Programmer le micro−contrôleur AVR avec GCC

par Guido Socher (homepage)

L´auteur:

Guido aime Linux, non seulement parce qu'il est amusant d'en découvrir toutes les possibilités, mais aussi grâce aux personnes impliquées dans son développement.

Traduit en Français par:
Iznogood
<iznogood/at/iznogood−factory.org>

Résumé:

Le micro−contrôleur 8 Bits RISC AVR d'Atmel est un micro−contrôleur très courant. C'est un simple circuit intégré avec EEPROM, mémoire, convertisseur Analogique−Numérique, nombreuses entrées et sorties numériques, timers, UART pour la communication par RS 232 et beaucoup d'autre choses.

Le mieux, c'est néanmoins un environnement de programmation complet disponible sous Linux : vous pouvez programmer ce micro−contrôleur en C en utilisant GCC. Dans cet article, je vais expliquer comment intaller et utiliser GCC. J'indiquerai aussi comment charger le logiciel dans le micro−contrôleur. Tout ce dont vous avez besoin est un micro−contrôleur AT90S4433, un quartz de 4 Mhz, du câble et quelques composants très bon marché.

Cet article ne constitue que l'introduction. Dans un autre article, nous construirons un afficheur LCD avec quelques boutons, des entrées analogiques et numériques, un "chien de garde" matériel (watchdog) et quelques LEDs. L'idée est d'avoir un panneau de contrôle à usage général pour un serveur Linux, mais nous devons d'abord apprendre à installer l'environnement de programmation et c'est le propos de cet article.

Installation logicielle : le nécessaire

Pour utiliser l'environnement de développement GNU C, il vous faut :

binutils−2.11.2.tar.bz2	Disponible sur : ftp://ftp.informatik.rwth−aachen.de/pub/gnu/binutils/ ou ftp://gatekeeper.dec.com/pub/GNU/binutils/
gcc−core−3.0.3.tar.gz	

	Disponible sur : ftp://ftp.informatik.rwth−aachen.de/pub/gnu/gcc/ ou ftp://gatekeeper.dec.com/pub/GNU/gcc/
avr−libc−20020106 .tar.gz	La bibliothèque C d'AVR est disponible sur : http://www.amelek.gda.pl/avr/libc/. Vous pouvez aussi la télécharger depuis ce serveur : download page
uisp−20011025.tar.gz	Le programmateur d'AVR est disponible sur : http://www.amelek.gda.pl/avr/libc/. Vous pouvez aussi le télécharger depuis ce serveur : download page

Nous installerons tous les programmes dans /usr/local/atmel, afin de les séparer de votre compilateur C Linux proprement dit. Créez ce répertoire par la commande :

```
mkdir /usr/local/atmel
```

Installation logicielle : GNU binutils

Le paquetage binutils fournit tous les utilitaires bas−niveau nécessaires à la construction de fichiers objets. Il inclut un assembleur AVR (avr−as), un éditeur de liens (avr−ld), des outils de gestion de bibliothèque (avr−ranlib, avr−ar), des programmes pour générer des fichiers objet intégrables dans l'EEPROM du micro−contrôleur (avr−objcopy), un désassembleur (avr−objdump) et des utilitaires tels que avr−strip et avr−size.

Tapez les commandes suivantes pour construire et installer les "binutils" :

```
bunzip2 −c binutils−2.11.2.tar.bz2 | tar xvf −
cd binutils−2.11.2
./configure −−target=avr −−prefix=/usr/local/atmel
make
make install
```

Ajoutez la ligne /usr/local/atmel/lib au fichier /etc/ld.so.conf et tapez la commande /sbin/ldconfig pour régénérer le cache d'édition de liens.

Installation logicielle : AVR gcc

avr−gcc sera notre compilateur C.

Lancez les commandes suivantes pour le construire et l'installer :

```
tar zxvf gcc−core−3.0.3.tar.gz
cd gcc−core−3.0.3
./configure −−target=avr −−prefix=/usr/local/atmel −−disable−nls −−enable−language=c
make
make install
```

Installation logicielle : la bibliothèque C AVR

La bibliothèque C est toujours en cours de développement. L'installation peut donc sensiblement changer entre les versions. Je vous recommande d'utiliser la version indiquée dans la table ci−dessus si vous voulez suivre les instructions pas à pas. J'ai testé cette version et elle fonctionne bien pour tous les programmes que nous écrirons dans cet article et les suivants.

Initilisation de quelques variables d'environnement (la syntaxe est pour bash):
export CC=avr−gcc
export AS=avr−as
export AR=avr−ar
export RANLIB=avr−ranlib
export PATH=/usr/local/atmel/bin:${PATH}

./configure −−prefix=/usr/local/atmel/avr −−target=avr −−enable−languages=c −−host=avr
make
make install

Installation logicielle : Le programmateur

Le logiciel de programmation charge le code objet spécialement préparé dans l'EEPROM de notre micro−contrôleur.

Le programmateur uisp pour Linux est très bon. Il peut ête utilisé directement à partir d'un Makefile. Vous ajoutez simplement une règle "make load" et vous pouvez simultanément compiler et charger le logiciel.

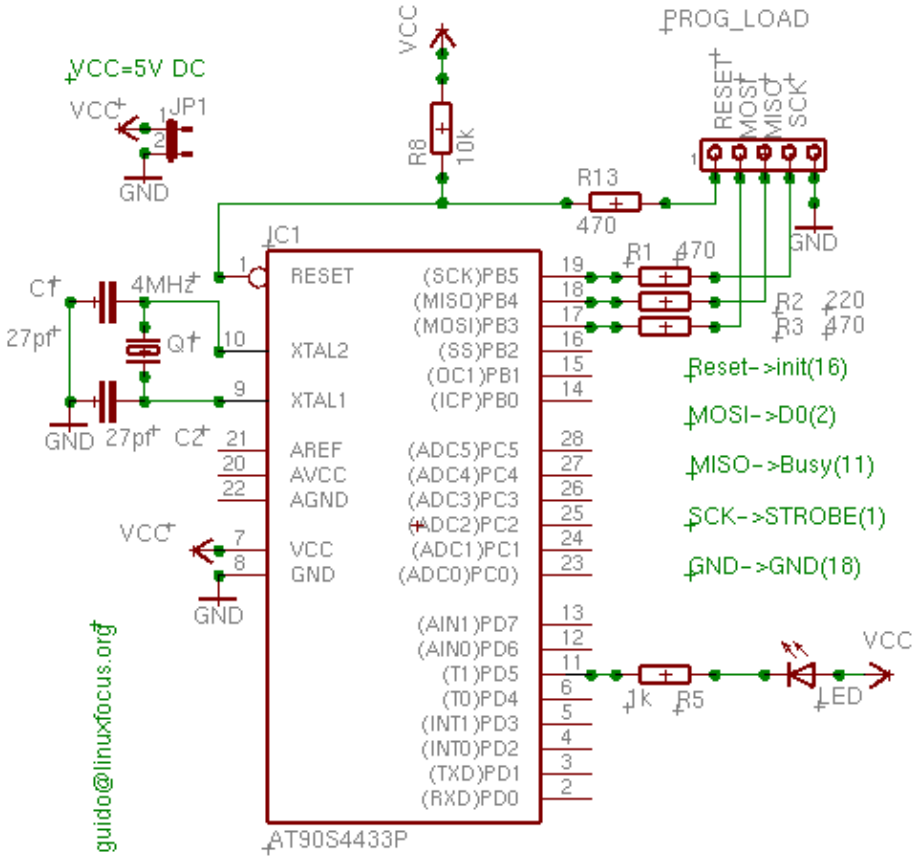
uisp est installé comme suit :

tar zxvf uisp−20011025.tar.gz
cd uisp−20011025/src
make
cp uisp /usr/local/atmel/bin

Un petit projet de test

Nous allons commencer par un petit circuit de test. Le but de ce circuit n'est que de tester notre environnement de développement. Nous pouvons l'utiliser pour compiler, télécharger et tester un programme. Le programme fera simplement clignoter une LED.

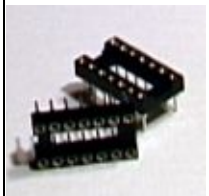
Je vous suggère de réaliser un petit circuit imprimé pour le micro−contrôleur. Vous pourrez l'améliorer ultérieurement pour faire vos propres expériences. C'est une bonne idée d'utiliser une carte test pour ce faire. Toutefois, ne tentez pas d'implanter l'AVR avec son quartz de 4 Mhz directement sur la carte de test. Il est plus sage d'utiliser quelques fils courts pour connecter les lignes d'entrées/sorties avec la carte de test car ces dernières ne sont pas conçues pour des circuits numériques rapides. Le quartz de 4 Mhz et les condensateurs doivent être physiquement très proches du micro−contrôleur.

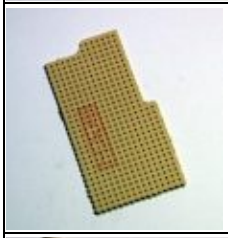
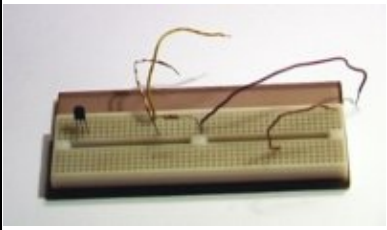


Les résistances sur le connecteur pour le programmateur ne sont pas réellement nécéssaires dans notre cas. Vous n'en aurez besoin que si vous avez prévu d'utiliser les lignes d'entrées/sorties du port B pour d'autres raisons.

Matériel nécessaire

Vous avez besoin des composants listés dans la table ci−dessous. Ils sont tous très courants et bon marché. Seul le micro−contrôleur est un peu plus cher, dans les 7.50 Euro. Bien que ce soit un micro−contrôleur très commun, il est possible qu'il ne soit pas disponible à la boutique du coin mais des distributeurs de matériel électronique un peu plus importants (comme www.reichelt.de (allemagne), www.conrad.de (allemagne), www.selectronic.fr (france), etc..., vous avez sûrement des sites identiques dans votre pays) doivent les avoir en stock.

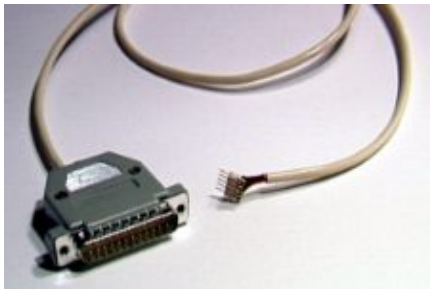
	1 processeur RISC AVR 8 bits AT90S4433 d'Atmel.
	support IC 2 x 14 broches ou support IC 1 x 28 broches de 7.5mm Le support 28 broches est un peu plus difficile à trouver. Habituellement, les supports 28 broches font 14 mm de large mais nous avons besoin d'un support de 7.5 mmm.

	1 résistance 10K (code couleur : marron, noir, orange) 3 résistances 470 Ohm (code couleur : jaune, violet, marron) 1 résistance 1K (code couleur : marron, noir, rouge) 1 résistance 220 Ohm (code couleur : rouge, rouge, marron) 1 quartz 4Mhz 2 condensateurs céramique 27pF
	Tout type de connecteur/support 5 broches pour le programmateur. J'achète habituellement ces connecteurs par bande et j'en retire 5.
	1 carte matricielle
	1 connecteur DB25 à brancher sur le port parallèle.
	1 LED
	Une carte test. Nous n'allons pas l'utiliser ici mais c'est très utile si vous voulez expérimenter l'AVR plus avant. Je suggère de laisser le micro−contrôleur avec le quartz et les condensateurs sur la carte matricielle et de connecter les lignes d'entrées/sorties par des câbles courts vers la carte de test.

En plus des composants ci−dessus, il vous faut une alimentation de 5 V, stabilisée électroniquement ou une pile de 4.5 V.

Construire le programmateur

L'AT90S4433 permet la programmation in situ.
Cela signifie que vous n'avez pas besoin de retirer le micro−contrôleur de la carte pour le programmer. Vous constaterez qu'il est possible d'acheter un programmateur tout prêt pour 50−150 Euro. Vous n'avez pas besoin d'investir une telle somme. Avec Linux, le logiciel uisp et un port parallèle libre, vous pouvez fabriquer un programmateur d'AVR très efficace et très simple . Ce n'est qu'un câble. Le câblage pour le programmateur doit être celui−ci :



Broche sur l'AVR	Broche sur le port parallèle
SCK (19)	Strobe (1)
MISO (18)	Busy (11)
MOSI (17)	D0 (2)
Reset (1)	Init (16)
GND	GND (18)

Le câble ne doit pas mesurer plus de 70 cm.

Écriture du logiciel

L'AT90S4433 peut être programmé en C à l'aide de gcc. Connaître l'assembleur de l'AVR peut être utile mais n'est pas nécessaire. La libc d'AVR est fournie avec un avr−libc−reference qui documente la plupart des fonctions. Harald Leitner a écrit un document contenant de nombreux exemples pratiques sur l'utilisation l'AVR et GCC (haraleit.pdf, 286Kb, à l'origine sur http://www.avrfreaks.net/AVRGCC/). Depuis le site d'Atmel's, (www.atmel.com, allez sur : avr products −> 8 bit risc−> Datasheets), vous pouvez télécharger la notice technique complète (localement : avr4433.pdf, 2361Kb) . Elle décrit tous les registres et l'utilisation du CPU.

Une chose à ne pas oublier à l'utilisation du 4433, c'est qu'il n'a que 128 octets de RAM et 4 Ko d'EEPROM. Cela signifie que vous ne devez pas déclarer de longues structures de données ou de longues chaînes de texte. Votre programme ne doit pas utiliser des appels de fonctions trop imbriqués ou la récursivité. Écrire une ligne comme
char string[90];
sera déjà de trop. Un entier fait 16 bits. Si vous avez besoin d'un entier plus petit, utilisez alors
unsigned char i; /* 0−255 */
Vous serez néanmoins surpris de la taille des programmes que vous pouvez écrire. C'est un processeur réellement très puissant !

Mieux vaut un exemple réel que toutes les théories. Nous allons écrire un programme qui fait clignoter la LED chaque 0.5 secondes. Ce n'est pas très utile mais suffisant pour commencer et pour tester l'environnement de développement et le programmateur.

```
void main(void)
{
    /* valide PD5 comme sortie */
    sbi(DDRD,PD5);
    while (1) {
        /* led on, pin=0 */
        cbi(PORTD,PD5);
        delay_ms(500);
```

```
        /* set output to 5V, LED off */
        sbi(PORTD,PD5);
        delay_ms(500);
    }
}
```

Le code ci−dessus démontre comme il est simple d'écrire un programme. Vous voyez simplement le programme principal. La fonction delay_ms est inclue dans le listing complet (avrledtest.c). Pour utiliser la broche PD5 comme sortie, vous devez définir le bit PD5 dans le registre de direction des données du port D (DDRD). Ensuite, vous pouvez initialiser PD5 à 0V par la fonction cbi(PORTD,PD5) (clear bit PD5) ou à 5V par sbi(PORTD,PD5) (set bit PD5). La valeur de "PD5" est définie dans io4433.h qui est inclue via io.h. Ne vous en préoccupez pas. Si vous avez déjà écrit des programmes pour des systèmes multi−utilisateurs ou multi−tâches tels que Linux, vous savez qu'il ne faut jamais programmer une boucle sans fin non bloquante. Ceci serait gaspiller du temps CPU et ralentirait beaucoup trop le système. Dans le cas de l'AVR, c'est différent. Nous n'avons pas plusieurs tâches et il n'y a pas d'autre programme en cours. Il n'y a pas non plus de système d'exploitation. Il est donc normal de l'occuper avec des boucles sans fin.

Compiler et charger

Avant de commencer, assurez−vous que /usr/local/atmel/bin figure bien dans la variable PATH. Si nécessaire, éditez .bash_profile ou .tcshrc et ajoutez :

export PATH=/usr/local/atmel/bin:${PATH} (pour bash)
setenv PATH /usr/local/atmel/bin:${PATH} (for tcsh)

Nous utilisons le port parallèle et uisp pour programmer l'AVR. Uisp utilise l'interface ppdev du noyau. Il est donc indispensable que les modules du noyau ci−dessous soient chargés :

/sbin/lsmod
parport_pc
ppdev
parport

Vérifiez qu'ils sont chargés par la commande /sbin/lsmod, sinon, chargez−les (en tant que root) par

modeprobe parport
modeprobe parport_pc
modeprobe ppdev

C'est bien d'exécuter ces commandes automatiquement pendant le démarrage. Vous pouvez les ajouter dans un script rc (i.e. pour Redhat /etc/rc.d/rc.local).
Pour utiliser l'interface ppdev en tant que simple utilisateur, root doit vous attribuer les droits d'écriture en lançant une fois la commande

chmod 666 /dev/parport0

Assurez−vous aussi qu'aucun démon d'impression ne fonctionne sur le port parallèle. Si vous en avez un, arrêtez−le avant de connecter le câble du programmateur. Maintenant, tout est prêt pour compiler et programmer notre micro−contrôleur.

Le paquetage de notre programme de test (avrledtest−0.1.tar.gz) inclut un fichier make. Tout ce que vous devez faire est de taper :

make
make load
Ceci compile et charge le logiciel. Je ne vais pas entrer dans le détail de toutes les commandes. Vous pouvez les voir dans le Makefile et elles sont toujours identiques. Je n'arrive pas moi−même à me les rappeler toutes. Je sais simplement que je dois utiliser "make load". Si vous voulez écrire un programme différent, remplacez toutes les occurrences de avrledtest dans le Makefile par le nom de votre programme.

Quelques "binutils" intéressants

Certains "binutils" sont plus intéressant que le processus de compilation proprement dit.

avr−objdump −h avrledtest.out

Montre la taille des différentes sections dans notre programme. .text est le code d'instruction et il est chargé dans l'EEPROM flash. .data contient les données initialisées telles que
static char str[]="hello";
et .bss contient les données globales non initialisées. Les deux sont à zéro dans notre cas. Le .eeprom contient les variables stockées dans l'eeprom. Je n'en ai jamais eu l'utilité. stab et stabstr contiennent les informations de débogage et ne seront pas dans l'AVR.

```
avrledtest.out:     file format elf32-avr

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000008c  00000000  00000000  00000094  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00800060  0000008c  00000120  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00800060  0000008c  00000120  2**0
                  ALLOC
  3 .eeprom       00000000  00810000  00810000  00000120  2**0
                  CONTENTS
  4 .stab         00000750  00000000  00000000  00000120  2**2
                  CONTENTS, READONLY, DEBUGGING
  5 .stabstr      000005f4  00000000  00000000  00000870  2**0
                  CONTENTS, READONLY, DEBUGGING
```

Vous pouvez aussi utiliser la commande avr−size pour obtenir ceci sous une forme plus compacte :

avr−size avrledtest.out

```
   text    data     bss     dec     hex filename
    140       0       0     140      8c avrledtest.out
```

Lors du travail avec AVR, vous devez vérifier que text+data+bss ne dépassent pas 4k et data+bss+stack (vous ne pouvez pas voir la taille de la pile, elle dépend du nombre d'appels de fonctions imbriquées) ne doivent pas dépasser de 128 octets.

Une autre commande intéressante

avr−objdump −S avrledtest.out

Elle génère un listing assembleur de votre code.

Conclusion

Vous en savez assez maintenant pour démarrer vos propres projets avec le matériel d'AVR et GCC. Il y aura d'autres articles dans LinuxFocus sur du matériel plus complexe et plus intéressant.

Références

- Libc et uisp : /www.amelek.gda.pl/avr/libc/
- GCC et binutils : ftp://gatekeeper.dec.com/pub/GNU/
- avrfreaks (attention, certaines personnes de ce site utilisent encore windows !?) : http://www.avrfreaks.net/
- l'assembleur tavrasm pour Linux : www.tavrasm.org
- webring AVR : R.webring.com/hub?ring=avr&list
- Versions pré−compilées de gcc : combio.de/avr/
- Tous les logiciels et les documents mentionnés dans cet article
- Le site web d'atmel : www.atmel.com/

2005−02−14, generated by lfparser_pdf version 2.51