TP1, 5TC-BED ez430-RF2500
Prise en main, programmation, assembleur, debuggeur

September 26, 2019

But du TP

En cours, on vous a présenté l'architecture du processeur MSP430 et l'ensemble des périphériques associés à ce processeur sur les cartes "eZ430-RF2500". Ce TP a pour but de prendre en main les outils de programmation associés et de vous faire manipuler quelques éléments du micro-contrôleur, notamment les LEDs, et comprendre sur cet exemple simple le mécanisme de fonctionnement des micro-contrôleurs.

Pour la suite, étant donnée le peu de temps dont nous disposons pour nous approprier cette architecture, nous avons choisi l'approche pédagogique suivante: Un certain nombre de pilotes de périphériques (*driver*) vous sont fournis, nous vous guidons pour comprendre ces programmes afin de pouvoir ensuite les utiliser pour une programmation de plus haut niveau.

1 Installation de l'environnement de programmation

1.1 Activation de la chaîne de compilation

La programmation du eZ430-R2500 se fait en C, nous utiliserons le compilateur `mspgcc` qui est installé sur les machines du département. Si cette installation ne fonctionne pas, vous pouvez temporairement installer la chaîne de compilation à partir de Moodle (lien intitulé "Archive contenant la toolchain mspgcc4.6"). Vous pouvez par exemple le faire sur votre propre machine), voir les instructions en annexe A.

En 2018, Sur la distribution (open suze) déployée sur les machines du département, la DSI a déployé la chaîne de compilation msp430 dans le répertoire `/opt/msp430-toolchain`. Pour pouvoir l'executer il faut mettre à jours les variable `$PATH` et `$LD_LIBRARY_PATH`. Voici le contenu du fichier `env.sh` que vous trouverez sur Moodle, copiez le dans votre `.bashrc` ou récuperez le fichier `env.sh` et appelez le depuis votre `.bashrc`.

```
#!/bin/sh
export MSP430DIR=/opt/msp430-toolchain
#We must be in the toolchain directory
export PATH=${MSP430DIR}/bin:${MSP430DIR}/bin/bin:$PATH
export LD_LIBRARY_PATH=${MSP430DIR}/lib:${LD_LIBRARY_PATH}
```

1.2 Récupération des sources des drivers du MSP430

Les sources des programmes que nous allons charger sur la clef sont récupérables via Moodle ainsi que les documentations associées. Récupérez l'archives `ez430-source.tgz` ("archive correcte des sources") et extrayez les dans votre répertoire courant. Par la suite, nous nommerons `$SRC` le répertoire où vous avez extrait les sources des programmes fournis.

Récupérez enfin sur Moodle le binaire de l'utiliraire appelé `ezconsole` qui nous permettra cd communiquer plus facilement avec le port série. Mettez le dans un répertoire qui est dans votre `$PATH`

2 Un premier programme : manipulation des LEDs

Compilation

- Placez vous dans le répertoire `$SRC/ez430-drivers`

- Tapez la commande `make`. Cela compile la libraire `libez430.a` à partir des drivers fournis dans le répertoire `src`
- Placez-vous dans le répertoire `$SRC/ez430-drivers/example/led_blink`
- Tapez la commande `make`.

Exercice 1 quel est l'exécutable produit? Où sont les sources compilées ? Ouvrez le fichier source `main.c` et essayez de comprendre le comportement du programme lorsqu'il s'exécute sur la clef. Ou est le code de la fonction `led_red_on()`

Téléchargement Cette étape consiste à télécharger le programme exécutable sur le module ez430

- Branchez sur un port USB la carte ez430.
- Lancez l'outil *mspdebug* avec en argument le *driver*, c'est à dire le type de périphérique que `mspdebug` doit s'attendre à trouver sur le port USB. Dans notre cas on exécutera: `mspdebug rf2500`
- Si tout se passe bien, vous vous retrouvez avec une invite :

  ```
  (msp-debug)
  ```

- **Si tout ne se passe pas bien** (ce qui ne sera probablement pas le cas en 2018), il faut faire une manipulation supplémentaire: décharger le module `cdc_acm` du noyeau. En effet, l'interface `usb` du msp430 contenant un driver de port série (chip FTDI), linux pense avoir affaire à un vieux modem série et charge donc le driver correspondant: `cdc_acm`, il faut donc le décharger du noyeau, la DSI vous a normalement mis les droit pour cela, exécutez:

  ```
  sudo /usr/sbin/modprobe -r cdc_acm
  ```

  Vous devez alors pouvoir lancer `msp_debug`
- à l'invite du débogueur, tapez (à moduler en fonction du répertoire duquel vous avez lancé mspdebug, ici c'est si vous vous trouvez dans `$SRC/ez430-drivers/example/led_blink`):

  ```
  (msp-debug) prog bin/led_blink.elf
  (msp-debug) run
  ```

Exercice 2 Quel comportement observez-vous sur le eZ430 ?

Exercice 3 Utilisez le Makefile pour faire la commande de chargement de code `make download`

Exercice 4 Trouver le source des fonctions utilisées (`leds_init(),led_red_switch(),etc.`), trouver les fichiers `leds.h`, `clock.h`, `watchdog.h`.

Exercice 5 Certaines macros ne sont pas définie ici, en partant du fichier:
`/opt/msp430-toolchain/msp430/include/msp430f2274.h`, trouver la définition de `P1OUT`.

3 Debug embarqué: utilisation de gdb en remote

`mspdebug` permet de lancer l'execution du programme en mode "`gdb`", ce qui permet de le debugger avec `gdb` comme vous debuggez un programme natif sur votre PC: `mspdebug` ouvre un port sur lequel `msp430-gdb` se connecte.

Exercice 6 Lancer l'exécution de mspdebug en mode "`gdb`"
`mspdebug rf2500 "gdb"`
`mspdebug` se connecte à la clé `ez430` et attend une connection sur le port 2000:
`Bound to port 2000. Now waiting for connection`

Exercice 7 Lancer `msp430-gdb` et connectez le au port 2000:
`msp430-gdb`
Pour se connecter au port 2000:
`(gdb) target remote localhost:2000`
Constatez la connection coté `mspdebug`.
`gdb` s'arrète au début du boot:
`0x00008000 in ?? ()`
il n'a pour l'instant pas les information de débuging car on n'a pas chargé le binaire dans le process `msp430-dbg` (`msp430-dbg` a pris la main sur le MSP mais ne connait pas le binaire exécuté).

Exercice 8 Chargez le binaire dans `gdb`:
`(gdb) file bin/led_blink.elf`

Exercice 9 Débuggez : mettez un breakpoint à la fonction `main`, lancer l'execution avec `continue` (la commande `gdb run` ne fonctionne pas car le programme est déjà en cours d'exécution):

```
(gdb) b main
Breakpoint 1 at 0x8042: file src/main.c, line 29.
(gdb) c
Continuing.
Breakpoint 1, main () at src/main.c:29
29     watchdog_stop();
(gdb)
(gdb) s
watchdog_stop () at src/watchdog.c:15
15 watchdog_backup = WDTCTL & 0x00FF;
(gdb) s
16 WDTCTL = WDTPW | WDTHOLD;
(gdb)
etc...
```

4 Création d'un programme assembleur simple

Exercice 10 Créez un nouveau répertoire TP1 dans le repertoire example (a coté du repertoire led_blink, et retapez[1] dans un fichier `tp1.s` le programme suivant:

```
.section .init9

main:
        /* initialisation de la diode rouge et verte */
        mov.b #3, &0x22
        mov.b #0, &0x26
        /* eteindre la diode rouge */
        mov.b #0, &0x21
        /* allumer la diode rouge */
        mov.b #1, &0x21
loop:
        jmp loop
```

Exercice 11 Produisez un exécutablr avec la commande suivante:
`msp430-gcc -mmcu=msp430f2274 -mdisable-watchdog -o tp1.elf tp1.s`
faites un petit Makefile pour éviter de le retaper à chaque fois

Exercice 12 Depuis mspdebug, transférez votre programme sur la carte en utilisant la commande `prog tp1.elf`, puis lancez-le avec la commande `run`. Constatez que la diode reste toujours allumée (c'est normal, on ne l'éteint jamais). Interrompez l'exécution en appuyant sur `Ctrl+C`.

[1] Vous pouvez aussi essayer de copier-coller depuis le PDF, mais il faudra pas venir vous plaindre que ça marche pas (ce qui sera probablement le cas). Un des objectif de cette question est de retaper explicitement le code, c'est réellement formateur de retaper les exemples.

Exercice 13 Utilisez la commande `msp430-objdum` pour visualiser de manière plus conviviale votre binaire:
`msp430-objdump -zhd tp1.elf > tp1.lst`
Retrouvez votre fonction dans le code tp1.lst.

À savoir: assemblage VS éditions de liens

Pour passer d'un programme en langage assembleur à un programme exécutable, il faut réaliser deux opérations:

1. l'*assemblage* consiste à convertir un fichier texte contenant des instructions vers un fichier binaire contenant les même instructions, mais en langage machine. L'outil qui fait ça, l'assembleur, est typiquement nommé `as`, et permet de passer d'un fichier `bidule.s` à un fichier `bidule.o`. C'est ce qu'on vient de faire dans l'exercice précédent. Mais ce n'est pas fini: le programme consiste peut-être en plusieurs morceaux, qu'il faut maintenant coller ensemble.

2. l'*édition de liens* consiste à coller ensemble plusieurs fichiers `machin.o`, et à placer chacun d'entre eux aux bonnes adresses, par exemple pour s'assurer qu'ils ne se marchent pas les uns sur les autres. L'outil qui fait ça, l'éditeur de liens, est typiquement nommé `ld`, et produit un fichier `truc.elf`

Invoquer ces différents outils comme il faut avec les bonnes options est compliqué et souvent source d'erreur. Heureusement, il existe aussi une commande générique `gcc` qui est beaucoup plus simple d'usage, et qui se charge d'appeler `as` et `ld` dans le bon ordre et avec les bons arguments. Ainsi, vous pouvez obtenir directement un exécutable avec la commande suivante:

```
msp430-gcc -mmcu==msp430f2274 -mdisable-watchdog -o truc.elf truc.s
```

5 Debug simplifié: directement avec `mspdebug`

`mspdebug` permet de faire un peu de debug, c'est moins lourd que de lancer `msp430-gdb` mais les outils de debug sont plus limités.

Exercice 14 Dans `tp1.s`, déplacez les instructions d'allumage et d'extinction à l'intérieur de la boucle infinie, et exécutez de nouveau votre programme. Constatez que la diode reste encore toujours allumée. En réalité, elle clignote bien, mais trop rapidement pour notre œil. C'est parce que la boucle tourne trop vite ! La fréquence du CPU est de 1MHz, et chaque instruction prend une poignée de cycles d'horloge, donc notre boucle tout entière tourne à plus de 100kHz. Interrompez de nouveau l'exécution, et au lieu de la relancer avec `run`, utilisez cette fois la commande `step` qui exécute une seule instruction machine. (faites donc `help run` et `help step` au passage). Constatez qu'en exécutant ainsi le programme en *mode pas-à-pas*, on arrive maintenant à voir ce qui se passe. Au bout de combien de `step` rentrez vous dans la boucle `loop`? Quelle est l'adresse en mémoire du label `loop`?

Vous allez maintenant devoir modifier votre programme. Pour la syntaxe ASM, aidez-vous des explications qui sont données dans les deux encadrés p.5 et p.6. Pour la mise au point, utilisez mspdebug. En plus des commande qu'on a vues jusqu'ici, vous aurez peut-être besoin de la commande `md` pour lire la mémoire, et de `setbreak` pour mettre des points d'arrêt. Commencez donc par taper `help md` et `help setbreak`.

Exercice 15 Modifiez votre programme afin de ralentir suffisamment la boucle infinie pour pouvoir observer le clignotement à l'oeil nu. Pour cela, vous allez rajouter une boucle (ou plusieurs) qui incrémente une variable (par exemple que vous mettrez dans le registre R4) jusqu'à atteindre une certaine valeur, par exemple 20000. Vous utilisez les commande `mov`, `dec` (decrementer) et `jnz` (jump if non zero) Faites valider par un enseignant.

Exercice 16 Comment faire pour atteindre une valeur plus grande pour le compteur, par exemple 100000?

Exercice 17 Pour ceux qui ont finit plus tôt, étudiez l'exemple utilisant un bouton et essayer de commander la led) partir du bouton, toujours en assembleur.

Utile pour le TP: La syntaxe ASM du msp430

Vous avez déjà rencontré ce jeu d'instructions en TD, mais avec une syntaxe un peu simplifiée. On vous présente ici la syntaxe que vous allez devoir utiliser en TP.

Opérations La plupart des instructions est de la forme `OPCODE SRC, DST`. OPCODE est l'opération souhaitée, par exemple ADD, XOR, MOV, etc. La liste complète est donnée page suivante. SRC et DST indiquent les opérandes sur lesquels travailler. Chaque opérande est de l'une des formes suivantes:

- un nom de registre: `R7`, `R15`... (utilisez les numéros, pas de «`SP`» ni «`PC`» etc.)
- une constante immédiate: `#42`, `#0xB600`...
- une case mémoire désignée par son addresse: `&1234`, `&0x3100`...

Par exemple, l'instruction `ADD &1000, R5` calcule la somme de R5 et de la valeur contenue dans la case d'adresse 1000, et range le résultat dans R5. Attention, certaines combinaisons n'ont pas de sens, et seront rejetées par l'assembleur avec un message d'erreur. Par exemple l'instruction `MOV R8, #36` ne veut rien dire.
Certaines instructions travaillent sur un seul opérande, et ont donc une syntaxe légèrement différente. Par exemple `INV DST` inverse chacun des bits de DST, ou `CLR DST` met DST à zéro. Reportez-vous à la liste page suivante pour plus de détails, et/ou à la doc: msp430x4xx.pdf page 61 et suivantes.

Drapeaux Certaines instructions, notamment les opérations arithmétiques et logiques, modifient le registre d'état (`R2`, cf encadré page ??), en particulier les drapeaux Z, N, C, V:

- Z est le *Zero bit*. Il passe à 1 lorsque le résultat d'une opération est nul, et il passe à 0 lorsqu'un résultat est non-nul.
- N est le *Negative bit*. Il passe à 1 lorsque le résultat d'une opération est négatif (en complément à deux) et il passe à 0 lorsqu'un résultat est non-négatif.
- C est le *Carry bit*. Il passe à 1 lorsqu'un calcul produit une retenue sortante, et il passe à 0 lorsqu'un calcul ne produit pas de retenue sortante.
- V est le *Overflow bit*. Il est mis à 1 lorsque le résultat d'une opération arithmétique déborde de la fourchette des valeurs signées (en complément à deux), et à 0 sinon.

La liste page suivante détaille l'effet de chaque instruction sur les quatre drapeaux: un tiret lorsque le drapeau n'est pas affecté, un 1 ou un 0 lorsque le drapeau passe toujours à une certaine valeur, et une étoile lorsque l'effet sur le drapeau dépend du résultat.

Sauts conditionnels Les instructions de branchement sont de la forme `JUMP label`. Regardez par exemple le programme page 3. Le saut peut être soit inconditionnel (instruction `JMP`), soit soumis à une condition sur les drapeaux. Par exemple, l'instruction `JNZ label` est un *Jump if Non-Zero*: elle sautera vers *label* si et seulement si le bit `Z` est faux.

Opérandes «word» ou «byte» Chaque instruction peut travailler sur des mots de 16 bits, ou sur des octets. Il faut pour cela on préciser `OPCODE.B`. Par exemple, l'instruction `MOV.B R10, &42` copie les 8 bits de poids faible de R10 vers l'octet situé à l'adresse 42, alors que l'instruction `MOV R10, &42` copie tout le contenu de R10 vers les deux octets situés aux adresses 42 et 43[a]. Reportez-vous à l'encadré page ?? pour plus de détails.

[a] Précision: les 8 bits de poids faible vont en 42, et les 8 bits de poids fort vont en 43. On dit que le msp430 est de type *little-endian*. Allez lire `https://fr.wikipedia.org/wiki/Endianness` si c'est la première fois que vous voyez ce mot.

Extrait de la documentation: msp430x4xx.pdf page 115

Mnemonic		Description	Operation	V	N	Z	C
ADC(.B)	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src,dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src,dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src,dst	Clear bits in destination	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	Set bits in destination	src .or. dst → dst	–	–	–	–
BIT(.B)	src,dst	Test bits in destination	src .and. dst	0	*	*	*
BR	dst	Branch to destination	dst → PC	–	–	–	–
CALL	dst	Call destination	PC+2 → stack, dst → PC	–	–	–	–
CLR(.B)	dst	Clear destination	0 → dst	–	–	–	–
CLRC		Clear C	0 → C	–	–	–	0
CLRN		Clear N	0 → N	–	0	–	–
CLRZ		Clear Z	0 → Z	–	–	0	–
CMP(.B)	src,dst	Compare source and destination	dst – src	*	*	*	*
DADC(.B)	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src,dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)	dst	Decrement destination	dst – 1 → dst	*	*	*	*
DECD(.B)	dst	Double-decrement destination	dst – 2 → dst	*	*	*	*
DINT		Disable interrupts	0 → GIE	–	–	–	–
EINT		Enable interrupts	1 → GIE	–	–	–	–
INC(.B)	dst	Increment destination	dst +1 → dst	*	*	*	*
INCD(.B)	dst	Double-increment destination	dst+2 → dst	*	*	*	*
INV(.B)	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		–	–	–	–
JEQ/JZ	label	Jump if equal/Jump if Z set		–	–	–	–
JGE	label	Jump if greater or equal		–	–	–	–
JL	label	Jump if less		–	–	–	–
JMP	label	Jump	PC + 2 x offset → PC	–	–	–	–
JN	label	Jump if N set		–	–	–	–
JNC/JLO	label	Jump if C not set/Jump if lower		–	–	–	–
JNE/JNZ	label	Jump if not equal/Jump if Z not set		–	–	–	–
MOV(.B)	src,dst	Move source to destination	src → dst	–	–	–	–
NOP		No operation		–	–	–	–
POP(.B)	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	–	–	–	–
PUSH(.B)	src	Push source onto stack	SP – 2 → SP, src → @SP	–	–	–	–
RET		Return from subroutine	@SP → PC, SP + 2 → SP	–	–	–	–
RETI		Return from interrupt		*	*	*	*
RLA(.B)	dst	Rotate left arithmetically		*	*	*	*
RLC(.B)	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC		Set C	1 → C	–	–	–	1
SETN		Set N	1 → N	–	1	–	–
SETZ		Set Z	1 → C	–	–	1	–
SUB(.B)	src,dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		–	–	–	–
SXT	dst	Extend sign		0	*	*	*
TST(.B)	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

Remarque chacune de ces instructions est documentée en détail dans la doc (msp430x4xx.pdf page 61 et suivantes). N'hésitez pas à vous y reporter si vous avez besoin de précisions.

A Annexe: Installation de la chaîne de compilation `mspgcc` sur Linux

A.1 A partir des binaire présents sur Moodle

:

- Récuperez l'archive sur Moodle (lien "Archive contenant la toolchain mspgcc4.6")
- Choisissez un répertoire, dans votre home ou dans `/etc/local` si vous avez les droits admin
- `tar xzvf toolchain.tar.bz2`
- Déscendez dans le répértoire `toolchain` et préparez l'environnement `cd msp430-toolchain` `source env.sh`
- N'oubliez par d'exécuter `env.sh` dans tous les shell où vous travaillez :

A.2 À partir des paquets Ubuntu

`sudo apt-get install binutils-msp430 gcc-msp430 msp430-libc mspdebug`
Assurez vous que la version de mspgcc est bien supérieure ou égale à 4.6