Installation de µClinux sur une carte Altera DE2-115 (3 h)

Objectifs

Ce TP vise à découvrir comment configurer un système d'exploitation Linux pour un système embarqué. Vous aurez également à tester un pilote de périphérique opérant avec le système d'exploitation que vous avez configuré.

1 Prérequis

Pour réaliser ce TP, il vous faut au préalable terminer le TP de l'UCE « Concepts et développement », demandant de configurer votre carte Altera à l'aide du logiciel QSys pour instancier un processeur SoftCore NiosII. Il vous faudra en particulier avoir à disposition un fichier .sopcinfo (nécessaire à la configuration du noyau µClinux) et un fichier .sof (nécessaire à la configuration du FPGA).

2 Installation et configuration de µClinux

Cette section explique comment installer et configurer la version µClinux utilisant la MMU à partir d'un environnement de développement Linux.

2.1 Clonage des dépôts git

L'ensemble des dépôts sont en principe déjà chargés sur votre PC dans le répertoire `/home/administrateur/TP-uClinux`.

Veuillez à ce que le répertoire contenant le noyau Linux 2.6 s'appelle `linux-2.6` et qu'il soit situé au même niveau que le répertoire `uClinux-dist`, sous peine d'avoir ultérieurement une erreur lors de la configuration de Linux avec `make menuconfig`.

Vérifiez que la version disponible de `make` sur le PC soit bien 3.81 ou une version antérieure.

2.2 Préparation du fichier dts

Un *Device Tree* permet de décrire le matériel dans un système embarqué. Ses informations sont à fournir au noyau durant la configuration, c'est pouquoi il vous est demandé de le créer avant de démarrer la configuration proprement dite de µClinux. QSys fournit un fichier .sopcinfo qui doit être converti vers un fichier .dts (*Device Tree*). Certaines informations ne pouvant pas être spécifiées dans le fichier .sopcinfo (comme les partitions flash, l'endroit depuis lequel le noyau doit être exécuté, les adresses MAC...), le fichier .dts peut prendre en compte des fichiers *boardinfo* additionels [1].

1. Se reporter à `http://www.alterawiki.com/wiki/Sopc2dts` pour avoir de plus amples informations.

Lancez l'interface graphique de SOPC2DTS à l'aide de la commande suivante exécutée depuis le répertoire contenant le dépôt de SOPC2DTS :

```
cd <rep installation>/sopc-tools/sopc2dts [2]
java -jar sopc2dts.jar --gui
```

Sélectionnez « *Choose file* » puis ouvrez votre fichier .sopcinfo de manière à obtenir un écran qui ressemble à la figure 1. Vérifiez que votre composant `LED` soit présent à droite dans la liste des composants et ait une couleur verte en arrière plan.

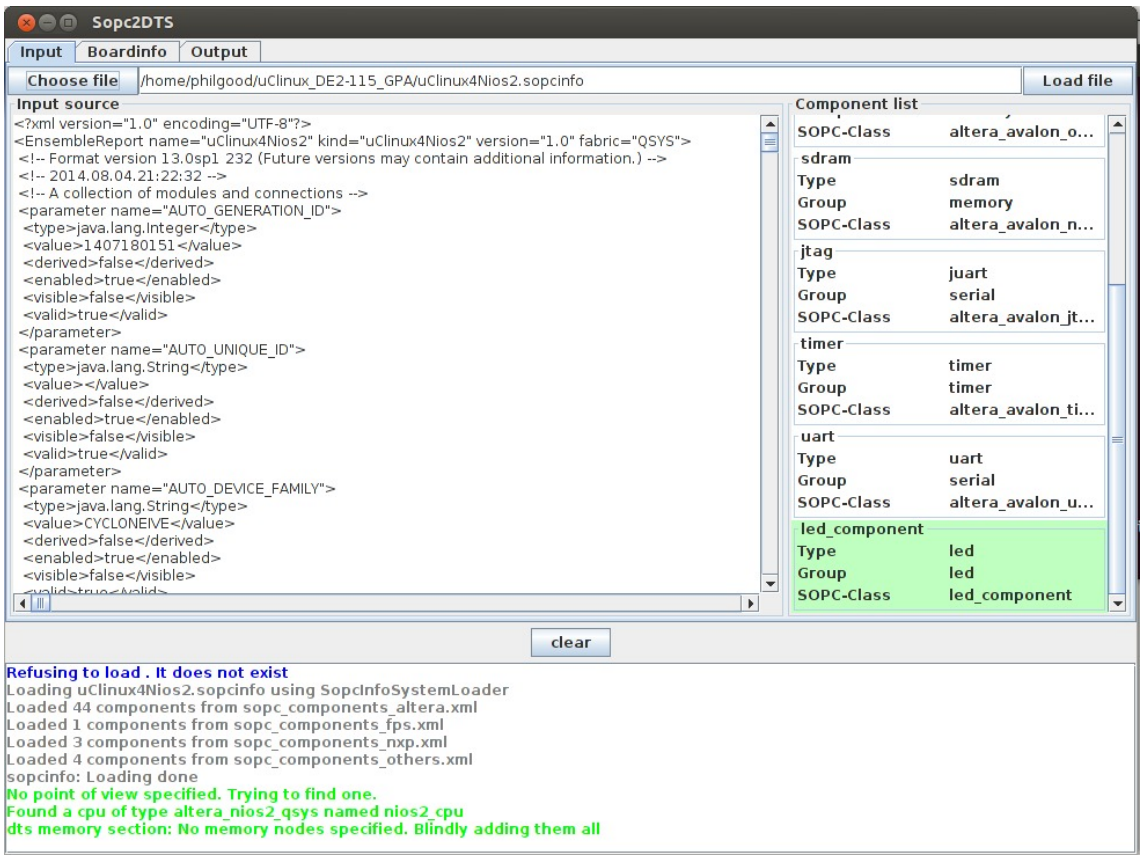

FIGURE 1 – Interface graphique de SOPC2DTS à l'ouverture du fichier .sopcinfo.

Il reste maintenant à configurer la manière dont est chargé le noyau au démarrage. Allez dans l'onglet « *Boardinfo* ». Sélectionnez « cpu : nios2_cpu » dans le champ « *Select master* ». Cliquez sur « *Apply* », ce qui devrait fixer le champ par défaut « *Kernel Bootargs* » à `debug console=ttyAL0,115200` (voir Fig. 2). Cette option indique que l'OS est chargé depuis la SDRAM et définit l'UART comme port de communication par une console série.

Enfin, sélectionnez l'onglet « *Output* » et faites une sauvegarde dans un fichier .dts. La fin de ce fichier devrait avoir un contenu identique à celui de la figure 3.

2. Le répertoire d'installation `<rep installation>` est en principe `/home/administrateur/TP-uClinux`.

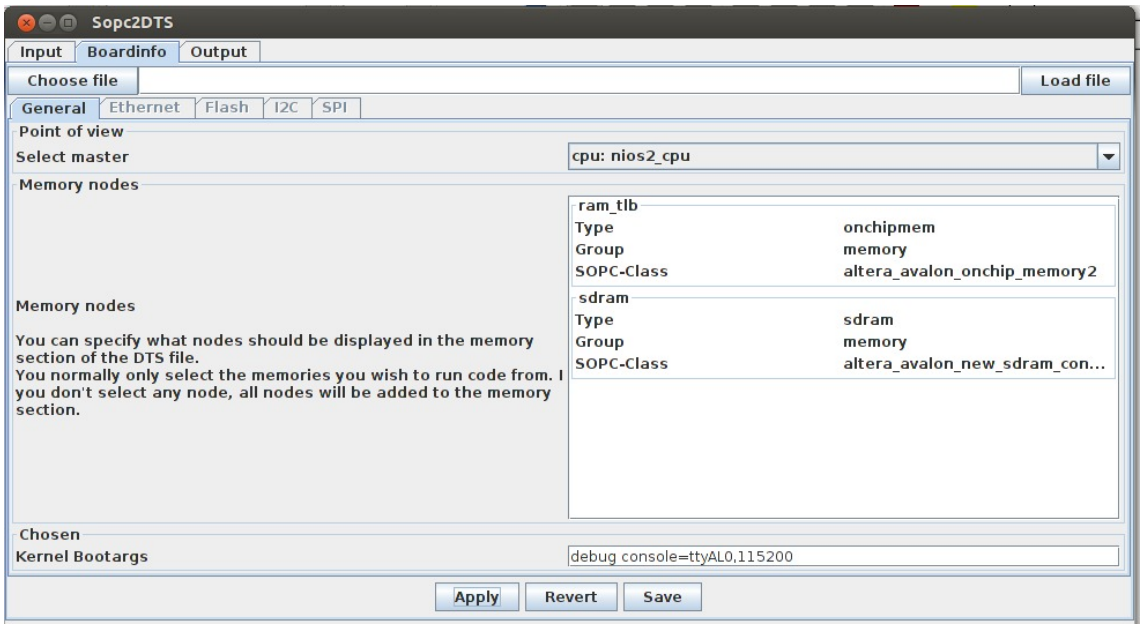

FIGURE 2 – Affichage de l'onglet « *Boardinfo* » de SOPC2DTS.

```
            clock-frequency = < 50000000 >; /* embeddedsw.CMacro.FREQ type NUMBER */
        }; //end serial@0x8001400 (uart)

        led_component: led@0x8001448 {
            compatible = "ALTR,led-1.0";
            reg = < 0x08001448 0x00000002 >;
        }; //end led@0x8001448 (led_component)
    }; //end sopc@0

    chosen {
        bootargs = "debug console=ttyAL0,115200";
    }; //end chosen
}; //end /
```

FIGURE 3 – Contenu du fichier .dts généré.

2.3 Configuration de µClinux

Avant de configurer le noyau, modifiez la variable d'environnement `PATH` pour que le système de configuration trouve la chaîne de compilation :
`export PATH=$PATH:<rep installation>/toolchain-mmu/x86-linux2/nios2-linux-gnu/bin`

Positionnez-vous dans le répertoire `uClinux-dist` et assurez-vous d'effacer les fichiers générés pour une configuration précédente avec les commandes `make mrproper` et `make clean`.

Lancez l'interface graphique de configuration de µClinux en exécutant la commande :
`make menuconfig`.
Si vous avez une erreur d'exécution, vérifiez que la bibliothèque NCURSES est bien installée :
`sudo apt-get install libncurses5-dev`

Le premier menu de `make menuconfig` permet de chosir la plateforme et le vendeur. Positionnez ces options comme indiqué à la figure 4.

Le deuxième menu permet de préciser ce qui doit être configuré. Sélectionnez uniquement l'item « *Customize Kernel Settings* » (voir Fig. 5) de manière à ne paramétrer que le noyau et à conserver

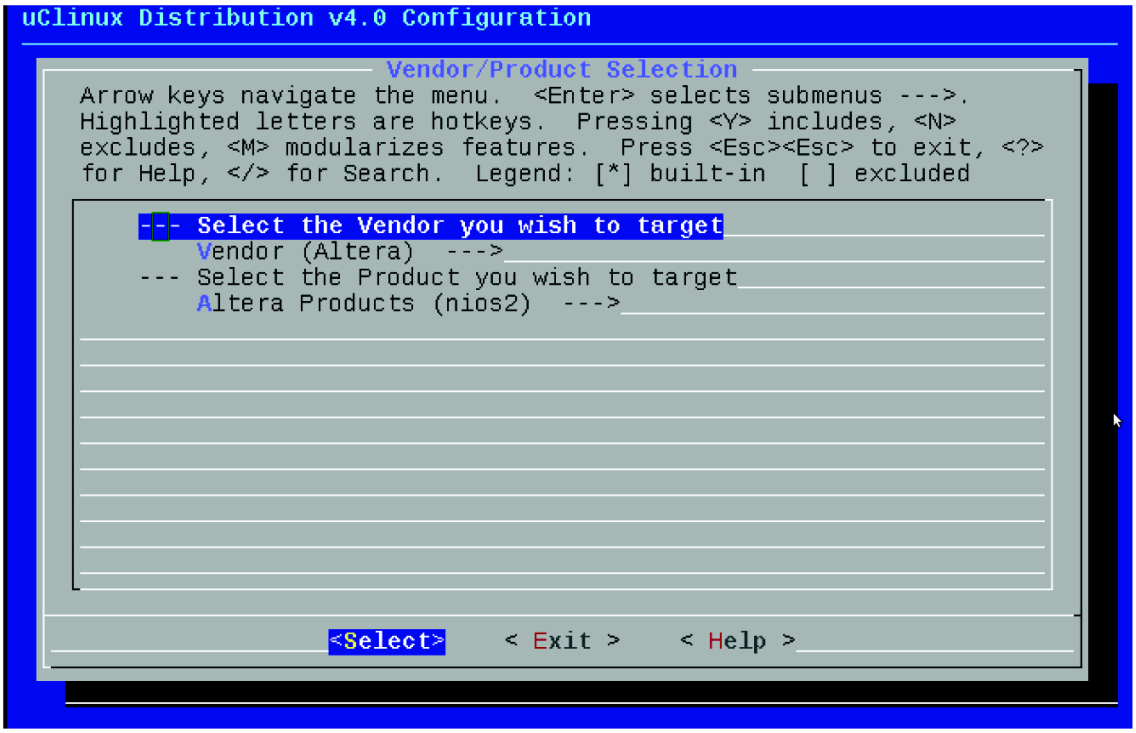

FIGURE 4 – Sélection du vendeur et de la plateforme.

la configuration par défaut des librairies et des applications installées.

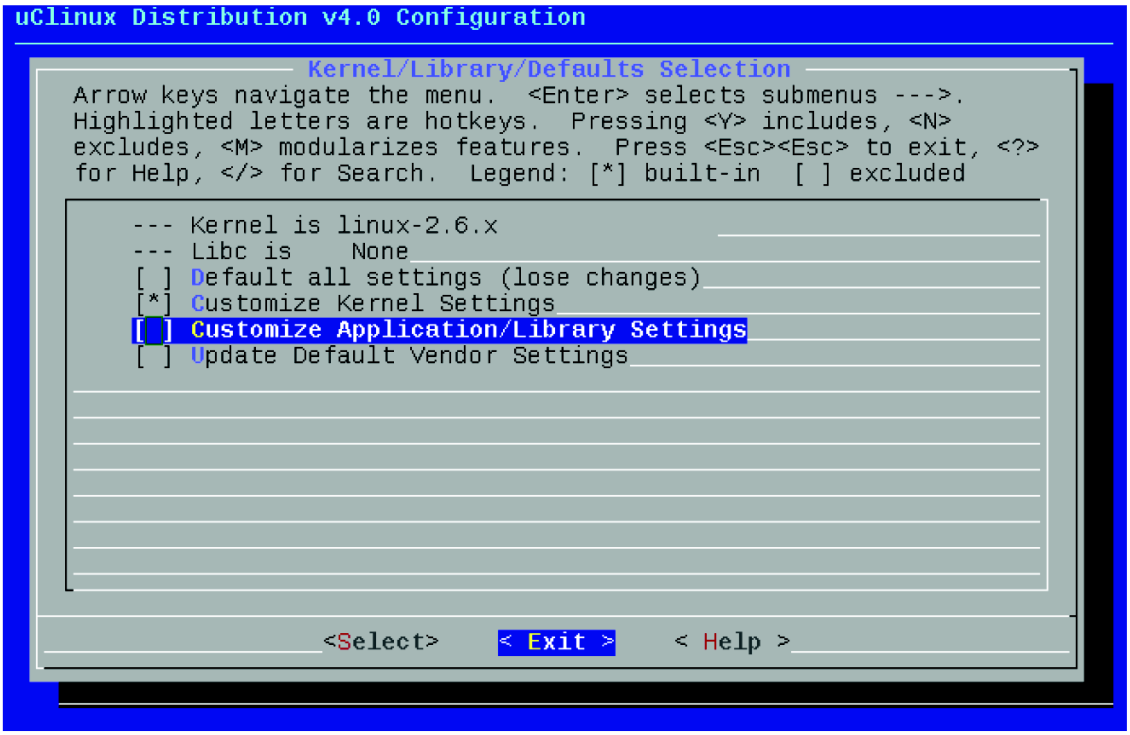

FIGURE 5 – Sélection du mode de configuration.

Quittez le menu. Lors de la première configuration, le terminal réapparait en proposant de régler manuellement différents paramètres. Validez avec entrée tous les paramètres par défaut.

Vous aurez ensuite une nouvelle fenêtre de configuration affichée (voir Fig. 6). Parcourez les différents menus en effectuant les configurations suivantes :

— Fixez l'adresse de base pour la mémoire SDRAM en fonction de l'adresse configurée auparavant avec QSys, qui est ici `0x0` (menu « *Platform options* », variable CONFIG_MEM_BASE) ;
— Indiquez la compilation et l'édition de liens du *Device Tree* (DTB) dans l'image du noyau (menu « *Platform options* », variable CONFIG_DTB_SOURCE_BOOL) ;
— Précisez la localisation du fichier *Device Tree* à prendre en compte (menu « *Platform options* », variable CONFIG_DTB_SOURCE) ;
— Activez le support de la MMU (menu « *Processor type and features* », variable CONFIG_MMU) ;
— Modifiez la fréquence du noyau à 1 000 Hz (menu « *Kernel features* », CONFIG_HZ_1000) ;
— Vérifiez que les communications séries UART Altera sont opérationnelles, avec et sans JTAG comme montré sur la figure 7 (menu « *Device drivers* » → « *Character devices* » → « *Serial drivers* », variables CONFIG_SERIAL_ALTERA_UART et CONFIG_SERIAL_ALTERA_JTAGUART) ; cette option sera utile pour ouvrir un terminal à distance sur la carte en utilisant une liaison série.

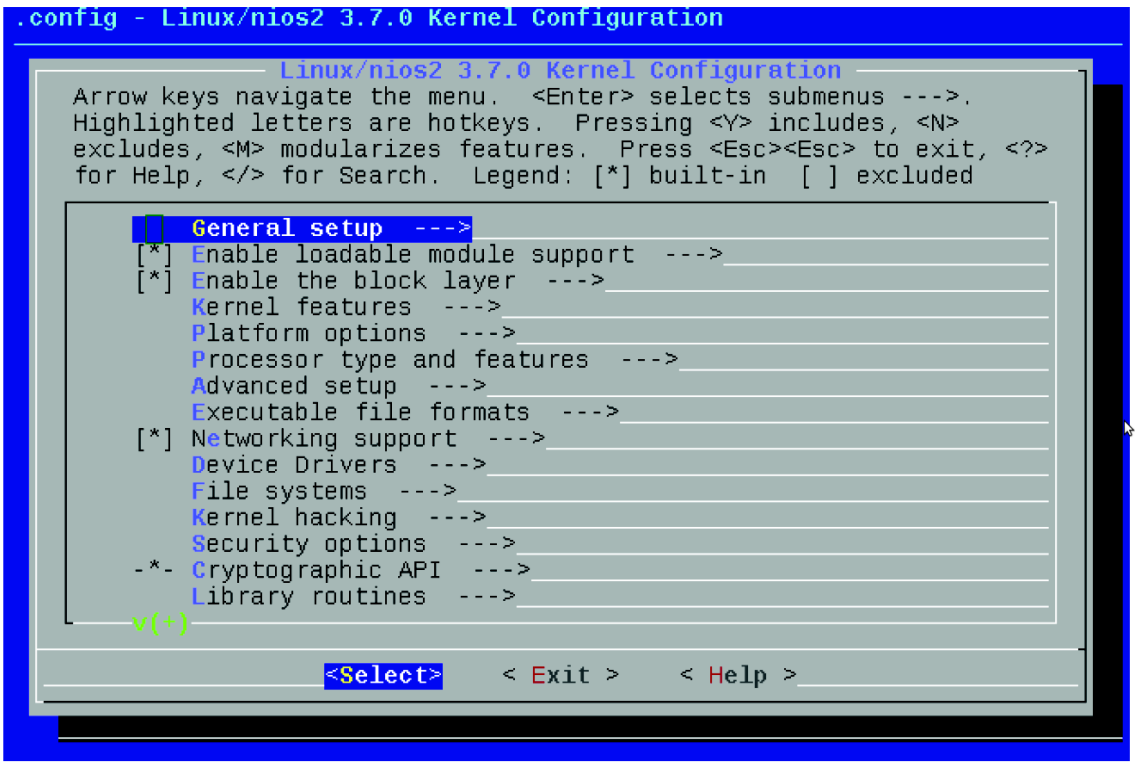

FIGURE 6 – Menu principal de configuration du noyau.

Vérifiez que l'ensemble des configurations ont été réalisées en saisissant / dans la fenêtre de configuration pour visualiser chacune des variables CONFIG_* mentionnées ci-dessus.

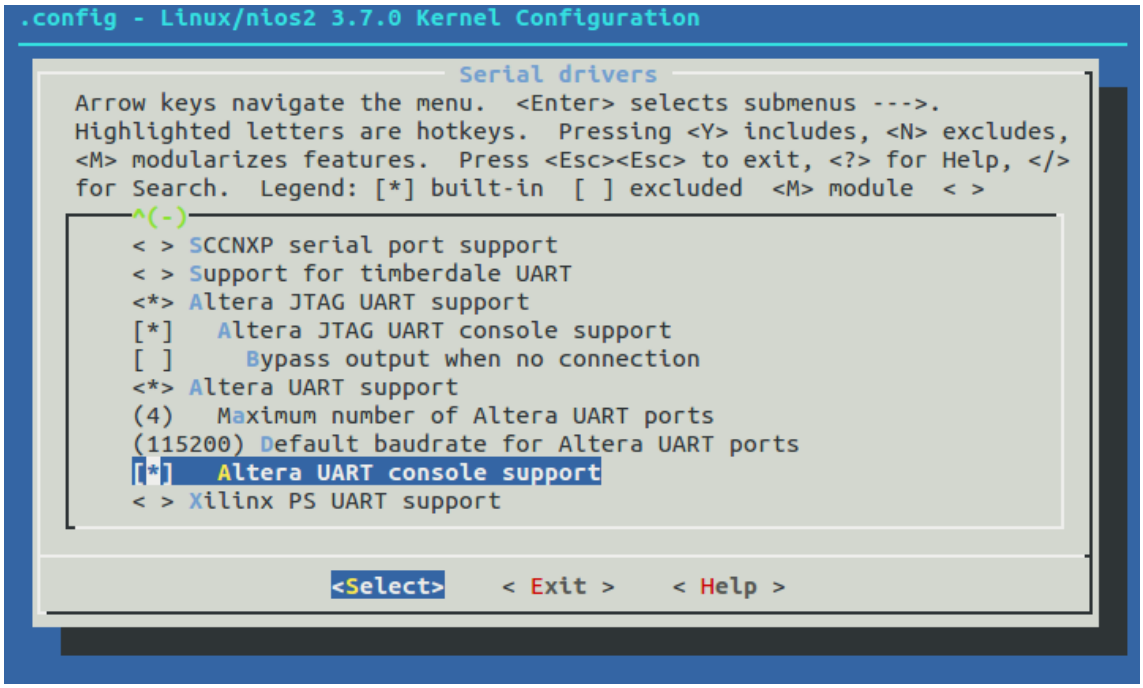

FIGURE 7 – Menu de configuration des consoles séries.

2.4 Compilation de µClinux

Maintenant que la configuration est effectuée, la compilation du système peut avoir lieu. Plusieurs bibliothèques sont nécessaires pour cette compilation. Voici une liste de paquets nécessaires sous Ubuntu :

```
sudo apt-get install uuid-dev git-core git-gui gcc bison \
flex gawk gettext ccache zlib1g-dev libx11-dev texinfo liblzo2-dev pax-utils \
uboot-mkimage corkscrew libc6-i386
```

La compilation se fait avec les commandes :

```
export PATH=$PATH:<rep installation>/toolchain-mmu/x86-linux2/bin
make
```

à saisir au niveau du répertoire `uClinux-dist`.

Si la compilation de µClinux s'est déroulée correctement, vous devriez voir apparaître la ligne suivante :

```
Kernel: arch/nios2/boot/zImage is ready
```

Ce fichier `zImage` est une image du système d'exploitation qui pourra être flashée directement dans la mémoire de la carte. Copiez ce fichier (situé dans le répertoire `uClinux-dist/linux-2.6x/arch/nios2`...) vers un répertoire personnel.

2.5 Chargement de µClinux sur la carte

Dans cette section, le chargement de µClinux est effectué dans la SDRAM de la carte. Le PC est à relier à la carte Altera au niveau de l'entrée USB-Blaster par un câble USB AB.

L'ensemble des scripts de chargement de fichiers sur la carte sont situés dans le répertoire `~/altera/13.0sp1/nios2eds`. Positionnez-vous dans ce répertoire. Ajoutez également les commandes

suivantes dans le PATH :
`export PATH=$PATH:~/altera/13.0sp1/nios2eds/bin/gnu/H-i686-pc-linux-gnu/bin/`
`export PATH=$PATH:~/altera/13.0sp1/nios2eds/bin/`
`export PATH=$PATH:~/altera/13.0sp1/quartus/linux`
et mettez à jour la variable LD_LIBARY_PATH :
`export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/altera/13.0sp1/quartus/linux`.

Le fichier .sof généré avec QUARTUS doit tout d'abord être envoyé vers la carte.
`nios2-configure-sof <répertoire>/<fichier>.sof`
`<fichier>.sof` étant réalisé lors du TP de Mr GOZLAN.

Il faut ensuite charger le noyau en mémoire.
`nios2-download -g <répertoire>/zImage`

Pour vérifier ce qui se passe sur la carte et interagir avec elle, un câble DB9 mâle/femelle est à utiliser pour établir la communication entre le PC et la carte DE2-115. Vous pouvez ouvrir un terminal à distance sur la carte à l'aide par exemple du logiciel CUTECOM. Pour vous assurer du port utilisé (`ttyS0` normalement), vous pouvez utiliser la commande
`dmesg | grep tty`.

Lancez le démarrage de µClinux sur la carte en exécutant la commande :
`nios2-terminal`.
Si tout fonctionne, vous devriez voir apparaître dans le terminal de CUTECOM un résultat similaire à la figure 8 et témoignant de la décompression de l'image de l'OS puis de son initialisation.

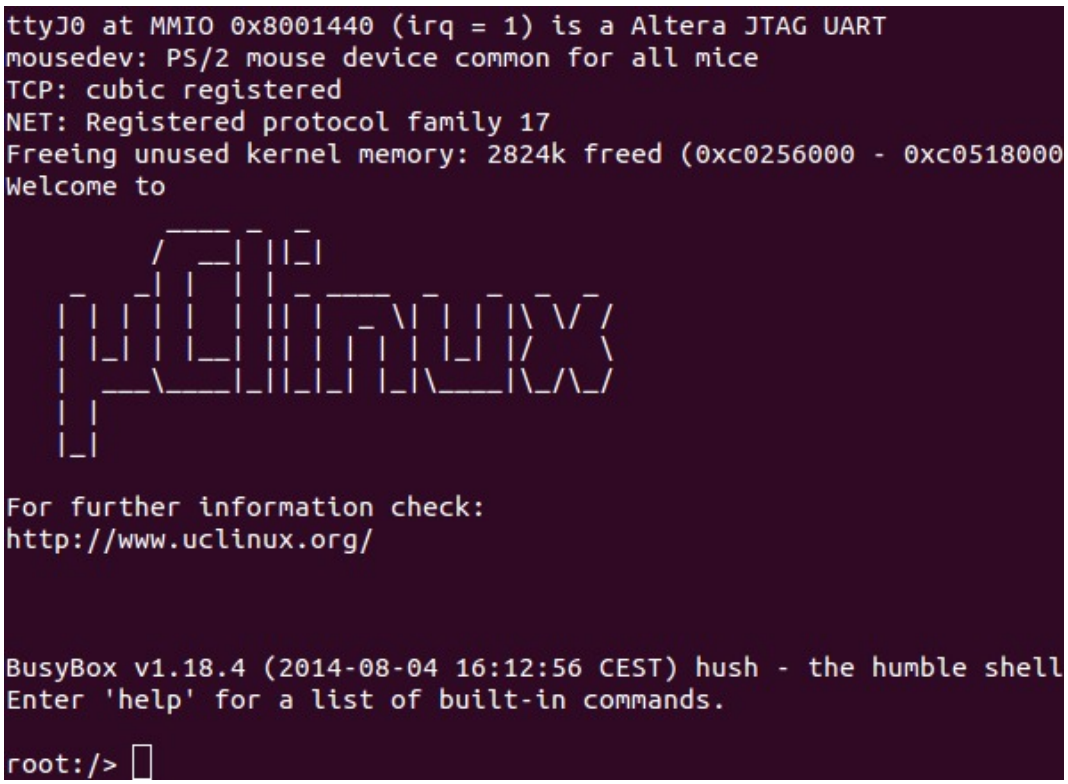

FIGURE 8 – Lignes s'affichant au démarrage de µClinux.

3 Configuration du noyau pour intégrer un pilote de périphérique

Maintenant que le système Linux est configuré et fonctionnel, il est possible de lui ajouter une prise en charge du matériel qui a été ajouté dans QSys (le composant `LED`). Dans cette partie du TP, vous aurez à modifier la configuration du noyau et à ajouter un pilote pour ce composant.

3.1 Configuration du noyau

Pour inclure le code du pilote lors de la compilation du noyau, il est néccessaire d'apporter quelques modifications.

1. Ajoutez l'option à `Kconfig` dans `<rep installation>/linux-2.6/drivers/misc`.

```
config LED
    tristate "LED custom hardware"
    help
       LED custom hardware
```

2. Ajoutez un fichier objet au `Makefile` dans `<rep installation>/linux-2.6/drivers/misc`

```
obj-$(CONFIG_LED) += led.o
```

3.2 Ajout de fichiers de périphériques au système de fichiers racine

Le pilote utilise un fichier de périphérique (*device file*) pour communiquer avec le matériel. Pour créer un tel fichier, ajoutez la ligne suivante dans `uClinux-dist/vendors/Altera/nios2/device_table.txt` :

```
/dev/led   c   666    0     0  250    0  -  -  -
```

3.3 Ajout du code source

Le code source du pilote de périphérique doit être écrit dans `linux-2.6/drivers/misc` et appelé `led.c`. Reportez-vous à l'annexe pour le code C du pilote. Ce code est également téléchargeable sur le site e-uapv.

Pour mettre en œuvre l'affichage des LED, complétez le code fourni aux endroits où il y a des commentaires :

— fonction `write_led()`,
— fonction `drv_probe()`,
— fin du pilote.

Vous pourrez vous inspirer de la documentation trouvée sur Internet et notamment du cours suivant : `http://www.cs.columbia.edu/~sedwards/classes/2014/4840/device-drivers.pdf`.

3.4 Compilation du noyau

Pour inclure le pilote, le noyau doit être reconfiguré puis recompilé. Positionniez-vous dans le répertoire `uClinux-dist` puis resaisissez les commandes suivantes :

```
make menuconfig
make
```

Lors de cette première étape, ajoutez une étoile en regard de « *LED custom hardware* », accessible par les menus « *Device drivers* » puis « *Misc Devices* » (voir Fig. 9). Suivant le mode de configuration

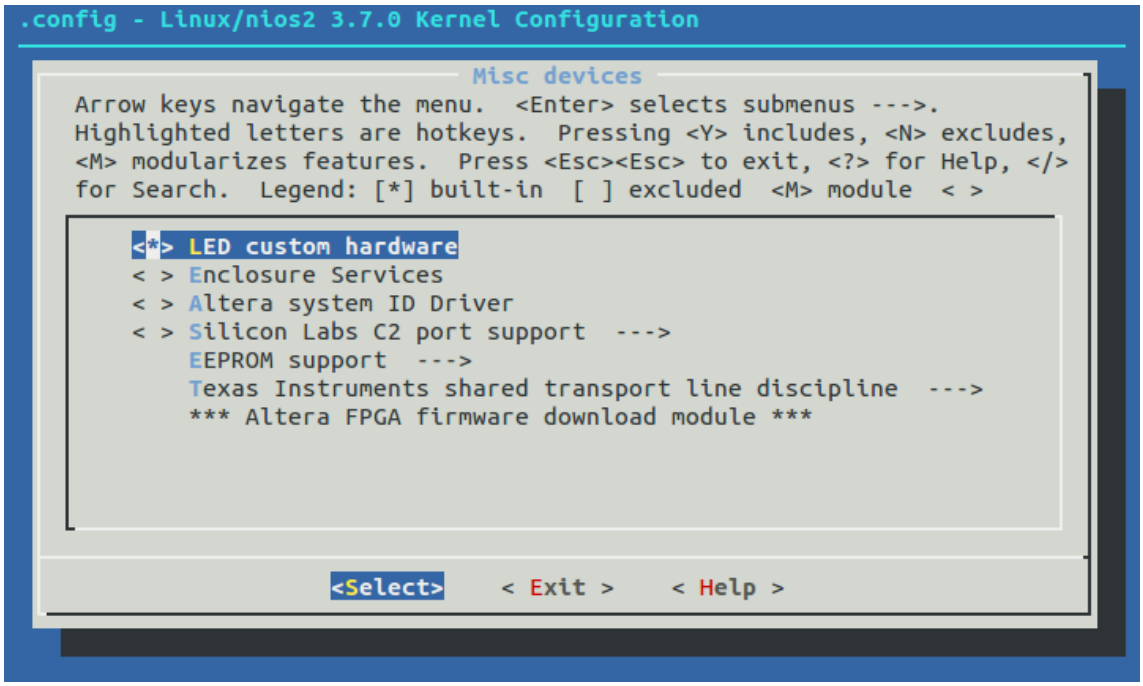

FIGURE 9 – Menu de configuration du module `LED`.

choisi, il se peut également que des questions vous soient posées directement dans le terminal (Fig. 9). Confirmez alors en particulier l'utilisation du module `LED`.

3.5 Test du pilote

Rechargez la nouvelle image du système sur la carte en répétant les opérations décrites à la section 2.5.

Si la compilation du noyau a été faite avec succès, le pilote devrait être capable d'afficher des valeurs sur les LED à partir de l'espace utilisateur du système Linux. Pour tester cette fonctionnalité, entrez la commande suivante dans l'invite de commande de µClinux :

```
echo 1 > /dev/led
```

Les LED devraient afficher 1 en n'allumant que la première LED.

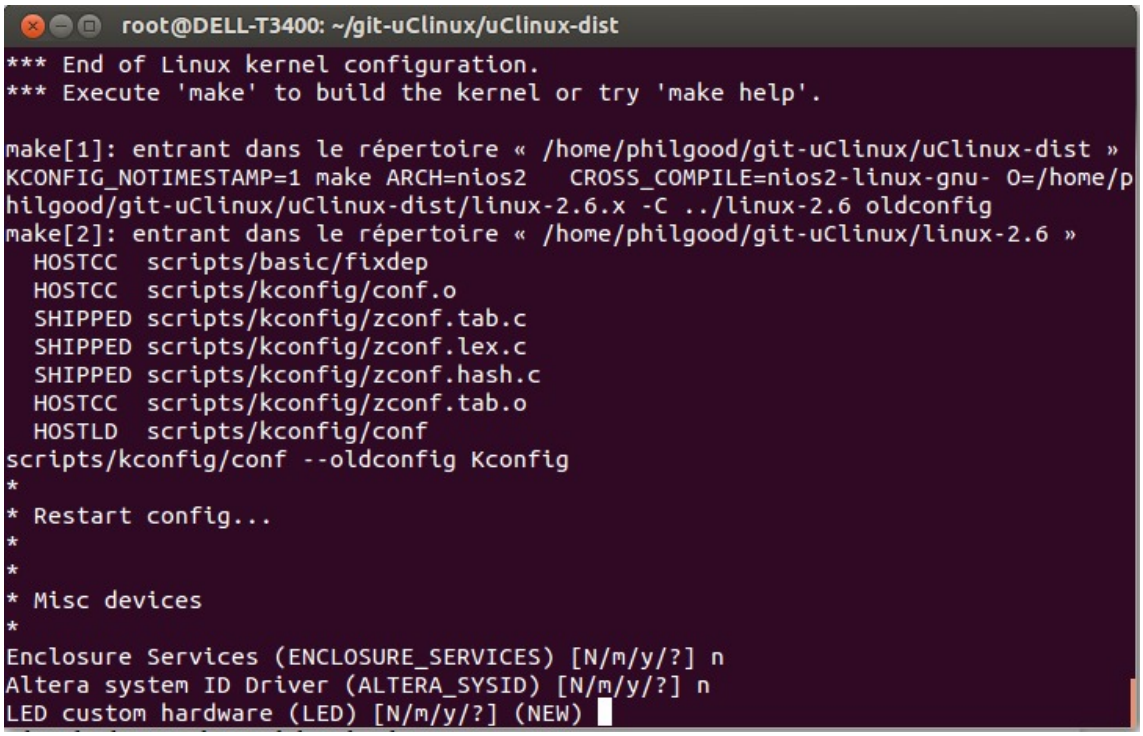

FIGURE 10 – Extrait des sorties de `make menuconfig` avec le module `LED`.

3.6 Nettoyage du code

Remettez les fichiers suivants dans leur état initial, c-à-d avant les ajouts nécessaires à la prise en compte du module `led` :

- `linux-2.6/drivers/misc/Kconfig`
- `linux-2.6/drivers/misc/Makefile`
- `uClinux-dist/vendor/altera/nios2/device_table.txt`

4 Annexe : Code C du pilote de périphérique pour le composant LED

```
/* Driver du moduleLed pour noyau Linux*/
/* TP uCLinux sur DE2-115
 */

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/mod_devicetable.h>
#include <linux/platform_device.h>
#include <linux/of_device.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/ioport.h>
#include <asm/uaccess.h>
#include <asm/io.h>

MODULE_AUTHOR("Johan␣Granath/GPA2015");
MODULE_LICENSE("GPL");
```

```
MODULE_DESCRIPTION("LEDG[7..0] driver for Altera DE2-115");
MODULE_SUPPORTED_DEVICE("none");

#define LED_MAJOR 250

static ssize_t write_led(struct file *, const char *, size_t, loff_t *);
static char led_str[3];
int *led_mem=0;


static struct of_device_id of_match_led[] __devinitdata =
{
        { .compatible = "ALTR,led-1.0", },
        {}
};

MODULE_DEVICE_TABLE(of, of_match_led);

static struct file_operations fops_led =
{
        .write = write_led,
};

static ssize_t write_led(struct file *fp, const char *buf, size_t len, loff_t *offset)
{
        int not_copied, led_value;

        // utilisation de copy_from_user pour remplir led_str
        // not_copied= ???

        led_str[len] ='/0';

        // conversion de led_str en led_value

        // ecriture de la variable led_value a l'adresse led_mem

        return len-not_copied;
}

static int __devinit drv_probe(struct platform_device *op)
{
        struct resource *res;
        if(!of_match_device(of_match_led, &op->dev))
                return -ENODEV;
        res = platform_get_resource(op, IORESOURCE_MEM, 0);
        if(!res)
                return -ENODEV;
        if(!request_mem_region(res->start, resource_size(res), "led"))
                return -ENOMEM;

        // remplissage de la variable led_mem


        if(!led_mem)
                return -ENOMEM;
        if(register_chrdev(LED_MAJOR, "led", &fops_led))
        {
                printk("register_chrdev: led failed\n");
                return -EIO;
        }
        return 0;
}

static int __devinit drv_remove(struct platform_device *op)
{
        struct resource *res;
        res = platform_get_resource(op, IORESOURCE_MEM, 0);
        if(!res)
```

```
                return -ENODEV;
                release_mem_region(res->start, resource_size(res));
        return 0;
}

static struct platform_driver platform_driver_led =
{
        .probe = drv_probe,
        .remove = drv_remove,
        .driver =
        {
                .name = "led",
                .owner = THIS_MODULE,
                .of_match_table = of_match_led,
        },
};


static int __init mod_init(void)
{
        int ret;
        ret = platform_driver_register(&platform_driver_led);
        return ret;
}



static void __exit mod_exit(void)
{
        platform_driver_unregister(&platform_driver_led);
        unregister_chrdev(LED_MAJOR, "led");
}

// specification du module_init et du module _exit utilises
```