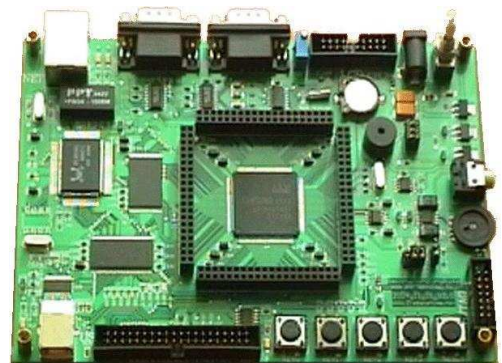


Pragmatec

Produits et services dédiés aux systèmes embarqués

uClinux User Guide

ARM7 Development Starter Kit





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

uClinux – User Guide



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

uClinux – User Guide

Le kit de développement ARM7 est un kit réalisé par la société PRAGMATEC S.A.R.L., société située à Grenoble (www.pragmatec.net). Il est basé sur une carte de développement ARM7, 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 ce kit un environnement de développement complet et immédiatement opérationnel, 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 démontrer la simplicité d'utilisation d'une telle plate-forme avec le système d'exploitation Linux, et tout particulièrement sa version embarquée : uClinux. Nous aborderons ici un exemple concret d'application au travers d'une carte d'extension sur bus I2C.

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.

TABLE DES MATIERES

1	<i>Préambule</i>	5
	La carte de développement.....	5
	Station Linux ou Windows™.....	6
2	<i>Pilotage d'une carte d'extension i2C</i>	7
	Connexion à une FS44B0X-I.....	7
	Détermination des broches d'extension.....	8
	Le bus I2C.....	9
	Le périphérique MCP23016	10
	Utilisation sous uClinux.....	11
3	<i>Driver i2c pour S3C44B0X</i>	12
	Génération du driver	12
	Interface /dev/i2c0	13
	Chargement du driver	14
4	<i>Création d'une application</i>	16
	Codage de l'application	16
	Création du Makefile	18
	Chargement du programme.....	18

uClinux – User Guide



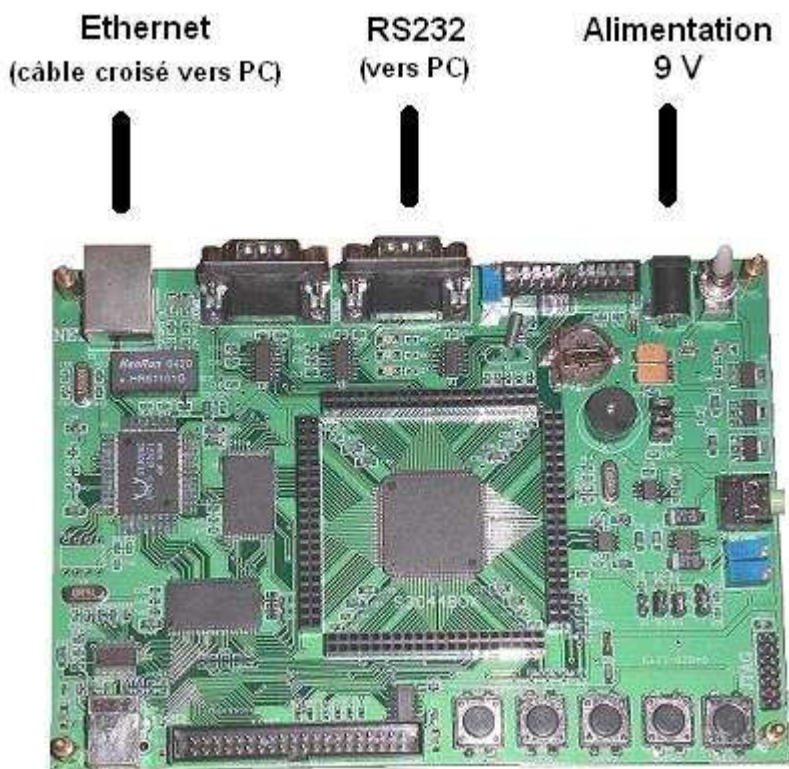
1 Préambule

Le but de ce guide d'utilisation est de montrer simplement comment utiliser une carte de développement Pragmatec (par exemple la FS44B0X-I) pour y associer des modules d'extension. 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.doc » 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 utiliser un de ces périphériques (le bus I2C) afin de piloter une carte d'extension quelconque.

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 ou 12V.



uClinux – User Guide

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 FedoraCore4, 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.

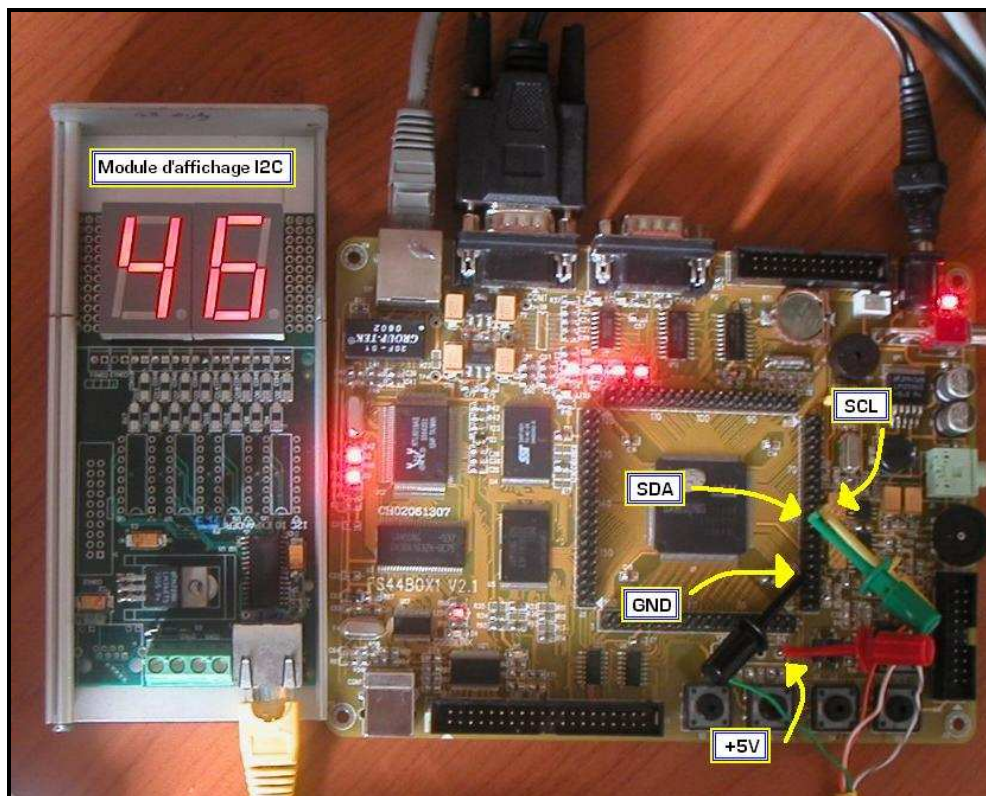


2 Pilotage d'une carte d'extension i2C

Ce chapitre présente la façon dont va être connectée la carte d'extension I2C à la carte FS44B0X-I. Cette carte d'extension est équipée d'un IOExpender de Microchip, le MCP23016. Il sera présenté dans ce même chapitre afin de vous permettre de réaliser votre propre carte très facilement.

Connexion à une FS44B0X-I

Notre carte d'extension I2C comporte un IOExpender MCP23016 qui sert essentiellement à piloter 2 afficheurs 7 segments avec point. La carte se connecte à la FS44B0X-I à l'aide de grip-fils directement connectés aux connecteurs 40 broches mâles de la carte CPU :



La carte d'extension est alimentée par la carte CPU, nous avons donc besoin de 4 fils :

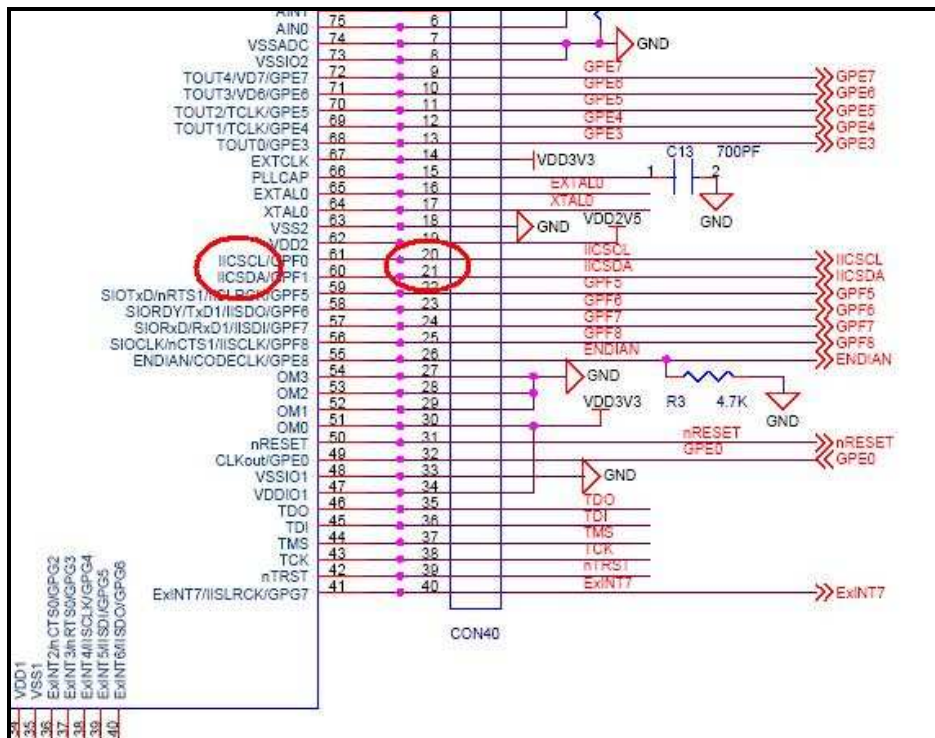
- **rouge** : alimentation de la carte en 5V
- **noir** : masse
- **vert** : ligne I2C SDA (pull-up sur la carte CPU)
- **jaune** : ligne I2C SCL (pull-up sur la carte CPU)

Le but est de coder une application qui permette d'afficher une valeur quelconque sur les 2 afficheurs 7 segments. L'application utilisera le driver I2C de Pragmatec pour uClinux et la carte FS44B0X-I.

Détermination des broches d'extension

Pour connecter votre carte d'extension I2C à la carte FS44B0X-I vous avez donc besoin de connaître l'emplacement exacte des signaux « 5V », « GND », « SDA » et « SCL ». Pour cela il vous faut utiliser le document appelé « FS44B0X_I.pdf » et présent sur le CDROM livré avec les kits.

Ce document est la version PDF du schéma électrique de la carte réalisé sous PROTEL DXP. Sur la première page vous trouverez le numéro des broches qui sont affectées aux signaux SCL et SDA :



Sur votre carte vous devez alors repérer la broche 1 du connecteur de droite. Pour cela retournez la carte et identifier la broche dont la forme de la pastille est un carré et non pas un rond : il s'agit de broche 1. Retournez la carte à nouveau afin de visualiser les composants. Pour trouver les broches 20 et 21 il vous suffit alors de compter... mais attention, toujours en quinconce ! La broche 2 se trouve en face de la broche 1 et la broche 3 est en dessous de la broche 1, et ainsi de suite...

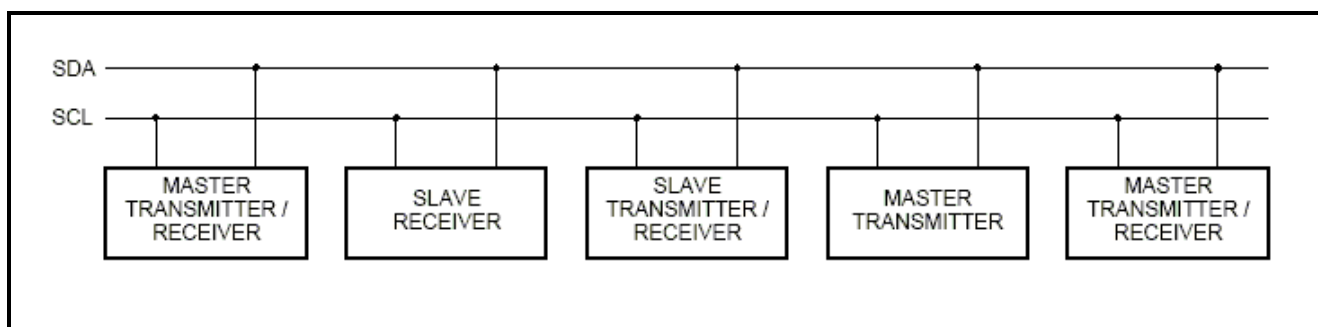
Pour ce qui est des alimentations, vous trouverez sur la carte FS44B0X-I un connecteur d'alimentation IDE avec une point à 12V, un autre à 5V et les 2 centraux à GND. Il se trouve juste en dessous du processeur et des connecteurs d'extension.

uClinux – User Guide

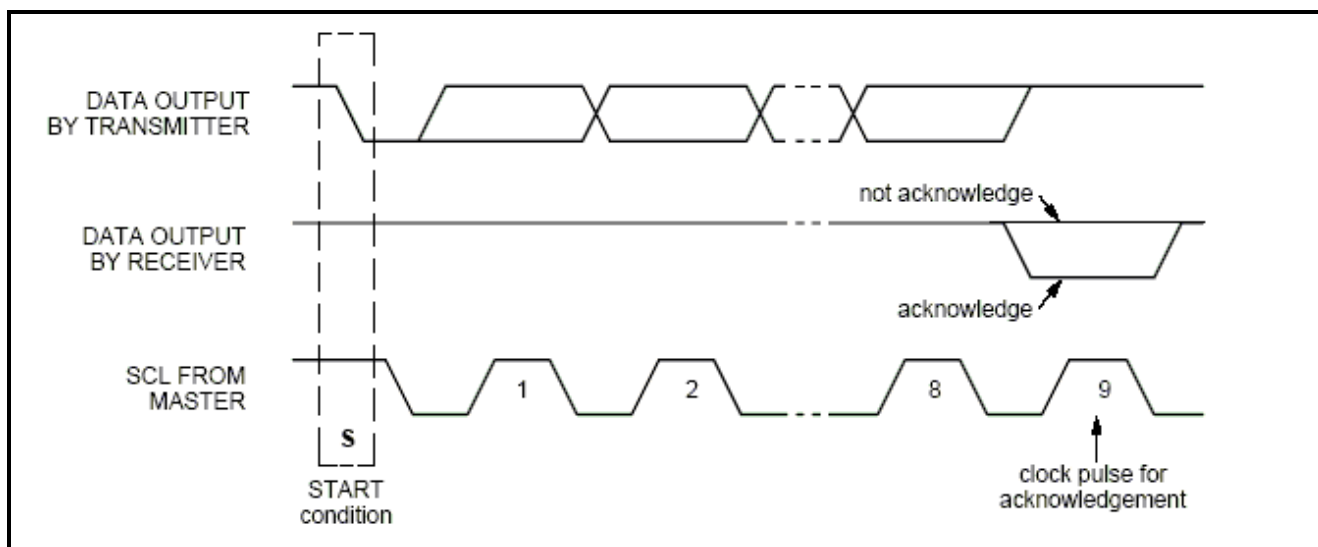
Le bus I2C

Le bus i2c est un bus série synchrone, c'est-à-dire que la transmission des données de fait en série sur un seul fil (bit après bit) et qu'un autre signal sert à transmettre l'horloge. Ainsi sur chaque front d'horloge le bit de donnée est échantillonné afin de constituer un ou plusieurs octets.

Ce bus est dit multi-maître car il permet de connecter plusieurs systèmes i2c sur un même bus et que chaque système a la possibilité d'émettre (mode MASTER) et de recevoir des données (mode SLAVE) :



Les données sont donc transmises en série d'un maître (émetteur) vers un esclave (le récepteur). Chaque récepteur se doit d'acquitter la donnée reçue en maintenant la ligne SDA à 0 lors de la fin de la transmission de l'octet de donnée :



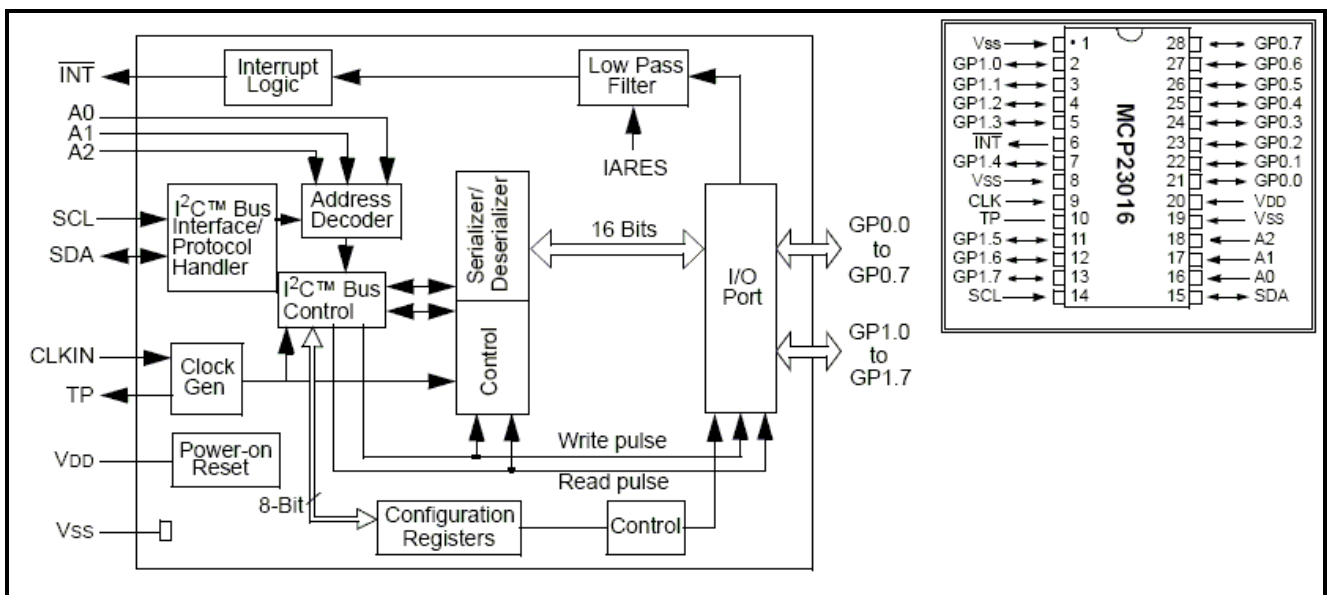
Comme plusieurs esclaves peuvent être connectés sur le bus i2c il convient de transmettre l'adresse du destinataire avant chaque octet transmis. Ainsi un seul et unique esclave sera concerné par la ou les données transmises.

uClinux – User Guide

Le périphérique MCP23016

Le composant MCP23016 de la société Microchip fait parti des composants dits « IOExpander », c'est-à-dire « extenseur de ports » en français. Cela signifie que si vous souhaitez bénéficier de quelques IO pour piloter des relais ou encore lire des boutons poussoirs, vous pouvez le faire à l'aide d'un IOExpander qui vous offrira cette fonctionnalité au travers du bus i2c.

Le MCP23016 est un IOExpander de 16 bits, il vous permet donc d'obtenir 16 bits de données utilisables indépendamment en entrée ou en sortie.



Comme la plupart des composants sur bus i2c, le MCP23016 dispose de quelques broches qui vous permettent de spécifier l'adresse i2c du périphérique. Une adresse i2c est toujours sur 7 bits et non pas 8 bits, le bit de poids faible étant utilisé pour spécifier s'il s'agit d'une action de lecture ou d'écriture. La datasheet du MCP23016 précise que l'adresse du composant se compose comme suit :

1.6 Address Decoder

The last three LSb of the 7-bit address are user-defined (see Table 1-4). Three hardware pins (<A2:A0>) define these bits.

TABLE 1-4: DEVICE ADDRESS

0	1	0	0	A2	A1	A0
---	---	---	---	----	----	----

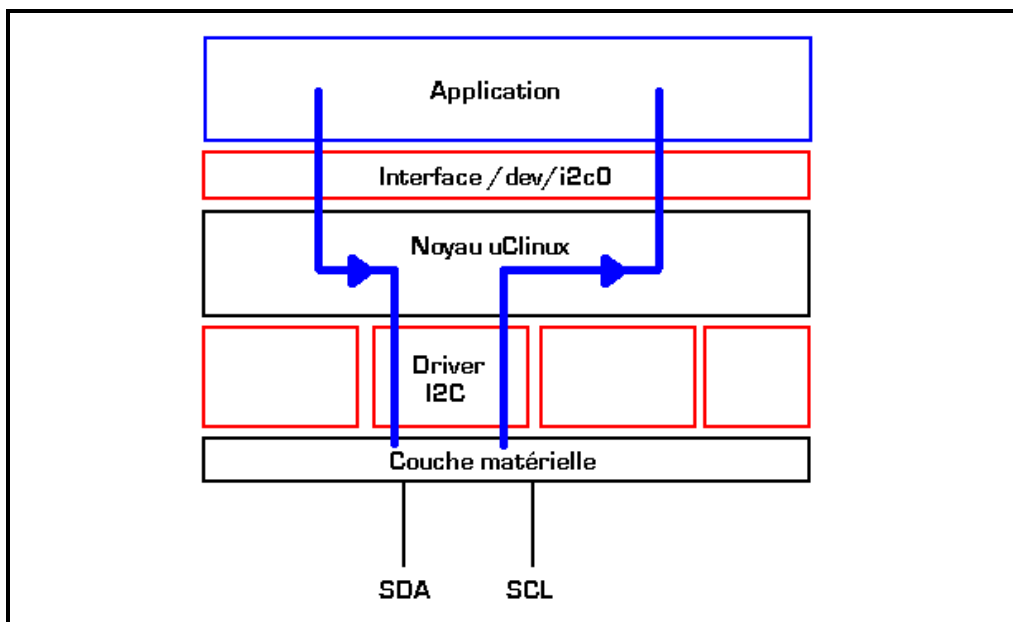
Si A2 = GND, A1 = GND et A0 = 5V, l'adresse de la cible sera 0x21, le décalage à gauche pour gérer le bit de lecture ou d'écriture étant automatiquement géré par le driver et le matériel du S3C44.

uClinux – User Guide

Utilisation sous uClinux

Avec uClinux vous pourrez développer des applications vous permettant d'utilisation des systèmes d'extension connectée au bus i2c. Toutefois il est de responsabilité du noyau de gérer le matériel, ce ne sera donc pas à vous de coder le protocole i2c à proprement parler.

Le noyau qui est pré-chargé sur la FS44B0X-I ne permet pas de gérer le bus i2c du processeur S3C44B0X. Pour cela il faut lui adjoindre un driver que nous allons charger depuis un shell afin de l'associer dynamiquement au noyau : on parle alors de « module ».



Il faut donc :

- Recompiler le driver i2c pour S3C44B0X et uClinux
- Coder une application afin d'utiliser le driver i2c
- Charger le driver i2c sur la cible
- Exécuter l'application

Pour s'interfacer avec le driver, l'application devra dialoguer avec un fichier d'interface présent sur la distribution uClinux pré-chargé sur la carte : /dev/i2c0. Pour envoyer des données au module i2c, il suffit d'écrire dans le fichier /dev/i2c0, et pour lire des octets en provenance du module i2c, il suffit de lire le fichier /dev/i2c0.



3 Driver i2c pour S3C44B0X

Le présent chapitre présente le driver `i2c_s3c44.o`, la façon de le générer et de le charger sur la cible.

Génération du driver

Le driver i2c est un module chargeable dynamiquement sur la cible. Il crée en quelque sorte une extension dynamique du noyau en lui fournissant de nouvelles capacités le temps que le module reste chargé.

Le nom du module i2c est « `i2c_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 lien à la volée du code grâce à une sorte d'éditeur de lien dynamique contenu dans le noyau.

Pour générer le driver i2c, allez dans le répertoire `linux-2.4/drivers/i2c` de la distribution uClinux de votre carte et compilez le driver à l'aide de la commande « `make` » :

```
xavier@localhost:~/uClinux-pragm-new/linux-2.4.x/drivers/i2c/S3C44 - Shell No. 5 - Konsole
Session Edit View Bookmarks Settings Help
[xavier@localhost S3C44]$
[xavier@localhost S3C44]$
[xavier@localhost S3C44]$
[xavier@localhost S3C44]$
[xavier@localhost S3C44]$ ll
total 36
-rwxrwxr-x 1 xavier xavier 12368 Aug 21 21:30 i2c_s3c44.c
-rwxrwxr-x 1 xavier xavier 490 Aug 22 01:14 Makefile
drwxrwxr-x 4 xavier xavier 4096 Aug 21 19:32 utils
[xavier@localhost S3C44]$
[xavier@localhost S3C44]$ make clean
rm -f i2c_s3c44.o
[xavier@localhost S3C44]$
[xavier@localhost S3C44]$ make
arm-elf-gcc -Dlinux -DMODULE -D_KERNEL__ -D_linux__ -Dunix -D_uClinux__ -DEMBED -I -I -fno-builtin -nostartfi
les -I../..../include -I. -c i2c_s3c44.c
cp i2c_s3c44.o /tftpboot
[xavier@localhost S3C44]$ ll
total 48
-rwxrwxr-x 1 xavier xavier 12368 Aug 21 21:30 i2c_s3c44.c
-rw-rw-r-- 1 xavier xavier 7564 Sep 4 10:58 i2c_s3c44.o
-rwxrwxr-x 1 xavier xavier 490 Aug 22 01:14 Makefile
drwxrwxr-x 4 xavier xavier 4096 Aug 21 19:32 utils
[xavier@localhost S3C44]$
```

Le fichier ainsi généré s'appellera « `i2c_s3c44.o` ».

uClinux – User Guide

Interface /dev/i2c0

Si vous souhaitez communiquer avec des périphériques i2c au travers du driver i2c_s3c44.o, il vous faut accéder au fichier /dev/i2c0.

Les différentes actions possibles sont les suivantes :

- **Open** : il s'agit de la première opération à effectuer sur le fichier. En effet avant de pouvoir effectuer la moindre action sur le driver il convient de demander l'autorisation au noyau. Par exemple si le driver i2c n'est pas préalablement chargé, le noyau vous refusera l'accès au fichier /dev/i2c0
- **Close** : c'est le pendant de l'opération « open ». Il permet de signaler au noyau et au driver que l'application ne souhaite plus utiliser le périphérique i2c. Si vous oubliez d'utiliser l'opération « close » à la fin de votre application, le noyau uClinux le fera automatiquement pour vous.
- **ioctl** : comme nous l'avons vu précédemment, le bus i2c permet d'avoir plusieurs périphériques connectés au bus, il convient donc de préciser l'adresse de l'esclave sur le bus i2c. La commande « ioctl » est une sorte de fourre-tout du driver qui nous sert ici à préciser cette adresse de destination.
- **Write** : afin de transmettre des données au périphérique i2c, il faut les envoyer au driver au travers de l'opération « write ».
- **Read** : afin de lire des données en provenance d'un périphérique i2c, il faut le demander au driver au travers de l'opération « read ».

Si vous souhaitez savoir comment utiliser ces opérations et comment les coder en langage C, nous vous invitons à consulter un ouvrage qui traite des accès aux fichiers en C, ou tout simplement en tapant dans un shell la commande « man open » pour la commande « open » et ainsi de suite.



```
Telnet 192.168.0.30
PRAGMATEC0 login: root
Password:
Welcome to
          uClinux
For further information check:
http://www.uclinux.org/
http://www.pragmatec.net/
BusyBox v1.00 (2006.06.27-14:35+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.
[~]ls -l /dev/i2c*
crw-rw-rw- 1 root  root  89,  0 Jan  1 1970 /dev/i2c0
[~]
```

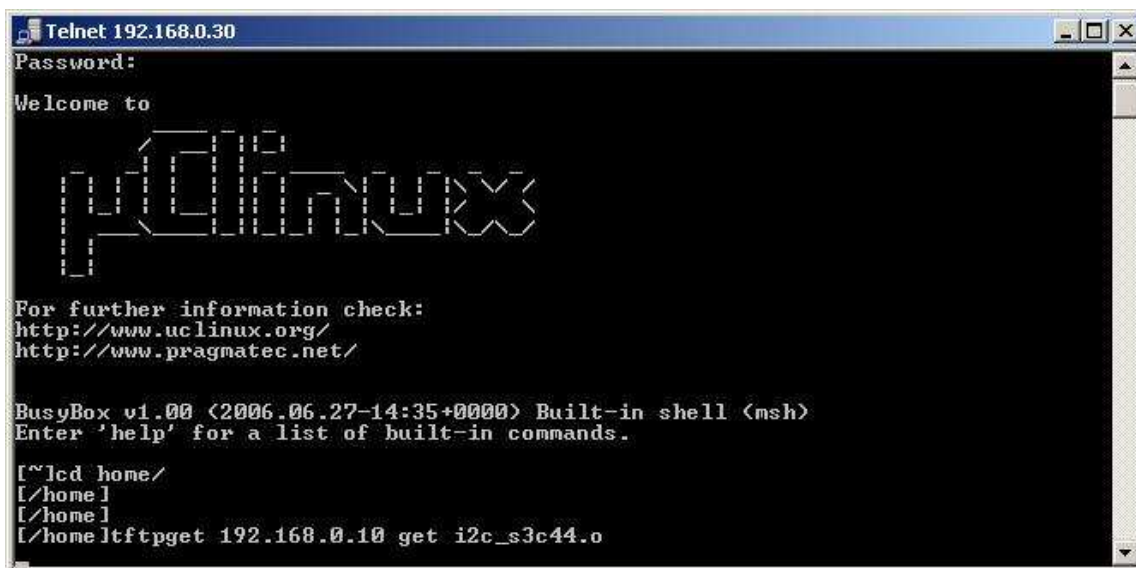
Le chapitre 4 présente un exemple de mise en œuvre incluant la syntaxe des différentes commandes.

uClinux – User Guide

Chargement du driver

Une fois généré, le driver devra être transféré sur la cible puis chargé en mémoire. Pour éviter de refaire cette opération après chaque démarrage de la carte nous allons le sauvegarder en mémoire NAND et modifier le script de démarrage.


La compilation du driver a permis de copier le fichier sous le répertoire /tftpboot de la station. Nous allons pouvoir transférer le fichier à l'aide du programme « **tftpget** ». Sous Windows™, utilisez le programme TFTP32.exe en tant que serveur TFTP.



```
Telnet 192.168.0.30
Password:
Welcome to
uCLinux
For further information check:
http://www.uclinux.org/
http://www.pragmatec.net/
BusyBox v1.00 (2006.06.27-14:35+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.
[~/]lcd home/
[~/home]
[~/home]
[~/home]tftpget 192.168.0.10 get i2c_s3c44.o
```

Nous pourrions aussi utiliser FTP...

Vérifions le nombre de module déjà chargé sur la cible à l'aide de la commande « **lsmod** » :



```
Telnet 192.168.0.30
PRAGMATEC0 login: root
Password:
Welcome to
uCLinux
For further information check:
http://www.uclinux.org/
http://www.pragmatec.net/
BusyBox v1.00 (2006.06.27-14:35+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.
[~/]lcd home
[~/home]lsmod
Module          Size Used by
[~/home]
```




4 Création d'une application

Nous allons à présent coder une application qui utilise le driver i2c préalablement chargé. Cette application configurera les 16 bits du MCP23016 en sortie afin d'afficher sur les 2 afficheurs 7 segments des valeurs numériques.

Codage de l'application

Voici un exemple d'utilisation du driver i2c_s3c.o au travers de l'interface /dev/i2c0 :

```
/* Exemple I2C sur S3C44B0X */

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

#include <linux/i2c.h>

/* Valeur en hexa à appliquer aux 7 segments pour afficher un chiffre */
unsigned char tab[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

/* accessMCP /dev/i2c0 42 */
int main(int argc, char**argv)
{
    unsigned char data[10];
    int fd, ret, i, j, n;
    char i2c_device[255];

    if (argc < 3)
    {
        printf("Usage : accessMCP [DEVICE] [Addr]hex\n");
        printf("Ex   : accessMCP /dev/i2c0 42\n");
        return (-1);
    }

    strcpy(i2c_dev, argv[1]);
    if ((fd = open(i2c_device, O_RDWR)) < 0)
    {
        perror("open");
        printf("Error opening %s\n", i2c_device);
        return (-1);
    }

    printf("\n--- Test d'écriture I2C ---\n\n");

    sscanf(argv[2], "%x", &n);
    ret = ioctl(fd, I2C_SLAVE, n);
    if (ret < 0)
    {
        perror("I2C_SLAVE ioctl cmd");
        close(fd);
        return (-1);
    }
}
```

uClinux – User Guide

```
/* Paramétrage du MCP23016 : 16 bits en sortie */
data[0] = 0x06; data[1] = 0x00; write(fd, &data[0], 2); // IODIR0
data[0] = 0x07; data[1] = 0x00; write(fd, &data[0], 2); // IODIR1
data[0] = 0x02; data[1] = 0xff; write(fd, &data[0], 2); // IOLAT0
data[0] = 0x03; data[1] = 0xff; write(fd, &data[0], 2); // IOLAT0

for (i = 0, j = 0; i < sizeof(tab); i++)
{
    data[0] = 0x02;
    data[1] = tab[i];
//    printf("Mise a ON des segments (@MCP23016:0x%x / Reg= 0x%02x / Data=
//    0x%02x)\n", n, data[0], data[1]);

    write(fd, &data[0], 2);
    if (i == 9)
    {
        i = 0;
        j++;
        data[0] = 0x03;
        data[1] = tab[j];
        write(fd, &data[0], 2);
    }
    if ((i == 6) && (j == 4)) break;
    usleep(50000);
}
close(fd);
return 0;
}
```

L'exécution du code se déroule en 3 temps :

- ✓ Analyse des paramètres , ouverture de l'interface spécifiée (/dev/i2c0) et indication au driver de l'adresse de la cible à l'aide de la commande « ioctl »
- ✓ Paramétrage du MCP23016 en écrivant dans les registres 6 et 7 pour la direction des 16 bits d'IO, et les registres 2 et 3 pour la valeur affectée aux bits (0 à la fin de cette séquence d'initialisation)
- ✓ Une boucle FOR dont le but est de créer un compteur sur les 2 afficheurs allant de « 00 » à « 64 », ce qui correspond à la valeur apparente sur la photo située en début de document.

Un délai de 50000µs soit 50ms est demandée entre 2 incréments du compteur afin de ne pas faire défiler le compteur trop rapidement.

Comme vous pouvez le constater les actions spécifiées au driver i2c sont en tout point similaires à une manipulation de fichier (open, close, read, write,...) avec en plus l'utilisation de la commande « ioctl » dont le but est de préciser l'adresse I2C du MCP23016.

Pour faire fonctionner ce code, tapez sur la cible la commande : `accessMCP /dev/i2c0 42.`

uClinux – User Guide

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/i2c/S3C44/util de la distribution uClinux Pragmatec :

```
DISTRIB_DIRECTORY = ../../../../..
UCLIBC_DIR = $(DISTRIB_DIRECTORY)/lib/uClibc
INCLUDEDIR_LINUX_ = $(DISTRIB_DIRECTORY)/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 = accessMCP.c
OBJ_WR = accessMCP.o

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

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

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 trouve les fichiers accessMCP.c et Makefile.

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

Chargement du programme

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

accessMCP /dev/i2c0 42