Programmer les microcontrôleurs AVR avec GCC, libc 1.0.4

par Guido Socher (homepage)

L´auteur:

Guido aime Linux car c'est réellement un bon système pour développer son propre matériel.

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

Résumé:

Le microcontrôleur RISC 8 bits AVR d'Atmel est un microcontrôleur très commun. Ce microcontrôleur est un circuit avec une EEPROM, une Ram, un convertisseur Analogique vers Numérique, beaucoup d'entrées/sorties numériques, des compteurs, un UART pour les communications RS232 et bien d'autres choses.

Le meilleur est néanmoins qu'un environnement complet de programmation est disponible sous Linux: Vous pouvez programmer ce microcontrôleur en C en utilisant GCC.

J'ai déjà écrit un article en Mars 2000 sur le même sujet. Beaucoup de choses ont changé dans le développement d'avr−libc et le microcontrôleur AT90S4433 utilisé en 2002 n'est plus fabriqué par Atmel. Cet article est donc une mise à jour de celui de Mars 2002. Je vais utiliser libc−1.0.4 et le microcontrôleur ATmega8.

Cet article n'est qu'une introduction et dans une prochaine série d'articles, nous construirons des circuits intéressants mais basé cette fois sur l'ATmega8.

Introduction

Plusieurs personnes ont été intéressées dans la programmation d'un microcontrôleur, article écrit en 2002. Néanmoins la première étape pour installer l'environnement de développement et son lancement est le plus difficile. Si quelque chose ne fonctionne pas correctement, alors vous n'avez aucune aide pour connaître l'origine du défaut. Est−ce le câble de programmation? Est−ce le circuit qui est en cause? Une installation incorrecte? Un port parallèle désactivé dans le bios? Les modules de noyau pour ppdev mal compilés? Il peut y avoir un tas de raisons pour que les choses aillent mal.

Pour rendre l'entrée dans le monde excitant des microcontrôleurs plus facile, shop.tuxgraphics.org vous offre maintenant un CD bootable avec un manuel et le matériel de programmation. Tout ce dont vous avez besoin est de démarrer depuis le CD et tout est configuré, prêt à fonctionner. Aucune installation logicielle n'est nécessaire et rien n'est modifié sur votre ordinateur.

J'utilise moi aussi ce CD depuis longtemps car les circuits que je construis survivent à plusieurs générations de noyaux et d'installations de logiciels sur mon PC. Si, plus tard, je veux mettre à jour quelques logiciels de microcontrôleurs, je n'ai alors pas à me demander si le fonctionnement de mon environnement de développement sur mon Linux est encore fonctionnel. Je démarre simplement depuis le CD et tout est prêt à fonctionner.

Indépendement de ce CD, je vais expliquer l'installation de l'environnement de développement GCC avr dans les paragraphes suivants. Si vous avez le CD de tuxgraphics continuez alors avec le chapitre "Un petit projet de test".

Installation logicielle: Ce dont vous avez besoin

Pour utiliser l'environnement de développement GNU C, vous avez besoin des logiciels suivants:

binutils−2.15.tar.bz2	Disponibles depuis: ftp://ftp.gnu.org/gnu/binutils/ ou tout miroir. I.e: ftp://gatekeeper.dec.com/pub/GNU/binutils/
gcc−core−3.4.2.tar.bz2	Disponible depuis: ftp://ftp.gnu.org/gnu/gcc/ ou tout miroir. I.e: ftp://gatekeeper.dec.com/pub/GNU/gcc/
avr−libc−1.0.4.tar.bz2.tar	La bibliothèque C pour AVR est disponible depuis: http://savannah.nongnu.org/projects/avr−libc/
uisp−20040311.tar.bz2	Le logiciel de programmation AVR est disponible depuis: http://savannah.nongnu.org/projects/uisp

Nous allons installer tous les programmes sur /usr/local/avr. Cela permet de garder le programme séparé de votre compilateur C Linux. Créez ce répertoire avec la commande:

```
    mkdir /usr/local/avr

Vous pouvez déjà l'ajouter à votre PATH:
    mkdir /usr/local/avr/bin
    export PATH=/usr/local/avr/bin:${PATH}
```

Installation logicielle de: GNU binutils

Le paquet binutils fournit tous les utilitaires bas niveau nécessaires pour construire les fichiers objets. J'ai inclus un assembleur AVR (avr−as), un linker (avr−ld), des outils de traitement de bibliothèques (avr−ranlib, avr−ar), les programmes pour générer les fichiers objets chargeables dans l'EEPROM du microcontrôleur (avr−objcopy), un désassembleur (avr−objdump) et des utilitaires tels que avr−strip et avr−size.

Lancez les commandes suivantes pour construire et installer les binutils :

```
tar jxvf binutils-2.15.tar.bz2
cd binutils-2.15/
mkdir obj-avr
cd obj-avr
../configure --target=avr --prefix=/usr/local/avr --disable-nls
make

# comme root:
make install
```

Ajoutez la ligne /usr/local/avr/lib au fichier /etc/ld.so.conf et lancez la commande /sbin/ldconfig pour reconstruire le cache linker.

Installation logicielle: AVR gcc

avr−gcc sera notre compilateur C.

Lancez la commande suivante pour le construire et l'installer:

```
tar jxvf gcc-core-3.4.2.tar.bz2
cd gcc-3.4.2

mkdir obj-avr
cd obj-avr
../configure --target=avr --prefix=/usr/local/avr --disable-nls --enable-language=c

make

# comme root:
make install
```

Installation logicielle: La bibliothèque C AVR

La bibliothèque C est maintenant assez stable comparée à celle que j'ai présenté en Mars 2002.
Lancez la commande suivante pour la construire et l'installer:

```
tar jxvf avr-libc-1.0.4.tar.bz2.tar
cd avr-libc-1.0.4
PREFIX=/usr/local/avr
export PREFIX
sh -x ./doconf
./domake
```

```
cd build
#comme root:
make install
```

Installation logicielle: Le Programmateur

Le logiciel du Programmateur charge le code objet spécialement préparé dans l'EEPROM de notre microcontrôleur.

Le programmateur uisp pour Linux est très bon. Il peut être utilisé directement depuis un Makefile. Vous avez simplement à ajouter une règle "make load", ce qui vous permet de compiler et charger le logiciel en une passe.

uisp est installé comme suit:

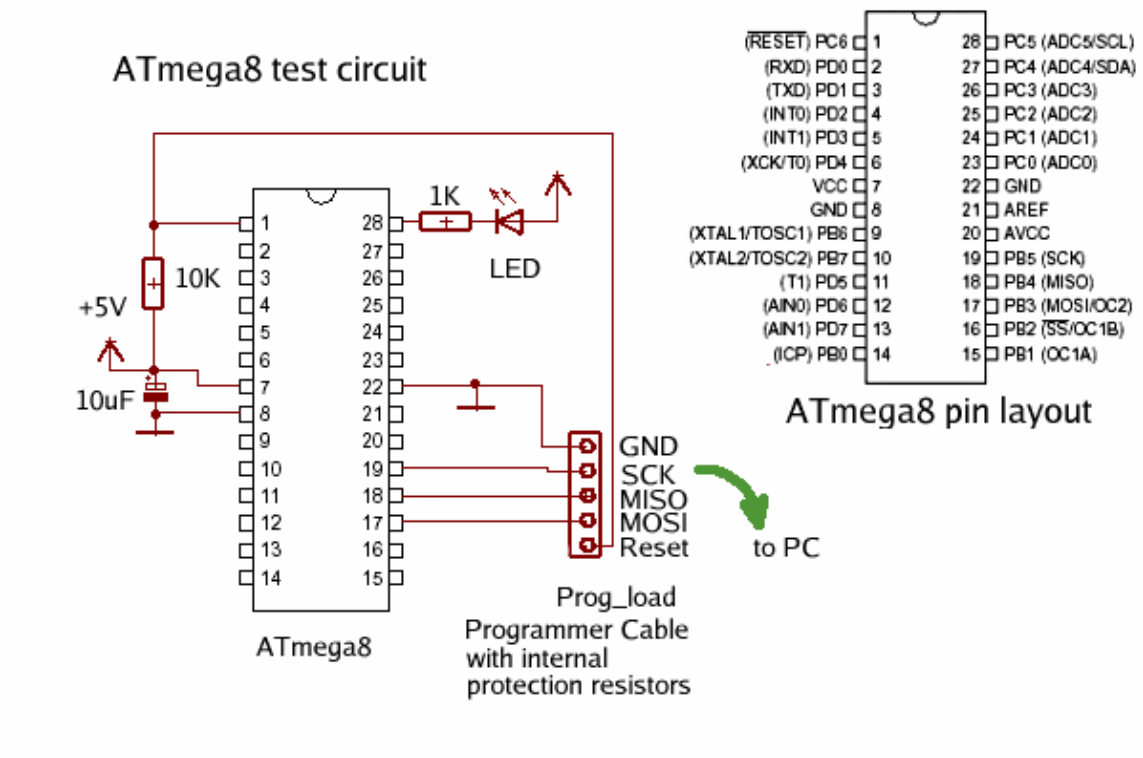
```
tar jxvf uisp-20040311.tar.bz2.tar
cd uisp-20040311
./configure --prefix=/usr/local/avr
make

# comme root:
make install
```

Un petit projet de test

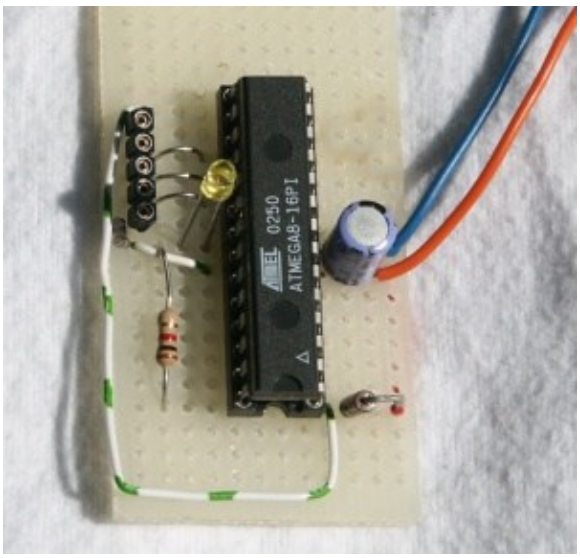
Nous allons débuter avec un petit circuit de test que vous pourrez étendre plus tard.

Ce circuit peut aussi être utilisé comme un simple environnement de test pour du logiciel plus complexe. Vous pouvez facilement tester le logiciel chargé et lier des capteurs ou un équipement de mesure.

Notre programme de test, présenté ici, ne fera que clignoter une LED.



Matériel nécessaire



Vous avez besoin des composants listés dans la table ci–dessous. Bien qu'il soit un microcontrôleur commun, il peut ne pas être disponible dans la boutique du coin mais il le sera chez des distributeurs d'électronique plus gros comme www.conrad.de (allemagne), www.selectronic.fr (france), digikey.com (US, CA), etc...
Vous pouvez aussi obtenir le kit complet ou seulement le microcontrôleur sur shop.tuxgraphics.org

1 x ATmega8 version DIP, processeur Atmel 8 bit Avr risc.
1 x 28 broches 7.5mm support de CI Le support 28 broches est un peu plus difficile à obtenir. Ces supports 28 broches sont souvent à 14mm de large mais nous avons besoin d'un support à 7.5mm.
1 x résistance 10K (code couleur: marron,noir,orange) 1 x résistance 1K (code couleur: marron,noir, rouge) 1 x condensateur électrolytique 10uF Quelques câbles

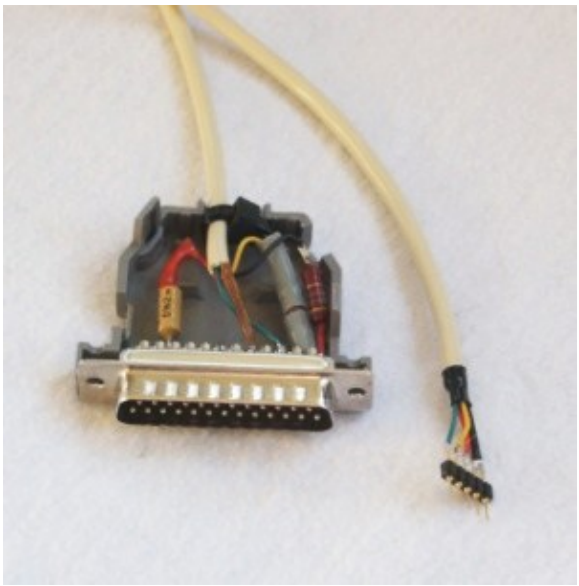
1 x LED une plaque pré−percée
Ce qui suit est nécessaire pour le programmateur (ce n'est pas nécessaire si vous avez le "kit de programmation AVR Linux de tuxgraphics): 1 x connecteur DB25 à brancher sur le port parallèle. Tout type de connecteur/support à 5 broches pour le programmateur. Je recommande l'utilisation de connecteurs tulipes (similaires aux supports de circuits) et d'en détacher 5 broches. 1 x résistance 220 Ohms (code couleur: rouge, rouge, marron) 3 x résistance 470 Ohms (code couleur: jaune, violet, marron)

En plus des composants ci−dessus, vous avez besoin d'une alimentation DC de 5V électroniquement stabilisée ou vous pouvez utiliser une pile de 4.5V comme alimentation.

Vous avez probablement remarqué que nous n'avons pas besoin de quartz. C'est parce que l'ATmega8 possède maintenant un oscillateur intégré. Cet oscillateur peut être utilisé lorsque la gestion du temps n'est pas vitale. Si vous voulez néanmoins construire un équipement de mesure précis ou si vous souhaitez utiliser l'interface UART/RS232 alors, vous avez besoin d'un quartz. Le type d'oscillateur utilisé peut être défini par les bits fusibles que vous pouvez modifier avec le programmateur. Par défaut (configuration d'usine), l'oscillateur d'1Mhz est actif.

Fabriquer le matériel du programmateur

Les microcontrôleurs AVR permettent la programmation in situ (ISP).
Cela signifie que vous n'avez pas besoin de retirer le microcontrôleur de la carte pour le programmer. Vous verrez que vous pouvez obtenir différents matériels de programmation pour 50−150 Euro. Néanmoins, avec Linux, il est aussi possible de fabriquer un programmateur très simple qui fait le travail. Vous avez besoin d'un port parallèle libre sur votre ordinateur et du câble suivant.



Notez que c'est un programmateur amélioré comparé à celui présenté dans l'article de Mars 2002. Nous plaçons les résistances de protection dans le programmateur. Cela permettra de préserver un peu d'espace et des composants sur la carte. Le câblage pour le câble de programmation doit être comme suit:

broche sur pcb	broche sur AVR	résistance de protection	broche sur port parallèle
5	Reset (1)	−−	Init (16)
4	MOSI (17)	470 Ohms	D0 (2)
3	MISO (18)	220 Ohms	Busy (11)
2	SCK (19)	470 Ohms	Strobe (1)
1	GND	−−	GND (18)

Le câble ne doit pas faire plus de 70cm.

Les résistances de protection peuvent être placées dans le connecteur comme montré sur l'image à droite.

Écrire des programmes

L'ATmega8 peut être programmé en C pur avec l'aide de gcc. La connaissance de l'assembleur AVR peut être utile mais n'est pas nécessaire.

La libc AVR est fournie avec avr−libc−user−manual−1.0.4.pdf (1139921 octets) qui documente toutes les fonctions disponibles en C. Sur le site d'Atmel (www.atmel.com, allez dans: avr products −> 8 bit risc−> Datasheets), vous pouvez y télécharger la documentation technique complète. Elle décrit tous les registres et comment utiliser le CPU.

Une chose à garder en mémoire lors de l'utilisation d'un microcontrôleur est qu'il ne possède que peu d'octets en mémoire vive. Cela signifie que vous ne devez pas déclarer de grandes structures de données ou des chaînes de caractère. Votre programme ne doit pas utiliser des fonctions avec beaucoup de niveaux d'imbrications ou de récursions.

Rien de mieux qu'un peu de pratique à la place des grandes théories. Nous allons écrire un programme qui fait clignoter notre LED avec un intervalle de 0.5 secondes. Pas très utile mais très bon pour débuter.

avr−libc a beaucoup changé. Avant, vous positionniez un bit sur un port avec sbi et vous l'effaciez avec cbi. Ces fonctions sont maintenant obsolètes. Je présente d'abord la "bonne vieille méthode":

```
/* defines for future compatibility */
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

void main(void)
{
      /* INITIALIZE */
      /* enable PC5 as output */
      sbi(DDRC,PC5);


      /* BLINK, BLINK ... */
      while (1) {
            /* led on, pin=0 */
            cbi(PORTC,PC5);
            delay_ms(500);
            /* set output to 5V, LED off */
            sbi(PORTC,PC5);
            delay_ms(500);
      }
}
```

L'exemple suivant fait exactement la même chose mais utilise la nouvelle syntaxe:

```
void main(void)
{
      /* INITIALIZE */
      /* enable PC5 as output */
      DDRC|= _BV(PC5);


      /* BLINK, BLINK ... */
```

```
        /* PC5 is 5 (see file include/avr/iom8.h) and _BV(PC5) is 00100000 */
        while (1) {
              /* led on, pin=0 */
              PORTC&= ~_BV(PC5);
              delay_ms(500);
              /* set output to 5V, LED off */
              PORTC|= _BV(PC5);
              delay_ms(500);
        }
}
```

Le petit bout de code ci−dessus montre comme il est facile d'écrire un programme. Vous voyez seulement le programme principal, la fonction delay_ms est incluse dans le listing complet (avrm8ledtest.c). Pour utiliser la broche PC5 comme sortie, vous avez besoin d'initialiser le bit PC5 dans le registre de direction de données pour le port C (DDRC). Après ceci, vous pouvez mettre PC5 à 0V avec la fonction cbi(PORTC,PC5) (clear bit PC5) ou à 5V avec sbi(PORTC,PC5) (set bit PC5). La valeur de "PC5" est définie dans iom8.h qui est incluse par io.h. Vous n'avez pas à vous en inquiéter. Si vous avez déjà écrit des programmes pour multi−utilisateurs / des systèmes multi−tâches tels que Linux, vous savez qu'il ne faut jamais programmer une boucle sans fin non bloquée. Cela serait une perte de temps CPU et cela ralentirait de beaucoup 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. Il n'y a même pas de système d'exploitation. Il est considéré comme normal d'occuper des boucles indéfiniment.

Compiler et charger

Avant de démarrer, assurez−vous que /usr/local/avr/bin est dans le PATH. Si nécessaire, éditez votre .bash_profile ou .tcshrc et ajoutez:

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

Nous utilisons le port parallèle et uisp pour programmer l'AVR. Uisp utilise l'interface ppdev du noyau. Vous devez donc avoir les modules de noyau suivants chargés:

```
# /sbin/lsmod
parport_pc
ppdev
parport
```

Contrôlez avec la commande /sbin/lsmod qu'ils sont chargés et si ce n'est pas le cas, chargez−les (comme root) avec:

```
modprobe parport
modprobe parport_pc
modprobe ppdev
```

Une bonne idée est d'exécuter ces commandes automatiquement au démarrage. Vous pouvez les ajouter comme un script rc (i.e. pour Redhat /etc/rc.d/rc.local).
Pour utiliser l'interface ppdev comme utilisateur normal, root doit vous donner l'accès en écriture en lançant la commande suivante une fois

chmod 666 /dev/parport0

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

Le paquet pour notre programme de test, (avrm8ledtest−0.1.tar.gz) inclut un make file. Tout ce que vous avez à faire est de saisir:

make
make load

Cela compilera et chargera le logiciel. Je ne vais pas entrer dans les détails de toutes les commandes. Vous pouvez les voir dans le Makefile et elles sont toujours identiques. Je ne peux pas toutes me les rappeler. J'ai juste besoin de savoir utiliser "make load". Si vous voulez écrire un programme différent, remplacez alors toutes les occurences de avrm8ledtest dans le Makefile avec le nom de votre programme.

Quelques binutils intéressants

Quelques binutils sont plus intéressant que le processus de compilation actuel.
Ces utilitaires n'ont néanmoins pas beaucoup changé depuis Mars 2002. Jetez un coup d'oeil au chapitre "Quelques "binutils" intéressants" sur l'article231 de Mars 2002.

Idées et suggestions

L'ATmega8 est compatible à l'AT90S4433 pour la plupart des utilisations. Vous devez programmer les bits fusible pour utiliser l'oscillateur externe et les circuits précédemment présentés devraient fonctionner avec quelques changements mineurs. Malheureusement, je n'ai pas encore le temps de re−tester tous les circuits pour l'ATmega8. Si vous voulez la sécurité, utilisez l'AT90S4433 pour les vieux articles. Si vous n'avez pas peur de trouver et régler des problèmes, tentez alors l'ATmega8 avec les vieux articles/circuits.

Vous avez ici une liste des précédents articles sur des circuits:

- Un panneau de contrôle LCD pour votre serveur Linux
- Alimentation pilotée par micro−contrôleur
- Un compteur de fréquence 1Hz−100Mhz avec afficheur LCD et interface RS232
- Affichage LCD USB sous Linux avec chien de garde et boutons
- Construire un robot autonome détecteur de lumière

Notez que le programmateur présenté ici inclus déjà les résistances de protection qui ont été placées sur la carte dans les précédents articles avec des circuits. Pour utiliser le nouveau programmateur avec les anciennes cartes, vous avez simplement à remplacer les résistances de protection sur la carte avec des fils.

Références

- L'AVRlib de Pascal Stang: http://www.procyonengineering.com/avr/avrlib/index.html ou http://hubbard.engr.scu.edu/embedded/avr/avrlib/
- l'assembleur tavrasm pour Linux: www.tavrasm.org
- **Tous les logiciels et documents mentionnés dans cet article**
- Le site web d'atmel: www.atmel.com

- Les pages d'électronique de la boutique tuxgraphics: shop.tuxgraphics.org
  (Vous pouvez y obtenir le CD de programmation AVR pour Linux, les kits et les microcontrôleurs)

Site Web maintenu par l´équipe d´édition LinuxFocus © Guido Socher "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Translation information: en −−> −− : Guido Socher (homepage) en −−> fr: Iznogood <iznogood/at/iznogood−factory.org>

2005−01−22, generated by lfparser_pdf version 2.51