TP introduction au microcontrôleur 8 bits

É. Carry, J.-M Friedt

17 janvier 2021

Questions sur ce TP :

1. Quel est l'intérêt de l'outil `make` et de son fichier de configuration `Makefile`?
2. Quel est l'intérêt de définir le nom du programme par une variable dans une `Makefile`?
3. Quel est l'intérêt de séparer les fichiers contenant les fonctions liées à une même fonctionnalité du microcontrôleur?
4. Quelle option de `gcc` permet de lier un code à la bibliothèque statique `libtoto.a`?
5. Comment définir le chemin du répertoire contenant la bibliothèque `libtoto.a`?
6. Quel est le nom de l'interface du port série virtuel créé par Linux pour communiquer par USB avec le microcontrôleur lors de l'utilisation de la bibliothèque LUFA?

L'objet de cette séance de travaux pratiques est de

1. mieux comprendre le fonctionnement du compilateur `gcc` et en particulier ses options d'optimisation
2. organiser son code en modules compilés séparément, idéalement réutilisables (bibliothèques de fonctions)
3. automatiser la séquence de compilation de programmes séparés grâce à l'outil `make`
4. exploiter les connaissances acquises, en particulier sur `make`, pour exploiter la communication USB en se liant à la bibliothèque LUFA.

1 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 1 – 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.**

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 2 – 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){
    PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
    mon_delai(0xffff);
  }
  return 0;
}
```

Listing 3 – Fonction de retard : `separe_delay.c`

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

Listing 4 – Fichier d'entête de définition des fonctions : `separe_delay.h`

```
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 5 – Exemple de Makefile

```
all:  separe_blink.out

```

```
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 Compléments sur le `Makefile`

Afin d'éviter de modifier en plusieurs emplacements du `Makefile` qui servira tout au long de ce TP le nom du programme, nous introduisons un nouveau concept : la variable.

Une variable de `Makefile` se définit par VARIABLE=valeur et sera appelée par `$(VARIABLE)` (attention – la convention n'est pas la même que pour le *shell*[1] qui sépare le nom de variable du $ par des accolades au lieu de parenthèses.

Listing 6 – Exemple de `Makefile` avec variable.

```
1  MCU=atmega32u4
2  REP=$(HOME)/enseignement/ufr/platforms/Atmega32/
3  #TARGET=gpio_int
4  #TARGET=timer_ovfirq
5  #TARGET=timer_irq
6  #TARGET=wdt
7  TARGET=wdt_rs232
8  #TARGET=input_capture
9  #TARGET=input_capture_sendai
10 #TARGET=PRN_villedavray_header
11 #TARGET=switch_sendai
12 #TARGET=tmp
13 #TARGET=usart_int
14 #TARGET=gpr
15
16 all:  $(TARGET).hex
17
18 $(TARGET).hex: $(TARGET).out
19    avr-objcopy -Oihex $(TARGET).out $(TARGET).hex  # for MS-Windows (winavr requires .hex)
20    avr-objdump -dSt $(TARGET).out > $(TARGET).lst
21
22 $(TARGET).out: $(TARGET).o
23    avr-gcc  -L$(REP)/VirtualSerial/ -mmcu=$(MCU) -o $(TARGET).out $(TARGET).o -lVirtualSerial
24
25 $(TARGET).o: $(TARGET).c
26    avr-gcc -Wall -mmcu=$(MCU) -I$(REP)/VirtualSerial/ -I$(REP)/lufa-LUFA-140928/ -DF_USB=16000000UL \
27    -std=gnu99 -Os -c $(TARGET).c
28    avr-gcc -Wall -mmcu=$(MCU) -I$(REP)/VirtualSerial/ -I$(REP)/lufa-LUFA-140928/ -DF_USB=16000000UL \
29    -std=gnu99 -Os -S $(TARGET).c
30
31 flash: $(TARGET).out
32    avrdude -c avr109 -b57600 -D -p $(MCU) -P /dev/ttyACM0 -e -U flash:w:$(TARGET).out
33
```

1. C. Newham, *Learning the bash Shell : Unix Shell Programming (In a Nutshell)*, Ed. O'Reilly (2005)

```
34 clean:
35    rm *.o $(TARGET).out
```

4 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 */
```

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

```
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 -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.

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)