Développement sur Texas Instruments Stellaris LM4F120

Texas Instruments propose une gamme de microcontrôleurs centrée sur le Cortex M4 nommée LM4F. Cette plateforme est supportée par `libopencm3` en se liant sur les bibliothèques fournies dans le sous-répertoire `lm4f`.

1. Lorsque nous abordons une nouvelle gamme de microcontrôleurs, que faut-il connaître en plus de la nature du cœur du processeur et de ses périphériques ? en particulier comment renseigner le point de démarrage du programme stocké en mémoire non-volatile du microcontrôleur ? Quel fichier contient cette information (extension et exemple dans l'arborescence de `https://github.com/jmfriedt/stm32`) ?
2. En consultant la datasheet du `LM4F120H5QR` qui équipe la carte d'évaluation qui nous concerne, par exemple disponible à `https://www.mouser.com/datasheet/2/405/lm4f120h5qr-124014.pdf`, identifier l'adresse de départ du programme et en quoi elle diffère de celle du STM32F410 dont la datasheet est disponible par exemple à `https://www.st.com/resource/en/datasheet/stm32f410cb.pdf`
3. D'après la datasheet du LM4F120H5QR, où placer le pointeur de pile ? Que se passe-t-il si nous reprenons la configuration par défaut du pointeur de pile proposée pour le STM32F410 ?

En clonant le dépôt `https://github.com/jmfriedt/stm32`, démontrer votre capacité à compiler et exécuter le programme.

4. La compilation échoue à cause d'un fichier manquant. En s'inspirant d'un des fichiers fourni par le dépôt, proposer le fichier identifié en question 1 pour respecter les contraintes de la question 2. Ce fichier est nécessaire pour passer à la question suivante.
5. Compiler le programme fourni en exemple par le `Makefile.stellaris`. Il se pourrait qu'il faille ajuster les chemins pointant sur `libopencm3` dans `/home/jmfriedt`. Exécuter dans l'émulateur ARM `qemu` fourni par la distribution GNU/Linux sous la nomenclature `qemu-system-arm` appelé par
   `qemu-system-arm -machine lm3s6965evb -nographic -serial mon:stdio -kernel exec.elf`
   Noter que nous quittons `qemu` par `CTRL-a` puis `x`. Que fait le programme d'exemple ?
6. Modifier le programme proposé pour afficher le message "Bonjour le monde" : démontrer dans l'émulateur (il en sera de même pour toutes les questions qui suivent).
7. Modifier le programme proposé ci-dessus pour afficher le contenu d'une variable `s=0x12AB` codée sur 16 bits. Comment a été définie cette variable ? Démontrez que votre variable a la bonne taille.
8. Modifier le programme proposé pour afficher le contenu d'une variable `l=0x12AB56` codée sur 32 bits. Comment a été définie cette variable ? Démontrez que votre variable a la bonne taille.
9. Afficher l'emplacement mémoire où cette variable `l` a été stockée. Est-ce que cet emplacement est cohérent avec la réponse à la question 3 ?
10. Modifier la définition de la variable ci-dessus pour la placer sur le tas : comment faire ?
11. Afficher l'emplacement mémoire de la variable ainsi nouvellement définie. Est-ce que cet emplacement est cohérent avec la réponse à la question 3 ?

Nous considérons un programme qui manipule un tableau, de la forme

```
#include "uart.h"
int main();

void procedure(int *i)
{int k;
 char j=0x55;
 char l=0xAA;
 for (k=0;k<5;k++)
  {i[k]=k;}
}
```

```
int main()
{int i[3];
 int j=0,k=1;
 clock_setup();
 init_gpio();
 usart_setup();

 procedure(i);
 affshrt(j); mon_putchar(' ');
 affshrt(k); mon_putchar('\n');
}
```

12. Ce programme s'achève en affichant les valeurs de `j` et `k` égales à certaines valeurs. Quelles sont-elles ? Sont-elles cohérentes avec leur valeur d'initialisation ? Justifier la cause de votre observation.
13. Afficher l'emplacement en mémoire de la fonction `main()`. Cette position est-elle cohérente avec la réponse à la question 2 ?
14. En supposant que `k` a été placée sur la pile dans la procédure, afficher le contenu de la pile à $\pm$ 4 emplacements mémoire de part et d'autre de l'emplacement de `k`
15. Lorsqu'un programme saute dans une fonction ou une procédure, comment se rappelle-t-il comment continuer son exécution lorsque la fin de la fonction ou procédure est atteinte ?
16. En analysant la sortie de la question 14, pouvez vous identifier le mécanisme que vous venez de décrire à la question précédente ?
17. En affichant le code assembleur (`arm-none-eabi-objdump -dSt executable.elf`), est-ce que vous trouvez une explication cohérente à la réponse à la question précédente ?
18. Ajouter dans la procédure la définition d'un tableau `m` d'entiers codés sur 32 bits de 3 éléments. Remplacer l'assignation dans la boucle sur `k` vers la variable `i` par une assignation dans `m`. Exécuter le programme : que constatez vous sur l'affichage de `j` et `k`.
19. Quelle est à votre avis la cause de votre observation à la question précédente ?
20. Quelle aurait été la conséquence si un système d'exploitation supervisait l'exécution du programme ?

Corrections :

1. La carte mémoire (*memory map*) définit l'emplacement des divers blocs mémoire, en particulier la RAM sur laquelle nous placerons la pile et le tas, ainsi que la mémoire non-volatile (*flash* habituellement) qui contient les instructions (opcodes) du programme.

   Le point de démarrage est en début de mémoire non-volatile, soit pointée par le vecteur d'interruption *reset* soit par convention de l'initialisation du PC. Le fichier de configuration de l'éditeur de liens `ld` contient ces informations.

2. Page 90, table 2-4 de la documentation technique de Texas Instruments indique que la flash commence en 0x00000000. Page 44, figure 10 de la documentation technique de ST indique que la flash commence en 0x08000000.

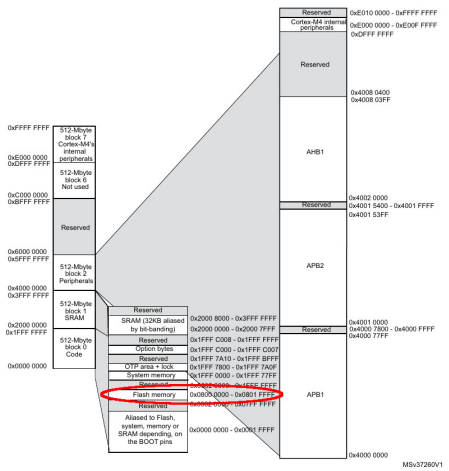
   Texas Instruments :

   Table 2-4. Memory Map

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x0003.FFFF	On-chip Flash	503
0x0004.0000	0x00FF.FFFF	Reserved	-
0x0100.0000	0x1FFF.FFFF	Reserved for ROM	489
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	488
0x2000.8000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	488
0x2210.0000	0x3FFF.FFFF	Reserved	-
FiRM Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	738
0x4000.1000	0x4000.1FFF	Watchdog timer 1	738
0x4000.2000	0x4000.3FFF	Reserved	-
0x4000.4000	0x4000.4FFF	GPIO Port A	622
0x4000.5000	0x4000.5FFF	GPIO Port B	622
0x4000.6000	0x4000.6FFF	GPIO Port C	622

   ST :

   Figure 10. Memory map



3. La pile se place en fin de la RAM, à l'adresse 0x20007FFFF. Si nous utilisons la configuration par défaut du STM32, à savoir l'adresse de départ +8 KB=0x1FFF (STM32F1) ou +26 KB=0x67FF (STM32F4), nous perdrons la partie de la mémoire excédentaire par rapport aux 32 KB disponibles sur le microcontrôleur Texas Instruments.
4. Modifier le fichier d'édition de lien `.ld` (option `-T` de la compilation par `gcc`) pour contenir
   `rom (rx) : ORIGIN = 0x00000000, LENGTH = 128K`
5. `make -f Makefile.stellaris`
6. Le programme

```
1 int main()
2 {clock_setup();
3  init_gpio();
4  usart_setup();
5  mon_puts("Bonjour le monde\n");
6 }
```

   compilé par `make` résoud le problème :

```
$ make run
qemu-system-arm -cpu cortex-m3 -machine lm3s6965evb -nographic -vga none -net none -serial mon:s
qemu-system-arm: warning: nic stellaris_enet.0 has no peer
Bonjour le monde
```

7. Une variable sur 16 bits se définit par un `short s=0x12AB;` et s'affiche par

```
1 void affchar(char c)
2 {char tmp;
3  tmp=c>>4;
4  if (tmp<10) mon_putchar('0'+tmp); else mon_putchar(tmp+'A'-10);
5  tmp=c&0xf;
6  if (tmp<10) mon_putchar('0'+tmp); else mon_putchar(tmp+'A'-10);
```

```
}

void affshrt(short s)
{affchar(s>>8);
 affchar(s&0xff);
}
```

La fonction `sizeof(s)` indique la taille de la variable et fournit `0002`.

8. Une variable sur 32 bits se définit par un `long` et son contenu s'affiche par :

```
void afflng(long l)
{affshrt(l>>16);
 affshrt(l&0xffff);
}
```

La fonction `sizeof(l)` indique la taille de la variable et fournit `0004`.

9. Un pointeur est défini sur 32 bits et indique l'emplacement d'une variable. L'instruction `afflng(&l);` indique `2000FFD0` qui est cohérent avec la position de la variable au sommet de la pile, dans la RAM.
10. `static l` place `l` sur le tas.
11. Cette fois l'emplacement est `20000004` qui est cohérent avec le tas en début de pile.
12. L'affichage de `j` et `k` respectivement indique `0004 0003`. Ces valeurs sont incohérentes avec l'initialisation à 0 et 1 respectivement, en l'absence d'assignation de ces variables dans le programme ailleurs qu'à leur définition. La cause de cette valeur erronée est le dépassement dans l'assignation du tableau pointé par `i` lors de l'appel de la procédure : l'assignation s'effectue sur 5 emplacements mémoire alors que le tableau dans la fonction `main` ne contient que 3 éléments.
13. `afflng(&main);` indique `0000054D` qui se trouve bien dans la mémoire flash
14. dans la procédure :

```
afflng((&main));mon_putchar('\n');
for (k=-4;k<4;k++)
  {afflng(*(&k+k)); mon_putchar('␣');}
```

indique `2000FFB8 0000053F 0000000A 2000FFD4 00000000 55AA055D 2000FFD0 00000591`

15. Le Program Counter (PC) est empilé lors de l'appel à la fonction, pour être dépilé à la fin de l'exécution de la fonction.
16. `main()` se trouve en 0000054D qui laisserait penser que la valeur du PC empilée est 00000591
17. `objdump` indique

```
58a:   4618            mov     r0, r3
58c:   f7ff ffaa       bl      4e4 <procedure>
590:   6978            ldr     r0, [r7, #20]
```

qui est bien cohérent avec une adresse de retour en 0x591 (rappel : en mode `thumb`, les adresses des opcodes sont impaires)

18. On ajoute `int m[3];` après la définition de `l` et la boucle sur `k` assigne maintenant `m[k]=k;`. L'affichage en fin de `main` de ne se fait plus.
19. L'assignation dans `m` écrase la pile contenant l'adresse de retour du PC : l'affichage de `j` et `k` ne se fait plus car la fin de la procédure ne retourne plus dans `main` pour en achever l'exécution.
20. Le programme a toutes les changes de tenter d'accéder à un segment mémoire qui ne lui est pas alloué, causant sa mort par le superviseur (ordonnanceur) du système d'exploitation. Sous GNU/Linux, le message est `Segmentation Fault`