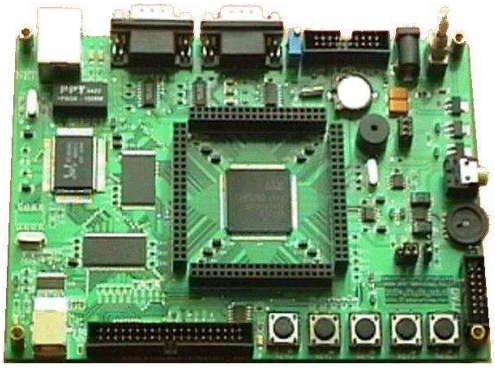
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

Pragmatec

Produits et services dédiés aux systèmes embarqués

uClinux - Driver

S3C44 uClinux driver developer guide



7 Janvier 2007 – Rev 1.00

Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

Les kits de développement ARM7 sont des kits réalisés par la société PRAGMATEC S.A.R.L., société située à Grenoble (www.pragmatec.net). Ils sont basés autour d'une carte de développement ARM et d'un processeur S3C44 ou S3C2410, largement utilisée en Asie depuis de nombreuses années. Il s'agit donc d'un produit efficace, fiable et disponible.

Pragmatec s'est attaché à faire de ces kits des environnements de développement complets et immédiatement opérationnels, avec une introduction en français et le reste des documents et exemples en langue anglaise. En cas de difficultés techniques vous bénéficiez de plus d'un support technique de la part de l'équipe support de Pragmatec : support@pragmatec.net.

Ce document a pour but de présenter les différentes étapes à suivre afin de réaliser un driver uClinux pour processeur S3C44. En effet, pour utiliser les périphériques hardwares du processeur, il importe de réaliser un driver, portion de code chargeable dynamiquement et qui permettra à n'importe quel processus de s'interfacer en toutes sécurités avec l'électronique de votre carte.

Ce document est la propriété de la société PRAGMATEC S.A.R.L. Il ne peut être reproduit et distribué sans l'accord de cette société.

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

TABLE DES MATIERES

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

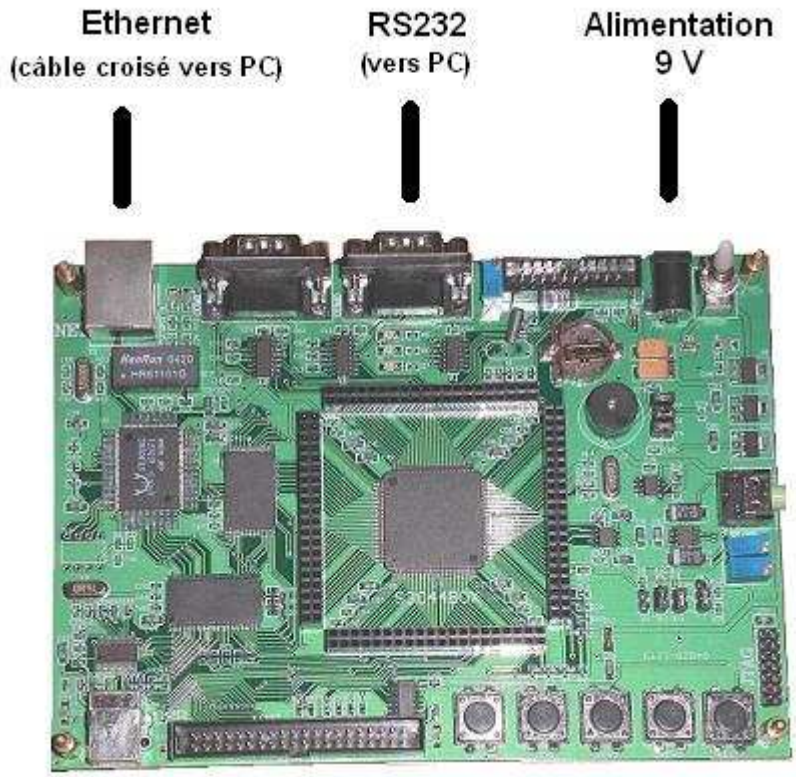
1 Préambule

Le but de ce guide de création d'un driver est de montrer qu'il n'est pas si difficile de développer son propre driver. Il est recommandé de consulter au préalable le document « uClinux_tutorial_S3C44B0.pdf » pour bénéficier d'une installation complète des outils de développement et du paramétrage de « minicom », ainsi que le document « uClinux_kernel_S3C44B0.pdf » si vous souhaitez modifier le noyau pré-installé.

La carte de développement

La carte ARM7 qui vous est proposée est basée sur le processeur S3C44BOX de Samsung. Elle possède de nombreux périphériques (USB, Ethernet, RS232, IDE, …) et ce document va vous démontrer comment écrire et exploiter un driver utilisant les ports de conversion analogique numérique 10 bits du CPU.

En plus de ce document et de votre carte de développement, vous aurez besoin d'une connexion RS232, d'un cordon Ethernet et de l'alimentation 9 à 12V.



Attention toutefois, les ports de conversions analogiques numériques du processeur S3C44 ne peuvent pas supporter une tension négative ou une tension positive supérieure à 2.5V. Protéger donc en conséquence votre carte de développement des surtensions.

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

Station Linux ou Windows™

Dans ce document nous recommandons l'utilisation d'une station **Linux**. Cela signifie que nous développerons sur un PC Linux et que nous compilerons le noyau et les programmes utilisateurs pour notre cible S3C44B0.

N'importe quel PC sous Linux fera l'affaire, mais nous avons choisi de réaliser ce tutorial depuis la FedoraCore, téléchargeable depuis le site de RedHat. Nous vous recommandons vivement d'utiliser cette distribution car l'environnement de développement Eclipse utilisé pour le debug y est déjà présent.

Vous pouvez aussi utiliser le logiciel « VMWARE » qui fonctionne depuis votre environnement Windows™. Ainsi vous pourrez y installer la FedoraCore comme indiqué dans le document « uClinux_kernel_S3C44B0.pdf » mais depuis Windows™.

Si vous êtes habitués à des développements sous Windows™ et l'utilisation d'interface graphique d'aide au développement (IDE), il est possible d'utiliser Eclipse sous Windows™ ainsi que la chaîne de compilation GCC au travers de cygwin. Reportez vous pour cela au document « uClinux_winux_s3c44B0.pdf ». Attention toutefois, vous ne pourrez réaliser que des applications par cette méthode, et en aucune façon des drivers.

Le noyau linux compilé pour votre cible est de la génération v2.4.
La chaîne de compilation GCC utilisée comme cross-compiler est la v2.95.3.

Enfin nous rappelons que le noyau destiné à la cible est un noyau uClinux. La différence majeure entre un noyau Linux et uClinux réside dans le fait que uClinux est une version dédiée aux processeurs qui ne bénéficient pas de la gestion de mémoire virtuelle (MMU) et d'unité de calcul à virgule flottante (FPU). Cette version est aussi appelée « NO_MMU / NO_FPU ». La librairie C nécessaire à la compilation des programmes utilisateurs a été aussi grandement allégée afin d'être plus efficace sur de petits systèmes embarqués (uClibc).

2 Création d'un driver minimaliste

Ce chapitre présente les étapes incontournables pour écrire et compiler un driver uClinux. Il s'agit ici de réaliser un driver pour utiliser le convertisseur analogique – numérique du processeur S3C44. Le driver sera chargé dynamiquement avec le noyau et nous verrons par la suite comment s'interfacer avec ce nouveau driver.

Création des fichiers

Le principe de fonctionnement est le suivant : nous allons créer un driver chargeable dynamiquement (module) qui pourra être utilisé depuis une application et le système de fichier de uClinux. Pour cela nous créerons un fichier « adc_s3c44.c » et son Makefile associé, ainsi qu'un programme de démonstration « accessADC.c » avec lui aussi un Makefile associé.

Le nom du module adc est « adc_s3c44.o ». Il s'agit d'un fichier objet (*.o) et pas d'un binaire exécutable car l'édition de lien est incomplète. Elle sera en fait terminée dynamiquement lors du chargement du module. C'est une des caractéristiques du noyau Linux que d'être capable de lier à la volée du code grâce à une sorte d'éditeur de lien dynamique contenu dans le noyau.

Le but est de coder un driver et une application qui permette d'afficher à l'écran la valeur numérique d'une quelconque entrée de l'ADC.

Fonctions primordiales

Un driver chargeable dynamiquement est appelé un module. Pour être chargé dynamiquement et donc lié dynamiquement au noyau il importe d'écrire un driver qui comporte au minimum 2 fonctions :

- init_module
- cleanup_module

A cela, ajoutez les directives d'inclusion (« #include »), quelques macros et définitions, et vous devriez obtenir le code suivant :

```
/*
 ************************************************************
 * Projet  : drivers/adc
 * Fichier : $RCSfile: adc_s3c44.c,v $
 * Auteur  : $Author: xmontagne $
 * Version : $Revision: 1.01 $
 * Date    : $Date: 2006-12-16 $
 * Tag     : $Name:  $
 ************************************************************
*/
```

```
#if defined(CONFIG_MODVERSIONS) && ! defined(MODVERSIONS)
#include <linux/modversions.h>
#define MODVERSIONS
#endif

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/irq.h>
#include <asm/uaccess.h>
#include <linux/signal.h>
#include <linux/sched.h>
#include <linux/interrupt.h>
#include <linux/ioport.h>
#include <linux/delay.h>
#include <asm/hardware.h>
#include <asm/io.h>
#include <linux/proc_fs.h>

/* Device name as it will appear in /proc/devices */
#define DEVICE_NAME          "ADC-S3C44B0X"
#define ADC_MAJOR            70

MODULE_LICENSE("GPL");

unsigned char VERSION = 1;
unsigned char ADCPSR  = 100;
unsigned int Sampling;
int channel[8];
int channelID = 0;

struct file_operations adc_fops =
{
        owner:          THIS_MODULE,
        read:           adc_read,
        write:          adc_write,
        open:           adc_open,
        ioctl:          null,
        release:        adc_close,
};

int init_module(void)
{
        unsigned int res, i_ms;
        printk("Loading ADC driver for S3C440X v1.%02d "__DATE__"\n", VERSION);

        writel  (readl(S3C44B0X_CLKCON)  |  (1<<12),  S3C44B0X_CLKCON);// Enable  ADC
clock
        //writel ( 0x01 | (0<<2), S3C44B0X_ADCCON ); // Enable ADC
        writel ( ADCPSR, S3C44B0X_ADCPSR );

        Sampling = (61000000/(2*(ADCPSR+1))/16);
        printk("ADCPSR = 0x%08x\n", readl(S3C44B0X_ADCPSR));
        ADCMSG("ADC conv freq = %dHz\n", (int)(Sampling));

        res = register_chrdev(ADC_MAJOR, DEVICE_NAME, &adc_fops);
        if (res < 0)
        {
                printk("Error registering driver.\n");
                return -ENODEV;
        }
```

```
        printk("ADC driver registered !\n");
        return 0;
}

void cleanup_module(void)
{
        writel ( 0x00 | (0<<2), S3C44B0X_ADCCON ); // Disable ADC
        unregister_chrdev(ADC_MAJOR, DEVICE_NAME);
        printk("Unloading ADC driver successfully\n");
}
```

Dans les fonctions « init_module » et « cleanup_module » vous remarquerez que l'essentiel du code tourne autour des appels à « register_dev » et « unregister_dev ». Ces 2 appels permettent d'enregistrer le driver auprès du noyau ou au contraire à le retirer de la base de données des drivers associés au noyau.

L 'appel à « register_dev » permet d'ajouter à la base de données des drivers en mode caractère notre nouveau driver avec les informations suivantes :

- Le **nom du driver** tel qu'il apparaît dans /proc/devices est DEVICE_NAME (« ADC_S3C44B0X »).
- Le **numéro de majeur** est ADC_MAJOR (« 70 »), qui correspond à l'interface /dev/adc. Veillez à ce que cette interface soit bien présente dans votre système de fichier.
- La structure qui définit les **interfaces du driver** est adc_fops. Elle précise les fonctions à appeler lors d'un « read », d'un « write », … depuis l'espace utilisateur, bref depuis votre programme.

Si l'on examine le code ci-dessus on remarque des appels à des fonctions « **printk** » qui ressemblent forts à des fonctions « printf ». Ceci est dû au fait que la fonction « printf » est une fonction de la librairie C (uClibc) et que le noyau n'utilise pas cette librairie ! Le noyau utilise donc une fonction spécifique qui est « printk » lorsque l'on a besoin d'afficher des informations à l'écran.

Si vous êtes connecté à votre cible en RS232 (via « minicom » ou « HyperTerminal »), vous verrez apparaître les messages issus des « printk » sur votre terminal.

Si vous êtes connecté à votre cible via TELNET, vous ne verrez pas directement apparaître les messages « printk ». Pour cela, utilisez la commande « dmesg » qui vous permettra d'afficher tous les messages « printk » depuis le démarrage du noyau. Il y a fort à parier que vos messages se trouvent à la fin…

Les dernières appels de fonctions inconnus sont les appels à « **writel** » et à « **readl** ». Ces fonctions permettent de lire ou d'écrire le contenu de n'importe quelle adresse mémoire et ceci sans se soucier de la notion de mémoire virtuelle… notre ARM7 et uClinux ne gérant pas la mémoire virtuelle !
Ainsi pour pouvoir utiliser les convertisseurs analogique-numériques du processeur, il est nécessaire de modifier les registres suivants :

- **S3C44B0X_CLKCON** : gestion de clock de l'ARM7. Permet de valider l'arrivée de l'horloge sur le convertisseur analogique-numérique.
- **S3C44B0X_ADCCON** : registre de contrôle du convertisseur. Nécessaire au paramétrage du hardware (démarrage, arrêt, ..).
- **S3C44B0X_ADCPSR** : prescaler du convertisseur. Avec ce registre vous pourrez définir la fréquence d'échantillonnage. Une variable globale « Sampling » est utilisée pour permettre de calculer la fréquence d'échantillonnage en Hertz en fonction de l'ADCPSR

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

Points d'entrée du driver

Vous aurez sans doute remarqué la présence d'une structure particulière appelée « adc_fops » :

```
struct file_operations adc_fops =
{
        owner:          THIS_MODULE,
        read:           adc_read,
        write:          adc_write,
        open:           adc_open,
        ioctl:          null,
        release:        adc_close,
};
```

Cette structure de type « struct file_operations » est en fait la définition des points d'entrées du driver. En effet les 2 fonctions « Cleanup_module » et « Init_module » servent au chargement dynamique du module, mais se peuvent être utilisé comme interface logicielle avec la partie applicative.

Sur uClinux comme sur tout système « Unix-like », un driver est accédé depuis l'application au travers d'un fichier. Dans notre cas, il s'agira du fichier « /dev/adc ». Comme pour tout fichier, vous devez au préalable y accéder via la fonction « open » avant d'y effectuer une opération « read » ou « write ».

Si vous appelez les fonctions « read » ou « write » sur un fichier texte du système de fichier, le noyau sait quoi faire, mais dans le cas de notre driver, le noyau ne peut deviner les actions à mener à l'issu de l'appel de ces fonctions. Il faut donc lui indiquer clairement les fonctions à appeler dans notre driver.

Il convient donc de compléter le driver pour y ajouter le code des fonctions correspondant à « read », « write », « open » et « close » :

```
int adc_open(struct inode *inode, struct file *file)
{
        ADCMSG("Device opened.\n");
        MOD_INC_USE_COUNT;
        writel ( 0x01 | (0<<2), S3C44B0X_ADCCON ); // Enable ADC
        return 0;
}

int adc_close(struct inode *inode, struct file *file)
{
        ADCMSG("Device closed.\n");
        MOD_DEC_USE_COUNT;
        writel ( 0x00 | (0<<2), S3C44B0X_ADCCON ); // Disable ADC
        return 0;
}
```

uClinux – Driver

Les fonctions « adc_open » et « adc_close » ne font qu'activer et désactiver le convertisseur ADC du S3C44B0X De plus elles utilisent une MACRO qui sert à incrémenter et décrémenter un compteur d'ouverture de fichier et connaître ainsi le nombre d'application qui est en court d'ouverture du fichier.

Le code de la fonction « adc_read » est le suivant :

```
ssize_t adc_read(struct file *file, char *buffer, size_t length, loff_t *offset)
{
        unsigned int ret, i;
        int value;
        ADCMSG("read called : length = %d\n", length);
        writel ( 0x00 | (channelID<<2), S3C44B0X_ADCCON );
        for (i = 0; i<500; i++);
        writel ( 0x01 | (channelID<<2), S3C44B0X_ADCCON );
        while(readl(S3C44B0X_ADCCON) & 0x01);  // To avoid the first flag error case
        while(!(readl(S3C44B0X_ADCCON) & 0x40));  // To avoid the second flag error case
        for (i = 0; i < ADCPSR; i++);
        value = readl(S3C44B0X_ADCDAT);
        ADCMSG("channel[%d] = 0x%08x\n", channelID, value);
        channel[channelID] = value;
        copy_to_user(buffer, &value, length);
        return length;
}
```

La fonction « adc_read » commence tout d'abord par positionner le canal « channelID » via lequel l'acquisition sera faite, et à désactiver puis réactiver l'ADC.

Ensuite, il convient d'attendre un certain laps de temps qui correspond au temps d'échantillonnage, fixé par la variable « ADCPSR ». Enfin, il ne reste plus qu'à lire la contenu du registre S3C44B0X_ADCDAT pour obtenir la valeur de conversion.

Pour de plus amples informations quant au convertisseur ADC du processeur et son utilisation, reportez-vous à la datasheet du S3C44B0X disponible sur le CDROM de votre kit de développement ou bien sur le site web de Pragmatec.

Le code de la fonction « adc_write » est le suivant :

```
ssize_t adc_write(struct file *file, const char *buffer, size_t length, loff_t *offset)
{
        unsigned int ret;
        ADCMSG("write called : length = %d\n", length);
        ret = 0;
        return ret;
}
```

La fonction « adc_wrtie » n'a pas d'intérêt dans notre cas, car il n'agit uniquement d'un driver ADC. La fonction retourne donc 0.

Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

Makefile

Le fichier Makefile est nécessaire à la compilation du driver. Le driver ADC que nous vous décrivons se trouve sous le répertoire « drivers/adc » de la distribution uClinux de Pragmatec. C'est donc à cet endroit que se trouve les 2 seuls fichiers dont nous avons besoin pour compiler le driver en tant que module : les fichiers « adc_s3c44.c » et « Makefile ».

Voici le contenu du fichier Makefile :

```
# Ana log to Digital Converter driver for S3C44B0X
# Makefile pour le projet drivers/adc

INCLUDEDIR_LINUX_ = ../../include

CC = arm-elf-gcc

CFLAGS = -Dlinux -DMODULE -D__KERNEL__ -D__linux__ -Dunix -D__uClinux__ -DEMBED -
I$(INCLUDEDIR_UCLIBC) -I$(INCLUDEDIR_DISTR_) -fno-builtin -nostartfiles -
I$(INCLUDEDIR_LINUX_) -I.
CFLAGS += -DADC_DEBUG

SRC_WR = adc_s3c44.c
OBJ_WR = adc_s3c44.o

all :
        $(CC) $(CFLAGS) -c $(SRC_WR)
        cp $(OBJ_WR) /tftpboot

clean :
        rm -f $(OBJ_WR)
```

Les points importants sont les suivants :

- **INCLUDEDIR_LINUX_** : indique l'emplacement des fichiers d'inclusions de uClinux
- **CFLAGS** : précise les options de compilations. Ce flag est surchargé par une ligne supplémentaire qui sert à activer les traces de débug (décrits dans le chapitre suivant). Pour le retirer, supprimer la ligne ou placer un « # » devant.
- **All** : point d'entré du Makefile. Une première ligne stipule la commande de compilation à effectuer sur le fichier « adc_s3c44.c » et une seconde ligne copie ce fichier dans le répertoire « /tftpboot » pour faciliter son transfert vers la cible.

Désormais pour compiler le driver, il suffira de lancer la commande « make » dans le répertoire du driver ADC. Mais un peu de patience, nous allons améliorer notre driver, car pour l'instant nous n'avons pas encore la possibilité de choisir le canal du convertisseur utilisé pour la conversion.

3 Amélioration du driver

Le présent chapitre présente quelques solutions pour améliorer notre driver, notamment grâce à la fonction « ioctl », ainsi qu'au système d'interface « /proc ».

Utilisation d'ioctl

Nous avons pu voir que pour accéder à notre driver nous utilisons une interface fichier « /dev/adc » et les interfaces logicielles offertes par la librairie C : open, close, read, write.

En fait il est aussi possible d'utiliser la fonction « ioctl », qui n'a guère d'utilité sur un fichier texte mais qui la plupart du temps se retrouve utilisée dans le cas de paramétrage des drivers. Elle sera utilisée par exemple dans le cas d'un paramétrage d'une communication série RS232 ou Ethernet.

Nous allons utiliser la fonction « ioctl » pour sélectionner le canal de la conversion analogique numérique. Auparavant la structure « adc_fops » affectait « null » au pointeur de fonction dédié à « ioctl ». Nous allons donc commencer par le modifier :

```
struct file_operations adc_fops =
{
        owner:          THIS_MODULE,
        read:           adc_read,
        write:          adc_write,
        open:           adc_open,
        ioctl:          adc_ioctl,
        release:        adc_close,
};
```

Ensuite nous allons ajouter la fonction « adc_ioctl » comme suit :

```
#define ADC_VERSION         0x400
#define ADC_SAMPLING        0x800
#define ADC_CHANNEL         0x500

int adc_ioctl (struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
    ADCMSG("timer3-%d ioctl, cmd: 0x%x, arg: %lx.\n", MINOR(inode->i_rdev),cmd, arg);

    switch ( cmd )
    {
    case ADC_VERSION:
        return VERSION;
        break;
    case ADC_SAMPLING:
        if (arg < 8000 || arg > 100000)
            return -EINVAL;
        ADCPSR = 61000000 / (arg*16);
        ADCPSR = (ADCPSR / 2) - 1;
        writel ( ADCPSR, S3C44B0X_ADCPSR );
        break;
```

```
            case ADC_CHANNEL:
                        if (arg < 1 || arg > 8)
                                    return -EINVAL;
                        channelID = arg - 1; // 0...7
            default :
                        break;
            }

      return 0;
}
```

La plupart du temps la fonction liée à « ioctl » est centrée sur le couple « switch/case » qui traite les paramètres passés à la fonction. Dans notre cas on trouvera :

- **ADC_VERSION :** retourne le numéro de version du driver
- **ADC_SAMPLING :** modifie la fréquence d'échantillonnage (entre 8000 Hz et 100000Hz)
- **ADC_VERSION :** précise le numéro de canal à échantillonner (numéro compris entre 1 et 8)

Ainsi, lorsque l'on souhaite effectuer une série d'acquisition sur le premier canal, il suffira d'utiliser la fonction « ioctl » au début pour sélectionner le canal, puis d'appeler autant de fois que nécessaire la fonction « read ».

Chaque fois que l'on souhaitera changer de canal il sera nécessaire en revanche d'appeler la fonction « iotctl ».

Interface /proc

Avec les fonctions d'interface que nous avons vu précédemment, nous allons donc pouvoir développer une application capable d'utiliser le driver et ainsi d'obtenir les valeurs de conversion sur les canaux choisis.

Toutefois il peut être intéressant de pouvoir « espionner » le fonctionnement d'un driver sans pour autant perturber son fonctionnement. Concernant notre driver, on pourrait ainsi connaître les dernières valeurs de conversion sur chacun des canaux. Mieux encore il serait souhaitable de pouvoir intervenir sur les paramètres comme le taux d'échantillonnage, sans pour autant devoir recompiler l'application ou le driver. Rien de tel pour la mise au point !

Et bien c'est ce que propose l'interface fichiers « /proc » de Linux. Ce répertoire n'est pas à proprement parler un répertoire physiquement présent dans le système de fichier. Il s'agit en fait d'une interface de communication avec le noyau et les drivers. Par exemple tapez dans une sessions shell la commande « cat /proc/meminfo ». Vous verrez que vous obtiendrez des informations sur la consommation mémoire de votre cible.

Pourtant c'est information ne sont pas stockées réellement dans un fichier mais sont fournies sous une forme de fichier à la demande par le noyau.

uClinux – Driver

Nous allons faire de même pour notre driver. Les interfaces que nous allons créer seront les suivantes :

- Les fichiers « **/proc/ADC/ChannelN** » sont destinés à être uniquement lus, car ils serviront à fournir la dernière valeur acquise sur ce canal.

- Le fichier « **/proc/ADC/Sampling** » est destiné à être lu ou écrit, car il permettra de connaître ou de positionner le taux d'échantillonnage.

Ces fichiers d'interface seront créés au chargement du noyau, mais devront disparaître lorsqu'on déchargera le driver. Les fonctions « Init_module » et « Cleanup_module » devront donc être modifiées comme suit :

```
int init_module(void)
{
   unsigned int res, i_ms;
   struct proc_dir_entry *proc_ADC;
   struct proc_dir_entry *proc_Channel1;
   struct proc_dir_entry *proc_Channel2;
   struct proc_dir_entry *proc_Channel3;
   struct proc_dir_entry *proc_Channel4;
   struct proc_dir_entry *proc_Channel5;
   struct proc_dir_entry *proc_Channel6;
   struct proc_dir_entry *proc_Channel7;
   struct proc_dir_entry *proc_Channel8;
   struct proc_dir_entry *proc_Sampling;

   printk("Loading ADC driver for S3C440X v1.%02d "__DATE__"\n", VERSION);

   writel ( readl(S3C44B0X_CLKCON) | (1<<12), S3C44B0X_CLKCON );// Enable ADC clock
   //writel ( 0x01 | (0<<2), S3C44B0X_ADCCON ); // Enable ADC
   writel ( ADCPSR, S3C44B0X_ADCPSR );

   Sampling = (61000000/(2*(ADCPSR+1))/16);
   printk("ADCPSR = 0x%08x\n", readl(S3C44B0X_ADCPSR));
   ADCMSG("ADC conv freq = %dHz\n", (int)(Sampling));

   res = register_chrdev(ADC_MAJOR, DEVICE_NAME, &adc_fops);
   if (res < 0)
   {
      printk("Error registering driver.\n");
      return -ENODEV;
   }

   proc_ADC = proc_mkdir("ADC", 0);
   proc_Channel1 = create_proc_read_entry("Channel1", 0, proc_ADC, adc_read_proc,
(void*)(&channel[0]));
   proc_Channel2 = create_proc_read_entry("Channel2", 0, proc_ADC, adc_read_proc,
(void*)(&channel[1]));
   proc_Channel3 = create_proc_read_entry("Channel3", 0, proc_ADC, adc_read_proc,
(void*)(&channel[2]));
   proc_Channel4 = create_proc_read_entry("Channel4", 0, proc_ADC, adc_read_proc,
(void*)(&channel[3]));
   proc_Channel5 = create_proc_read_entry("Channel5", 0, proc_ADC, adc_read_proc,
(void*)(&channel[4]));
   proc_Channel6 = create_proc_read_entry("Channel6", 0, proc_ADC, adc_read_proc,
(void*)(&channel[5]));
   proc_Channel7 = create_proc_read_entry("Channel7", 0, proc_ADC, adc_read_proc,
```

```
(void*)(&channel[6]));
   proc_Channel8 = create_proc_read_entry("Channel8", 0, proc_ADC, adc_read_proc,
(void*)(&channel[7]));
   proc_Sampling = create_proc_read_entry("Sampling", 0, proc_ADC, adc_read_proc,
(void*)(&Sampling));
   proc_Sampling->write_proc = adc_write_proc;

   printk("ADC driver registered !\n");
   return 0;
}


void cleanup_module(void)
{
  writel ( 0x00 | (0<<2), S3C44B0X_ADCCON ); // Disable ADC
  unregister_chrdev(ADC_MAJOR, DEVICE_NAME);
  remove_proc_entry ("ADC", NULL);
  printk("Unloading ADC driver successfully\n");
}
```

La fonction « create_proc_read_entry » permet d'associer chacune des variables concernées (Sampling, Channel0, …) à la fonction « adc_read_proc ». Cette dernière servira à convertir en texte la valeur de la variable.

De plus nous affectons le pointeur de fonction « write_proc » de « proc_Sampling » à l'adresse de la fonction « adc_write_proc ». Ainsi lorsque l'on souhaitera affecter une valeur de taux d'échantillonnage c'est cette fonction qui sera appelée. Une tentative d'écriture sur une quelconque autre variable n'aura aucun effet sur notre driver…

Dans la fonction « Cleanup_module », il suffira simplement de supprimer le répertoire « /proc/ADC » pour supprimer la totalité de l'interface.

La fonction « adc_read_proc » est assez générique. Vous pouvez la réutiliser telle quelle dans la plupart de vos drivers :

```
static int adc_read_proc(char *page, char **start, off_t off, int count, int *eof, void *data)
{
        char *out;
        int len;
        int *value;

        value = data;
        if (value == NULL)
                return -ENODEV;

        out = page;
        out += sprintf(out, "%d\n", *value);

        len = out - page - off;
        if (len < count)
            {
                        *eof = 1;
                if (len <= 0) return 0;
        } else {
```

```
            len = count;
        }
        *start = page + off;
        return len;
}
```

La fonction « adc_write_proc » est spécifique à la variable que vous souhaiterait modifier :

```
static int adc_write_proc(struct file *file, const char *buffer, unsigned long count, void *data)
{
        int value;
        int *ptr;

        if (count<0)
                return -EINVAL;
        if (count == 0)
                return 0;

        sscanf(buffer, "%d", &value);

        if ((data == &Sampling) && (value > 8000 && value < 100000))
        {
                ptr = data;
                *ptr = value;
              ADCPSR = 61000000 / (value*16);
              ADCPSR = (ADCPSR / 2) - 1;
              writel ( ADCPSR, S3C44B0X_ADCPSR );
              //Sampling = (61000000/(2*(ADCPSR+1))/16);
        }

        return count;
}
```

Traces de debug

Pour termnier ce chapitre nous vous présentons la MACRO de la fonction « ADCMSG » qui permet en fait d'afficher des traces du débug à l'aide de la fonction « **printk** » :

```
#ifdef ADC_DEBUG
        #define ADCMSG(fmt,args...) printk("ADC : " fmt,##args)
#else
        #define ADCMSG(fmt,args...)
#endif
```

Dans le cas d'une directive de compilation « ADC_DEBUG » déclaré dans le fichier de Makefile, la fonction « ADCMSG » se traduira par un « printk » commençant par « ADC….. ».

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

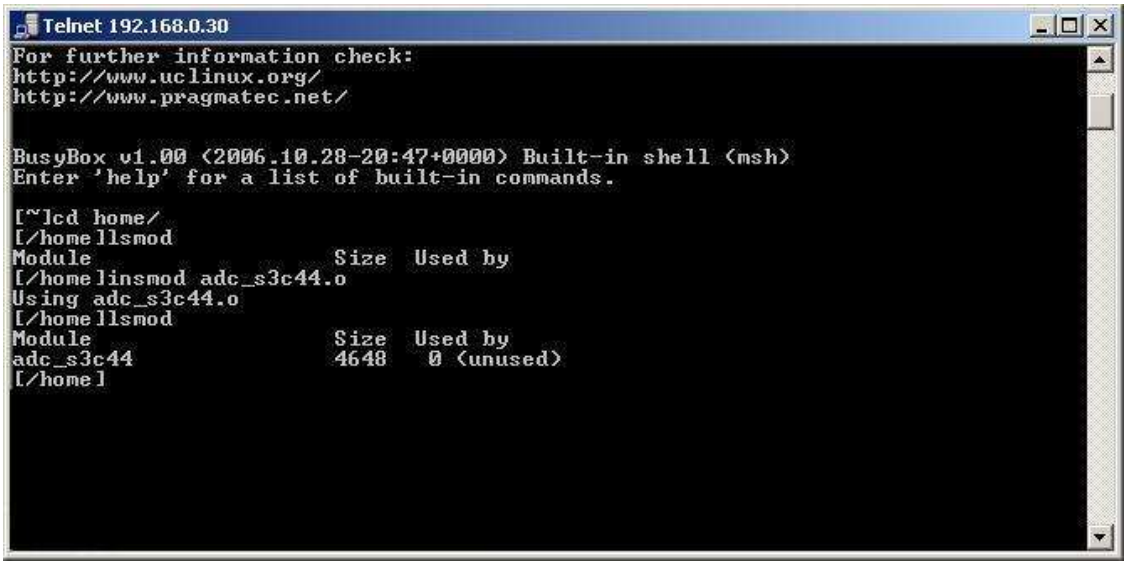
uClinux – Driver

Mise en oeuvre du driver

Pour pouvoir utiliser le driver il convient tout d'abord de le transférer sur la cible comme suit :

Nous pourrions aussi utiliser FTP ou NFS…

Utilisez les commandes « insmod » et « lsmod » pour charger le module en mémoire :



4 Utilisation du driver

Nous allons à présent coder une application qui utilise le driver ADC préalablement chargé. Cette application ne configurera pas le taux d'échantillonnage mais sélectionnera un canal afin d'en lire la valeur analogique convertie. Enfin, pour plus d'interactivité, le numéro de canal sera passé en paramètre de l'application.

Codage de l'application

Voici un exemple d'utilisation du driver adc_s3c44.o au travers de l'interface /dev/adc. Le fichier s'appelle « accessADC.c » :

```
/* ADC driver for S3C44B0X - uClinux 2.4 */

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>


#define ADC_VERSION          0x400
#define ADC_SAMPLING         0x800
#define ADC_CHANNEL          0x500

/* accessADC /dev/adc 0  */
int main(int argc, char**argv)
{
        short data;
        int fd, ret, n;
        char adc_device[20] ;

        if (argc < 3)
        {
                printf("Usage : accessADC [DEVICE]  [channel]\n");
                printf("Ex    : accessADC /dev/adc 1\n");
                return (-1);
        }

        strcpy(adc_device, argv[1]) ;
        if ((fd = open(adc_device,O_RDWR)) < 0 )
        {
                perror("open");
                printf("Error opening %s\n", adc_device);
                return (-1);
        }

        sscanf(argv[2], "%x", &n);
        printf("\n--- Test de lecture ADC channel %d ---\n\n", n);
        ret = ioctl(fd, ADC_CHANNEL, n);
        if(ret <0)
        {
                perror("ADC_CHANNEL ioctl cmd");
                close(fd);
                return (-1);
        }
```

```
        read(fd, &data, 2);
        printf("val[%d] = %d\n", n, (int)data);
        close(fd);
        return 0;
}
```

Nous avons mis en gras les appels aux fonctions d'interface du driver : open, close, read et ioctl.

L'exécution du code se déroule en 3 temps :

- ✓ Analyse des paramètres , ouverture de l'interface spécifiée (/dev/adc)
- ✓ Paramétrage du driver en précisant le numéro de canal
- ✓ Lecture du canal choisi et affichage de la valeur

Si à l'aide d'un potentiomètre vous parvenez à modifier la valeur analogique à l'entrée du canal choisi, vous verrez ainsi varier la valeur affichée par l'application.

Création du Makefile

Le fichier « Makefile » est un script qui permet de lancer la compilation du ou des fichiers ainsi que l'édition de lien des fichiers compilés.

Le fichier « Makefile » donnée en exemple est situé dans le répertoire linux-2.4/drivers/adc/util de la distribution uClinux Pragmatec :

```
INCLUDEDIR_DISTR_ = ../../../..
UCLIBC_DIR = $(INCLUDEDIR_DISTR_)/lib/uClibc
INCLUDEDIR_LINUX_ = $(INCLUDEDIR_DISTR_)/linux-2.4/include
INCLUDEDIR_UCLIBC = $(UCLIBC_DIR)/include
RUNTIME = $(UCLIBC_DIR)/lib/crt0.o $(UCLIBC_DIR)/lib/crti.o $(UCLIBC_DIR)/lib/crtn.o

CC = arm-elf-gcc

CFLAGS = -g -Dlinux -D__linux__ -Dunix -D__uClinux__ -DEMBED -I$(INCLUDEDIR_UCLIBC) -
I$(INCLUDEDIR_DISTR_) -fno-builtin -nostartfiles -I$(INCLUDEDIR_LINUX_) -I.

LDFLAGS = -L$(UCLIBC_DIR)/. -L$(UCLIBC_DIR)/lib

SRC_WR = accessADC.c
OBJ_WR = accessADC.o

all :
        $(CC) $(CFLAGS) -c -o $(OBJ_WR) $(SRC_WR)
        $(CC) $(CFLAGS) -WI,-elf2flt $(RUNTIME) $(LDFLAGS) -o accessADC $(OBJ_WR) -lc
        cp accessADC /tftpboot

clean :
        rm -f *.o *.gdb accessADC
```

Remarque : pour les opérations « all » et « clean », les lignes de commande associées doivent commencer par une tabulation et non une série d'espace. De plus la ligne « CFLAGS » ne doit pas comporter de retour à la ligne et doit être écrite sur une seule ligne.

Pour compiler votre application, tapez simplement « make » à l'endroit où se trouvent les fichiers accessADC.c et Makefile.

A l'issu de la compilation vous devriez obtenir un fichier « accessADC.o », un binaire « accessADC » qui est votre programme et un fichier « accessADC.gdb » qui sert au debug.

Transférez le programme « accessADC » sur la cible (via le protocole FTP, TFTP ou NFS).

Chargement du programme

Pour lancer le programme il faut tout d'abord qu'il soit exécutable. Pour cela tapez « chmod +x accessADC » sur la cible. Ensuite lancez le programme comme suit :

accessADC /dev/adc 1

Vous devriez obtenir le résultat suivant en faisant varier une tension analogique sur le canal choisi :

La valeur qui sera affichée sera comprise entre 0 et 1023, car le convertisseur analogique numérique du S3C44 est un convertisseur 10bits, soit 1024 possibilités.

5 Améliorations supplémentaires

Nous allons à présent aborder les notions de gestion d'interruption, de handler d'interruption, et d'utilisation d'un timer hardware du S3C44. L'idée ici est d'utiliser le timer3 du processeur afin qu'il exécute une conversion analogique numérique périodiquement, toutes les 10 ms par défaut. Ceci vous permettra de connaître la valeur de la tension d'entrée d'un canal au travers de l'interface /proc, et donc sans même avoir besoin de coder une application.

Utilisation du TIMER3

Nous choisissons de paramétrer le timer3 directement dans la fonction « Init_module ». Il est nécessaire pour cela de positionner un prescaler et un diviseur afin de créer une période de base, ici, elle est de 100µs (d'où les 10000KHz indiqué dans les commentaires).

La variable « i_ms » stipule le nombre de millisecondes durant lequel le timer3 s'écoulera. Ensuite il suffit de paramétrer le timer3 en mode *repeat* comme suit :

```
i_ms = TIMER3_VAL; // 100 ms by default

printk("Loading ADC driver for S3C440X v1.%02d "__DATE__"\n", VERSION);

writel ( readl(S3C44B0X_CLKCON) | (1<<12), S3C44B0X_CLKCON );// Enable ADC clock
//writel ( 0x01 | (0<<2), S3C44B0X_ADCCON ); // Enable ADC
writel ( ADCPSR, S3C44B0X_ADCPSR );

Sampling = (61000000/(2*(ADCPSR+1))/16);
printk("ADCPSR = 0x%08x\n", readl(S3C44B0X_ADCPSR));
ADCMSG("ADC conv freq = %dHz\n", (int)(Sampling));

// clear manual update bit, stop Timer3 (f d<82>cal<82> de 16: timer 3, 12: timer 2
writel ( readl ( S3C44B0X_TCON  ) & ( ~ ( 0xf << 16 ) ), S3C44B0X_TCON );
writel  ( readl  ( S3C44B0X_TCFG0  )  &  0xffff00ff,  S3C44B0X_TCFG0  );  //  (presc=190)  =>
320000 KHz
writel ( readl ( S3C44B0X_TCFG0 ) | 0x0000BE00, S3C44B0X_TCFG0 );
writel ( readl ( S3C44B0X_TCFG1 ) & 0xffff0fff, S3C44B0X_TCFG1 ); // set Timer 3 MUX 1/32
writel ( readl ( S3C44B0X_TCFG1 ) | 0x00004000, S3C44B0X_TCFG1 ); // => 10000 KHz
writel ( i_ms * 10, S3C44B0X_TCNTB3 );              // en ms.
writel (        1, S3C44B0X_TCMPB3 );
writel ( readl ( S3C44B0X_TCON ) & ( ~ ( 0x0f << 16 ) ), S3C44B0X_TCON ); // REPEAT + OFF
writel ( readl ( S3C44B0X_TCON ) |     ( 0x0a << 16 )  , S3C44B0X_TCON );

...

writel ( readl ( S3C44B0X_TCON ) & ( ~ ( 0x0f << 16 ) ), S3C44B0X_TCON );  // TIMER3 Start
writel ( readl ( S3C44B0X_TCON ) |     ( 0x09 << 16 )  , S3C44B0X_TCON );
```

N'hésitez pas à consulter la datasheet du processeur S3C44 pour comprendre le fonctionnement et le paramétrage des timers.

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

Handler d'interruption

Comme nous souhaitons effectuer les conversions analogiques numériques sous interruption, il faut en déclarer une et l'associer au timer3.

La fonction « Init_module » sera complétée comme suit :

```
	/* Attaching IRQ handler */
	res = request_irq(S3C44B0X_INTERRUPT_TIMER3, adc_handler, SA_INTERRUPT,
"TIMER3_ADC", 0);
	if (res)
	{
		printk("Unable to use IRQ%u\n", S3C44B0X_INTERRUPT_TIMER3);
		return(1);
	}
```

Pour cela nous utilisons la fonction « request_irq » qui associera la fonction « adc_handler » à l'interruption du timer3. L'activation du timer3 est effectuée par cette fonction, il n'est pas nécessaire de le faire par vous même.

Par la suite nous créerons la fonction du handler d'interruption :

```
void adc_handler(int irq, void *dev_id, struct pt_regs *regs)
{
	int i, value;
	writel ( 0x00 | (channelID<<2), S3C44B0X_ADCCON );
	for (i = 0; i<500; i++);
	writel ( 0x01 | (channelID<<2), S3C44B0X_ADCCON );
	while(readl(S3C44B0X_ADCCON) & 0x01);  // To avoid the first flag error case
	while(!(readl(S3C44B0X_ADCCON) & 0x40));  // To avoid the second flag error case
	for (i = 0; i < ADCPSR; i++);
	value = readl(S3C44B0X_ADCDAT);
	channel[channelID] = value;
}
```

Comme vous pouvez le constater, nous avons repris intégralement le code de la fonction « adc_read » qui permet d'effectuer la conversion analogique numérique et de mettre à jour la variable « channel[n] ».

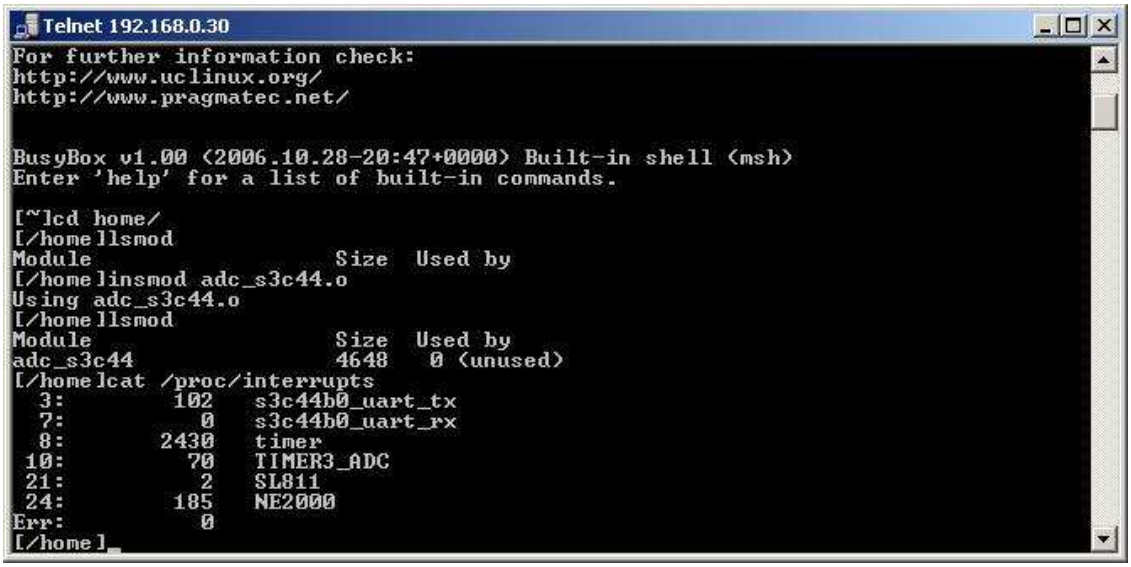
Remarque : *il peut être intéressant de pouvoir modifier la période du timer3 et pourquoi pas de sélectionner le canal en utilisant l'interface « /proc ». Modifiez donc le code en conséquence afin d'établir 2 entrées supplémentaires sous /proc/ADC.*

Pragmatec
Bâtiment EARHART
ZAC Grenoble Air Parc
38590 St Etienne de St Geoirs - France
www.pragmatec.net

uClinux – Driver

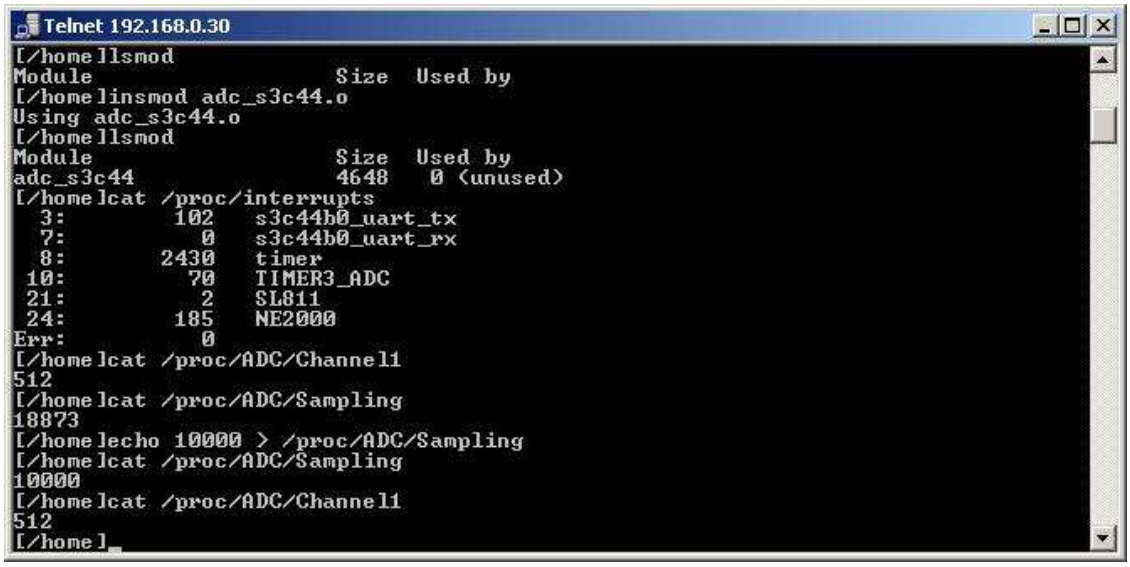
Mise en application

Utilisez les commandes « insmod » et « lsmod » pour charger le nouveau module en mémoire après l'avoir compilé :



Tapez la commande « cat /proc/interrupts ». Dans l'exemple ci-dessus on remarque que le timer3 a déjà subit 70 interruptions. Sa période étant de 100ms, il provoque l'appel du handler d'interruption 10 fois par seconde, et donc autant de fois une conversion analogique numérique.

Utilisez les commandes « cat » et « echo » via l'interface /proc/ADC, pour manipuler le driver et connaître les valeurs de conversion :



Toutes les sources sont présentes dans la distribution uClinux Pragmatec sous le répertoire « linux-2.4.x/drivers/adc ».