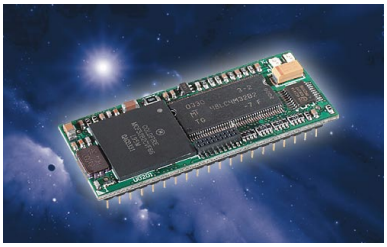
TP microcontrôleur 32 bits sous uClinux : le Coldfire

J.-M Friedt, 11 novembre 2008

Objectif de ce TP :
- présentation de uClinux, port de Linux pour système sans MMU
- présentation d'une plateforme supportant uClinux, ressources nécessaires
- méthode de développement : crosscompilation et NFS
- quelques exemples d'acquisitions et de communications



http://www.dilnetpc.com/dnp0065.htm

1 GNU/Linux

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 jamais comme administrateur.
- `/usr` contient les programmes locaux à un ordinateur : applications, entêtes de compilation (`/usr/include`), sources du système (`/usr/src`)
- `/lib` contient les librairies du système d'exploitation
- `/proc` 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.

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`
- `man`, manuel, est la commnde la plus importante sous Unix, qui donne la liste des options et le mode d'emploi de chaque commande.

2 Cross-compilation et exécution du binaire depuis le Coldfire

L'objectif de la cross-compilation est de générer un binaire exécutable sur architecture Motorola 68000 dont est issue le processeur Coldfire. La toolchain permettant cette compilaton a été obtenue prête à l'emploi á `http://www.uclinux.org/pub/uClinux/dist/`.

Une application est compilée sur PC au moyen de `m68k-elf-gcc` au lieu de `gcc`. Une fois l'exécutable généré, nous désirons y accéder depuis le Coldfire (ce binaire contient des opcodes 68000). uClinux fournit la communication UDP sur IP ainsi que le support de systèmes de fichiers ⇒ utilisation de NFS (*Network File System*).

1. se connecter depuis le PC sur le Coldfire :
   `telnet 192.168.0.126`
2. depuis le Coldfire, monter l'arborescence du PC :
   `mount IP_du_PC:/home /mnt`

Le répertoire `/home` du PC est désormais accessible sur le Coldfire dans `/mnt`. On peut y exécuter le programme qui y a été compilé.

> Noter que sous réserve d'accès au matériel, le *même* code source C pourra être compilé pour PC ou Coldfire et exécuté sur la plateforme adéquate.

Afin de compiler un programme, rentrer dans le répertoire approprié, vérifier la présence d'un fichier `Makefile`. Tapper `make clean` puis `make` pour compiler l'exécutable.

Prenons l'exemple le plus simple

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {printf("Hello World, sqrt(2)=%f \n",sqrt(2));return(0);}
```

Deux observations s'imposent :

- quelque soit l'environnement d'exécution, la sortie standard (`stdout`) est redéfinie de façon appropriée par le système (redirection vers un terminal ou le port série).
- le programme n'est plus une boucle infinie mais rend la main au système d'exploitation en fin d'exécution.

Ce programme se compile par

```
m68k-elf-gcc -m5307 -DCONFIG_COLDFIRE -Os -g -fomit-frame-pointer -Dunix -D__uClinux__ \
-DEMBED -fno-builtin -msep-data -Wl,-elf2flt -Wl,-move-rodata -o hello hello.c -lm
```

Noter la présence de `-Wl,-elf2flt` qui indique au compilateur de convertir le fichier au format ELF en format BFLT plus simple à charger sur un système embarqué :

```
$ file hello
hello: BFLT executable - version 4 gotpic
```

tandis que sur PC :

```
$ gcc -o hello hello.c -lm
$ file hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux
2.6.8, dynamically linked (uses shared libs), for GNU/Linux 2.6.8, not
stripped
```

Le Makefile fournit un outil souple pour éviter de se remémorer ces options :

```
PATH := $(PATH):/usr/bin:/usr/local/bin
CC = m68k-elf-gcc
EXEC = hello
OBJS = hello.o
UCPATH = /home/jmfriedt/uclinux/uClinux-dist
CINCL = -I$(UCPATH)/uClibc/include -I$(UCPATH)/lib/libjpeg -fno-builtin -msep-data -I$(UCPATH)/linux-2.4.x/include
LDLIB = -L$(UCPATH)/uClibc/. -L$(UCPATH)/uClibc/lib -L$(UCPATH)/uClibc/libc \
        -L$(UCPATH)/uClibc/libpthread -L$(UCPATH)/lib/libjpeg
CFLAGS = -m5307 -DCONFIG_COLDFIRE -Os -g -fomit-frame-pointer -Dunix -D__uClinux__ -DEMBED $(CINCL)
LDFLAGS = $(CFLAGS) -Wl,-elf2flt -Wl,-move-rodata $(LDLIB)

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS) -lc -lm
        chmod 777 $@

clean:
        rm -f $(EXEC) *.elf *.gdb *.o
```

Les variables d'environnement `UCPATH`, `CINCL` et `LDLIB` contiennent respectivement le chemin vers le répertoire contenant l'arborescence uClinux, les chemins vers les fichiers d'entête et les chemins vers les librairies. Nous n'avons qu'à changer `OBJS` (les objets intermédiaires nécessaires à la compilation) et `EXEC` (le binaire exécutable final) pour chaque nouveau programme.

Noter que des librairies aussi complexes que la gestion des threads (`libpthread`) et la compression jpeg sont disponibles.

Exercices :

1. lire et comprendre le fonctionnement du fichier `Makefile`
2. compiler manuellement et par Makefile le programme d'exemple afin de l'exécuter sur PC et sur Coldfire

3 Accés au matériel

Le coldfire 5282 qui équipe la carte DNP/5280 de SSV est équipé de nombreux périphériques : deux ports série, communication synchrone SPI et I2C, bus CAN, conversion analogique-numérique ... ainsi que des périphériques externes tels que le contrôleur ethernet. L'espace mémoire disponile, 16 MB de RAM et 8 MB de flash non volatile, est suffisant pour contenir un système linux embarqué fonctionnel avec des applications pouvant aller jusqu'au serveur web.

La grande différence entre la programmation sur ordinateur personnel et sur système embarqué est que l'interaction avec le matériel est beaucoup plus forte dans le second cas. Nous nous intéresserons donc en particulier à l'assignation des broches du circuit [1]. Pour ce qui nous intéressera, le port A se trouve aux broches 1 à 8, le port B de 9 à 16, le port C de 17 à 20 (4 bits). Pour retrouver l'orientation du composant, on note que l'ethernet se connecte entre 30 et 34. Le port A sert aussi d'entrée analogique [2] donc nous éviterons de nous en servir pour des applications numériques : PA4 à PA7 correspondent aux convertisseurs 0 à 3.

Deux grandes stratégies :

- en l'absence de distinction entre espace utilisateur et espace noyau (système sans MMU), on peut directement écrire à une adresse correspondant à un périphérique, comme on le ferait sur microcontrôleur.
- par soucis de portabilité du code, ou en présence d'un mode superviseur interdisant l'accès aux adresses des périphériques, nécessité d'un module noyau.

La société qui commercialise le circuit sur lequel nous travaillons propose un module noyau (qui a donc toutes les permissions d'accéter à toutes les adresses) permettant l'accès depuis l'espace utilisateur aux ports d'entrée sortie : `ssvhwa`.

La gestion des modules noyau nécessite 3 programmes :

- `lsmod` fournit la liste des modules chargés en mémoire
- `insmod module.o` charge en mémoire le module `module.o`
- `rmmod module` retire de la mémoire le module `module`, sous réserve de ne pas être utilisé par une programme utilisateur ni par par un autre module.

La communication entre l'espace noyau et l'espace utilisateur se fait au travers des *devices* situés dans `/dev`. Pour accéder aux fonctionnalités du module `ssvhwa.o`, ouvrir `/dev/ssvhwa` en lecture et écriture : `int d=open("/dev/ssvhwa", O_RDWR);` On pourra ensuite lire et écrire des valeurs au moyen des méthodes fournies par le module.

Exercices :

1. consulter le programme `ssvhwa.c` et identifier les méthodes de lecture et d'écriture, ainsi que leur mode d'accès.
2. consulter et comprendre le fonctionnement du module noyau `ssvhwa.o`
3. faire clignoter une diode depuis uClinux : le port B, bit 0, auquel est connecté la diode se trouve à l'adresse MCFBAR+0x001A001D (MCFBAR=0x40000000).

4 Lecture d'une valeur analogique

Comme dans tout microcontrôleur, l'accès aux périphériques se fait au travers de ports. Il faut connaître la fonction de chaque port pour y placer les bonnes valeurs.

Afin de convertir une tension analogique en mesure numérique, nous pourrions développer un module noyau dédié, mais pour le moment nous allons nous contenter de protoyper l'accès au convertisseur analogique-numérique au moyen de `ssvhwa`.

Comme sur un microcontroôleur, la principale difficulté tient en la lecture du protocole de conversion et en l'obtention des données acquises. La tâche est plus difficile sur des processeurs aussi évolués que le Colfire (en comparaison de microcontrôleurs “simples” 8 ou 16 bits) en ce que le matériel tente de décharger au maximum le processeur de tâches répétitives pour le laisser s'occuper du système d'exploitation et des programmes utilisateurs associés.

Dans le cas du Coldfire 5282, une queue de conversion programmable est implémentée pour récuperer automatiquement plusieurs valeurs sans intervention du processeur : la queue des opérations à réaliser se nomme `CCW` et le résultat de conversion se trouvera dans `RJURR`.

Il y a donc plusieurs registres à initialiser afin d'effectuer une conversion.

```
#define nbscan 1 // 64

int main (void)
```

[1] `http://www.dilnetpc.com/dnp0039.htm`
[2] `http://www.dilnetpc.com/dnp0040.htm`

```
{unsigned char data_b;
 unsigned short data_w,i;

 ssvhwa_open();
 ssvhwa_write8(DDRQA, 0x00);       // Port A[0-3] input
 ssvhwa_write16(QADCMCR, 0x0000);  // enable ADCs
 ssvhwa_write16(QACR0,0x7f);       // voir QACR0 pour definir QCK, default=19

// 52=AN52=PQA0, 53=AN53=PQA1, 62=(VH-VL)/2
 for (i=0;i<nbscan*2;i+=2)
     ssvhwa_write16(CCW+i,0x00C0|52); //queue1 read, 16 ck

while (1) {
 ssvhwa_write16(QASR0,0x0000);
 ssvhwa_write16(QACR1,0x2100);     // queue 1, single scan mode, software trig
 do {data_w=(ssvhwa_read16(QASR0));// attente de conversion ...
     printf("QASR0=%x ",data_w);
    } while ((data_w&0x8000)==0);
 printf("\n");
 printf("QASR1=%x\n",ssvhwa_read16(QASR1));
 printf("QACR1=%x\n",ssvhwa_read16(QACR1)); // check that scan is complete
 for (i=0;i<nbscan*2;i+=2)
   {data_w=ssvhwa_read16(RJURR+i);
    printf("%x=%d mV",data_w,(int)((float)data_w*3300./1.024));
  }
 printf("\n");
 }
 ssvhwa_close();return(0);
}
```

Exercices :

1. exécuter le programme et vérifier que la conversion fonctionne
2. modifier le programme afin de faire une moyenne sur 16 mesures

5 Communication TPC/IP

Un serveur est à l'écoute des connexions des clients. Dans le cas qui va nous intéresser ici, nous choisirons de placer le serveur sur le système embarqué (Coldfire) et le client est exécuté depuis le PC.

```
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>

#define MY_PORT         9999
#define MAXBUF          1024

int main()
{int sockfd;
 struct sockaddr_in self;
 char buffer[MAXBUF]={'a','b','c','d','e','\0'};

 sockfd = socket(AF_INET, SOCK_STREAM, 0);  // ICI LE TYPE DE SOCKET

 bzero(&self, sizeof(self));
 self.sin_family = AF_INET;
 self.sin_port = htons(MY_PORT);
 self.sin_addr.s_addr = INADDR_ANY;

 bind(sockfd, (struct sockaddr*)&self, sizeof(self));
 listen(sockfd, 20);

 while (1)
 {struct sockaddr_in client_addr;
  int clientfd,addrlen=sizeof(client_addr);

  clientfd = accept(sockfd, (struct sockaddr*)&client_addr, &addrlen);
  printf("%s:%d connected\n", inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
  send(clientfd, buffer, buffer, 0);
  send(clientfd, buffer, recv(clientfd, buffer, MAXBUF, 0), 0);
  close(clientfd);
 }
```

```
 close(sockfd);return(0);  // Clean up (should never get here)
}
```

Exercices :

1. compiler le serveur sur PC, l'exécuter et s'y connecter par `telnet 127.0.0.1 9999`. Que fait le serveur ?
2. compiler le serveur pour Coldfire, l'exécuter sur la plateforme embarquée, et s'y connecter par `telnet 192.168.0.126 9999`. Que fait le serveur ?

Références

[1] `http://www.unixgarden.com/index.php/embarque/premiers-pas-avec-la-carte-ssv-pnp5280`