permission ...), le motif à rechercher, et l'opération à effectuer.

Exemple : on trouve l'emplacement de `stdio.h` par `find /usr/include/ -name stdio.h -print` L'action `-print` est celle effectuée par défaut si aucune action n'est précisée.

**Rechercher tous les fichiers liés au microprocesseur 32u4 dans le répertoire des fichiers d'entête de** `avr-gcc`**, à savoir** `/usr/lib/avr`

Une option très puissance de `find` est l'exécution d'une commande sur le résultat de la recherche, par l'action `-exec`. Dans ce cas, l'argument est passé sous forme de la chaîne `{}` et la commande se termine par `\;`

Exemple : faire défiler toutes les photos d'un répertoire au moyen de l'affichage par la commande `display` de ImageMagick

```
find . -name '*.jpg' -exec display -delay 100 {} \;
```

## 3 Programmation en langage C

### 3.1 Hello World

1. Écrivez dans l'éditeur « `geany` » un programme « `hello.c` » qui affiche « `hello` ».
2. Compilez le programme dans un terminal avec la commande « `gcc -Wall -o hello hello.c` ».
3. Corrigez les erreurs de syntaxe et recompilez.
4. Exécutez le programme avec la commande « `./hello` »

### 3.2 Hello World (printf/scanf)

Écrivez un programme permettant de demander l'âge de l'utilisateur et de l'afficher

### 3.3 Bibliothèque math.h

Écrire un programme qui va demander la partie réelle et la partie imaginaire d'un complexe de la forme $a + jb$. Vous écrirez une fonction qui permet de calculer le module de ce nombre complexe ainsi qu'une fonction qui écrira son argument et ainsi affichera sa représentation polaire de la forme $\rho e^{(j\theta)}$.

### 3.4 Tableau

— Écrire un programme qui demande la taille d'un tableau d'entiers (on aura déclaré au préalable un tableau d'entiers de 50 cases et s'assurera que la taille demandée est inférieure à cette limite de 50), puis remplir, case par case, et afficher le contenu du tableau ainsi que la somme de tous les éléments.

### 3.5 Lecture écriture dans un fichier

À l'aide d'un éditeur de textes, créer un fichier `nombres.txt` qui contient une liste de nombres entiers. Dans le fichier, chaque nombre doit être suivi par un retour à la ligne. Écrire un programme qui affiche les nombres du fichier, leur somme et leur moyenne.

## 4 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**

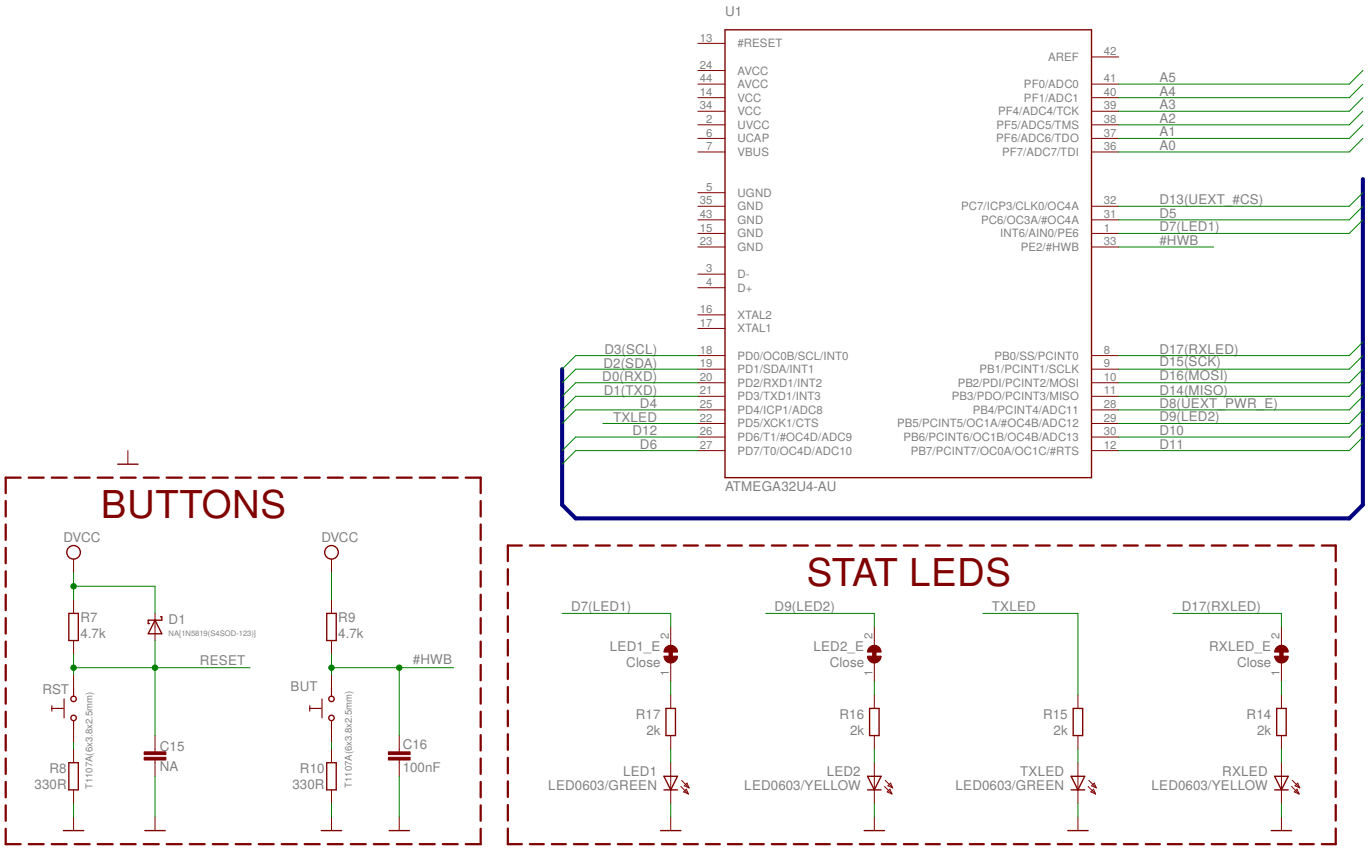


FIGURE 1 – Extrait du schéma de l'Olimexino-32U4

Le comportement des GPIO est décrit dans l'extrait de la Fig. 2 de la feuille descriptive (*datasheet*) de l'Atmega32U4 disponible à `www.atmel.com/Images/doc7766.pdf` (p.67) [1].

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){
```

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

**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

```
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`

## 4.1 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**.

## 4.2 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).

### 4.3 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

```
#include <avr/io.h> //E/S ex PORTB
#define F_CPU 16000000UL

void mon_delai(long duree)
{long k=0;
 for (k=0;k<duree;k++) {};
}

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

  while (1){
    PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
    mon_delai(0xffff);
  }
  return 0;
}
```

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.**

## 5 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`

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

#define F_CPU 16000000UL

#include "separe_delay.h"

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

  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.

# 6 Communication – appel à une bibliothèque

La liaison USB entre l'ordinateur et le microcontrôleur sert non seulement à alimenter le circuit mais permet aussi de communiquer. Une bibliothèque à cet effet est fournie, et fournit l'opportunité de lier notre programme avec ces nouvelles fonctionnalités.

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 fournissons une bibliothèque chargée de communiquer au travers de la liaison USB nommée `libVirtualSerial`. 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`.

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␣");
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
avr-gcc -mmcu=atmega32u4 -L. separe_delay.o VirtualSerial.o -o VirtualSerial.elf -lVirtualSerial
```

Le résultat de cette commande s'observe par `stty -F /dev/ttyACM0 57600 && cat < /dev/ttyACM0` (en pratique, une liaison sur USB native ne nécessite par de reconfigurer le *baudrate* par `stty`). Alternativement, un terminal connecté au port série tel que `minicom` offre les mêmes fonctionnalités avec une interface plus lourde.

**Modifier ce programme pour afficher la taille de variables de type `char`, `short`, `int` et `long`. En faire autant sur le PC. Que conclure quant à la gamme de valeurs stockées dans chaque type de variable? Qu'en conclure sur l'`int`? La fonction qui affiche la taille d'un type de variable est `sizeof()`.**

# 7 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

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

extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
extern FILE USBSerialStream;

int main(void){
  int val=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;
            aff[0]='0'+(val/16);aff[1]='0'+(val%10);aff[2]=0;
            fputs("F=\0", &USBSerialStream);fputs(aff, &USBSerialStream);fputs("\r\n\0", &USBSerialStream);
            delay(0xff);
            fprintf(&USBSerialStream,"\r\n");
            CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
            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);
```

```
25   while(1) {val=(PINF&0x01);
26             if (val!=ancien)
27                {cpt++;
28                 aff[0]='0'+(cpt/16);aff[1]='0'+(cpt%10);aff[2]=0;
29                 fputs("F=\0", &USBSerialStream);
30                 fputs(aff, &USBSerialStream);
31                 fputs("\r\n\0", &USBSerialStream);
32                }
33             ancien=val;
34             CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
35             CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
36             USB_USBTask();
37            }
38 }
```

**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.

# 8 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 4.3).

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*).

## 8.1 *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* [1] 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 #define delai_leds (16000*2)
5
6 void timer1_init()  // Timer1 avec prescaler=64 et mode normal
```

```
{TCCR1B |= (1 << CS11)|(1 << CS10);
}

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

  timer1_init();
  while(1)
    {if (TIFR1 & (1 << TOV1))
       {PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
        TIFR1 |= (1 << TOV1); // clear flag
       }
    }
}
```

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.

## 8.2 *CTC*

**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.

*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

```
#include <avr/io.h>
#define F_CPU 16000000UL

#define delai_leds (16000*2)

void timer1_init()  // Timer1 avec prescaler=64 et CTC (WGM=2)
{TCCR1B |= (1 << WGM12)|(1 << CS11)|(1 << CS10);
 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**

# 9 Communication – UART

Le port de communication asynchrone est probablement le périphérique le plus utile sur un microcontrôleur. Après avoir initialisé l'horloge qui définit le débit de données (*baudrate*) ainsi que la nature des transactions (nombre de bits par mot transmis, encapsulation), les caractères sont transmis dès que le registre d'émission est libre. L'universalité de ce mode de communication tient en sa simplicité : tout système numérique programmable propose ce port de communication, même sur des plateformes aussi complexes qu'un routeur wifi (`http://www.rwhitby.net/projects/wrt54gs`) ou une console de jeu (`http://nil.rpc1.org/psp/remote.html`).
Le périphérique du PC de transactions sur le port série asynchrone virtuel sur USB se nomme `/dev/ttyUSB0`.
**Qu'affiche le programme ci-dessous ? Quelle est la durée de transmission de chaque caractère ?**

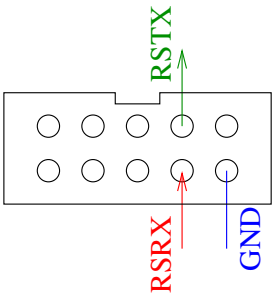


**FIGURE 5:** Branchement de la liaison RS232 sur connecteur UEXT (en tenant le circuit vers soi)

Listing 12 – Liaison UART en 9600 bauds

```
#include <avr/io.h>      // voir /usr/lib/avr/include/avr/iom32u4.h
#define F_CPU 16000000UL
#include <util/delay.h>  // _delay_ms

# define USART_BAUDRATE 9600

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

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

int main(void){
unsigned short baud;
char c=32;
  DDRB |=1<<PORTB5;
  DDRE |=1<<PORTE6;
  PORTB |= 1<<PORTB5;
  PORTE &= ~1<<PORTE6;

  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;

  while (1){
    PORTB^=1<<PORTB5;PORTE^=1<<PORTE6; // LED
    _delay_ms(100);

    uart_transmit(c);                  // affiche le message
```

```
40      c++; if (c==127) c=32;
41    }
42    return 0;
43  }
```

**Proposer une fonction chargée d'afficher une chaîne de caractères. Afficher la phrase "Hello World".**

**Proposer un programme qui reçoit des caractères sur le port série, et renvoie sur le port série le caractère suivant de l'alphabet de celui qui a été reçu.**

# Références


[1] Atmel, 8-bit Microcontroller with 16/32K Bytes of ISP Flash and USB Controller – ATmega16U4 & ATmega32U4, disponible à `www.atmel.com/Images/doc7766.pdf` (version 776F–AVR–11/10)