EUKREA Electromatique

CPUAT91

Guide de démarrage rapide

Référence : 070710_AT91_QUICKSTART_012 - Version : 1.2 / A

Ecrit par	**Vérifié par**
Eric BENARD	
(Eukréa)	()
Diffusion	**Diffusion**
( )	( )

CONFIDENTIEL

Contacter EUKREA	Téléphone :	+33 (0) 5 56 46 26 68
	Télécopie :	+33 (0) 5 56 46 04 77
	Email :	at91@eukrea.com
	Site Web :	www.eukrea.com

1- GESTION DU DOCUMENT

1.1- IDENTIFICATION

Nom du fichier	070710_AT91_QUICKSTART_012.sxw
Version	1.2 / A
Date de Création	10/07/2007
Auteur	EBD
Dernière Modification	12/07/2007
Status	Conforme à la v 2.0

1.2- HISTORIQUE DES MODIFICATIONS

Date Ed. / Rev.	***Pages modifiées***	***Modifications***	***Auteur***
05/07/2005 10 / A	Version Initiale	toolchain – u-boot – kernel	EBD
12/07/2005 11 / A	14-16; 19-22	Filesystem	EBD
10/07/2007 12 / A	-	Mise à jour correspondant au dernier package logiciel	EBD

2- TABLE DES MATIERES

3- DOCUMENTS DE REFERENCE

[CPUDOC]	ATMEL – AT91RM9200 Datasheet – doc1768.pdf
[CPUERRATA]	ATMEL – AT91RM9200 Errata Sheet – doc1768.pdf
[SDRAM]	MICRON – SDRAM – 256MSDRAM.pdf + Addendum
[FLASH]	INTEL – 28F128J3D_20855104.pdf + SpecUpdt
[PHY]	MICREL – KS8721BL - ks8721bl-sl.pdf
[EEPROM]	ST – M24C08 - M24C08-W.pdf

4- TERMINOLOGIE

5- INTRODUCTION

5.1- Objectif de ce Document

L'objectif de ce document est de permettre à l'acquéreur d'un kit de développement pour modules CPUAT91 de démarrer rapidement avec le module.

5.2- Lien avec d'autre documents

Les documents nécessaires à la compréhension du présent document sont :

- documentations techniques des composants.

5.3- Conventions

Code :

```
ceci est du code
```

Instructions à envoyer sur la console série du bootloader de la carte :

```
CPUAT91> instruction bootloader
```

Instruction à envoyer sur la console série Linux de la carte :

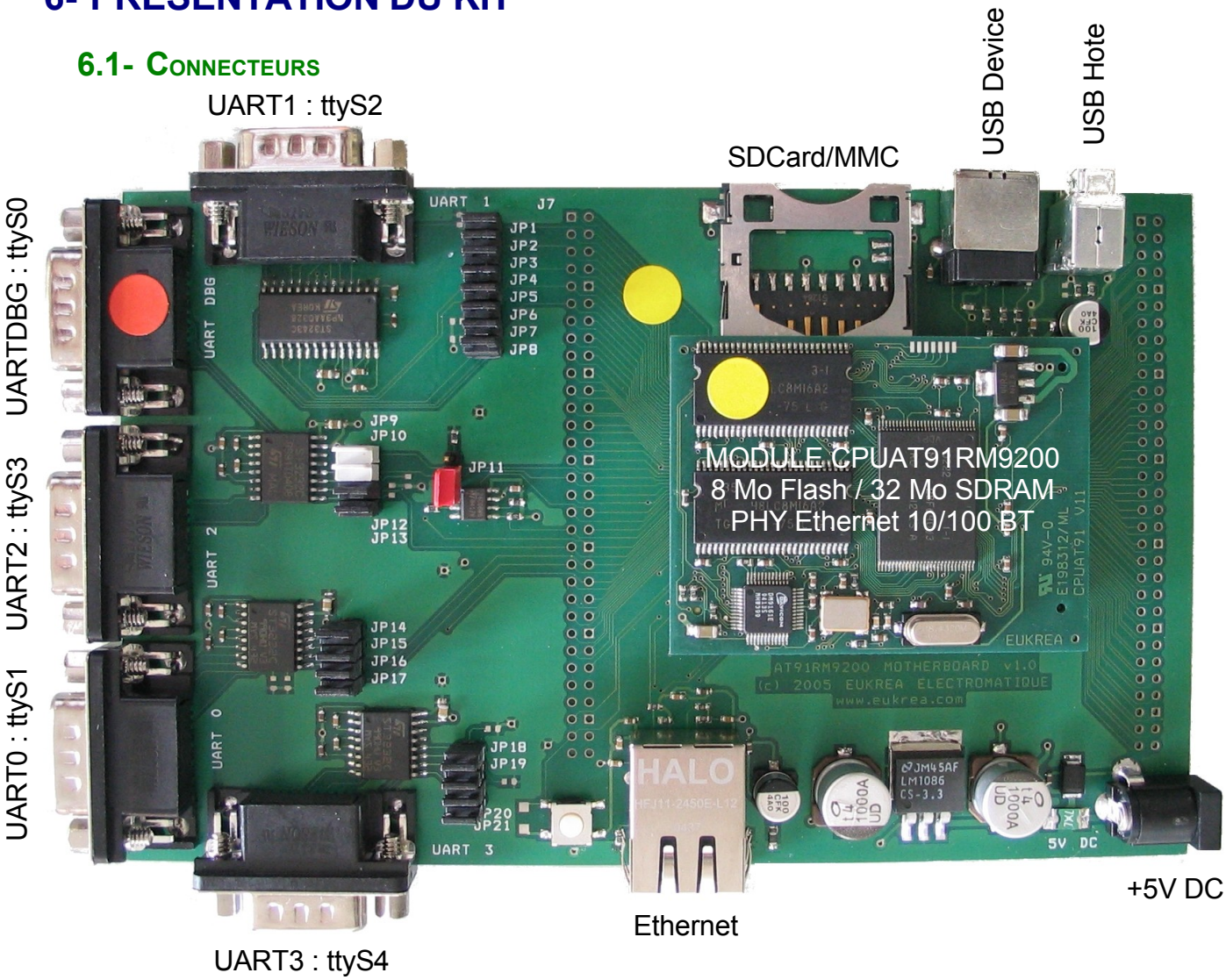
```
# instruction linux
```

Instruction à saisir sur l'hôte :

```
$ instruction hôte
```

6- PRESENTATION DU KIT

6.1- CONNECTEURS



6.2- AFFECTATION DES JUMPERS

UART0 : ttyS1	
JP14	RX
JP15	TX
JP16	CTS
JP17	RTS

UARTDBG : ttyS0	
JP9	RX
JP10	TX

UART1 : ttyS2	
JP1	RTS
JP2	DSR
JP3	CTS
JP4	DCD
JP5	RX
JP6	TX
JP7	DTR
JP8	RI

UART2 : ttyS3	
JP12	TX
JP13	RX

UART3 : ttyS4	
JP18	RX
JP19	TX
JP20	CTS
JP21	RTS

JP11 : BOOTMODE

6.3- Utilisation des GPIO

6.3.1- Interne au module

PA7 à PA16 : MAC Ethernet en mode RMII

6.3.2- Externe au module : avec tous les jumpers installés

PA25 & PA26 : I2C

PA30 & PA31 : UART Debug

PA17 & PA18 & PA20 & PA21 : UART 0

PB18 & PB19 & PB20 & PB21 & PB23 & PB24 & PB25 & PB26 : UART 1

PA22 & PA23 : UART 2

PA5 & PA6 & PB0 & PB1 : UART 3

PA27 & PA28 & PA29 & PB3 & PB4 & PB5 : SDCARD / MMC

PC2 : Détection d'insertion d'une carte SD/MMC

PC14 : Ré-énumération USB Device

PC15 : Détection de connexion de l'USB Device à un hôte

6.3.3- Recommandations

Attention à l'utilisation de PC6 (nWAIT) : cf [CPUERRATA].

Dans une configuration minimale (module télécommandé par le réseau), seules les GPIO PA7 à PA16 sont non disponibles pour l'utilisateur.

7- PRE REQUIS

Ce chapitre permet de valider l'environnement installé sur l'hôte de développement :

- compilateur,
- serveur tftp,
- terminal série,
- outils de transfert xmodem.

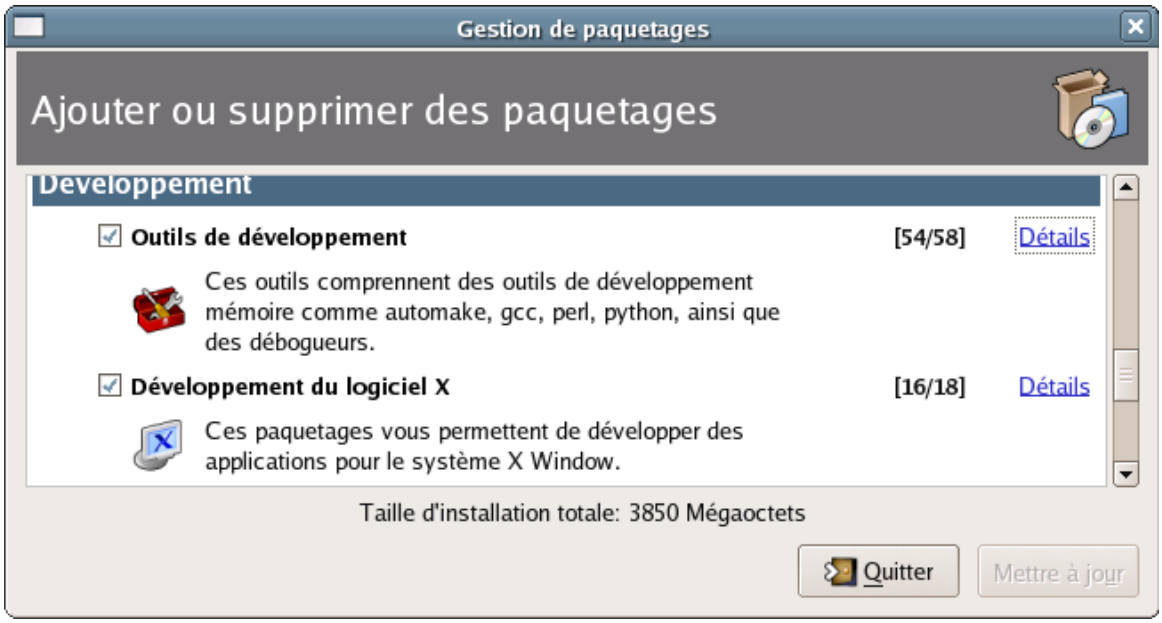
7.1- Distribution Fedora Core 4 ou 5

7.1.1- Compilateur

S'assurer que les outils gcc sont installés :

```
$ gcc -v
Utilisation des specs internes.
Target: i386-redhat-linux
Configuré avec: ../configure --prefix=/usr --mandir=/usr/share/man --
infodir=/usr/share/info --enable-shared --enable-threads=posix --enable-
checking=release --with-system-zlib --enable-__cxa_atexit --disable-
libunwind-exceptions --enable-libgcj-multifile --enable-
languages=c,c++,objc,java,f95,ada --enable-java-awt=gtk --with-java-
home=/usr/lib/jvm/java-1.4.2-gcj-1.4.2.0/jre --host=i386-redhat-linux
Modèle de thread: posix
version gcc 4.0.0 20050519 (Red Hat 4.0.0-8)
```

Dans le cas où gcc ne serait pas installé ou des problèmes de librairies seraient rencontrés, nous recommandons d'utiliser l'outil d'installation d'applications : Environnement de bureau > Paramètres du système > Ajouter/Supprimer des applications



Sélectionner « Outils de développement « et « Développement du logiciel X ».

Nous recommandons aussi de sélectionner « Développement de logiciel KDE » et « kdevelop » (en suivant le lien « Détails ».

7.1.2- SERVEUR TFTP

Les opérations suivantes sont à effectuer en **root**.

```
$ yum install tftp-server
.../...
Installed: tftp-server.i386 0:0.40-6
Dependency Installed: xinetd.i386 2:2.3.13-6
Complete!
```

Editer la configuration de xinetd afin d'activer le serveur tftp.

```
$ nano /etc/xinetd.d/tftp
# default: off
# description: The tftp server serves files using the trivial file transfer \
#       protocol.  The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network-aware printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
        disable                 = no
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

La séquence CTRL-O + CTRL-X permet de sauver et quitter

```
$ /etc/init.d/xinetd restart
Arret de xinetd :                                          [  OK  ]
Démarrage de xinetd :                                      [  OK  ]
```

Définir le propriétaire du répertoire /tftpboot : pour simplifier, le développeur aura les droits sur le répertoire /tftpboot

```
$ chown -R mon_login:mon_login /tftpboot/
```

7.1.3- TERMINAL SÉRIE

Droits sur le port série

Il est tout d'abord nécessaire d'autoriser le développeur à accéder au port série :

Editer le fichier /etc/group

Ajouter le nom de l'utilisateur à la ligne uucp:

```
.../...
uucp:x:14:uucp,login_developpeur
.../...
```

La modification sera prise en compte au prochain démarrage.

Installation des terminaux série

Nous allons installer 3 terminaux : gtkterm, minicom et ckermit. Le développeur est libre d'utiliser celui qu'il souhaite, selon ce qu'il souhaite faire.

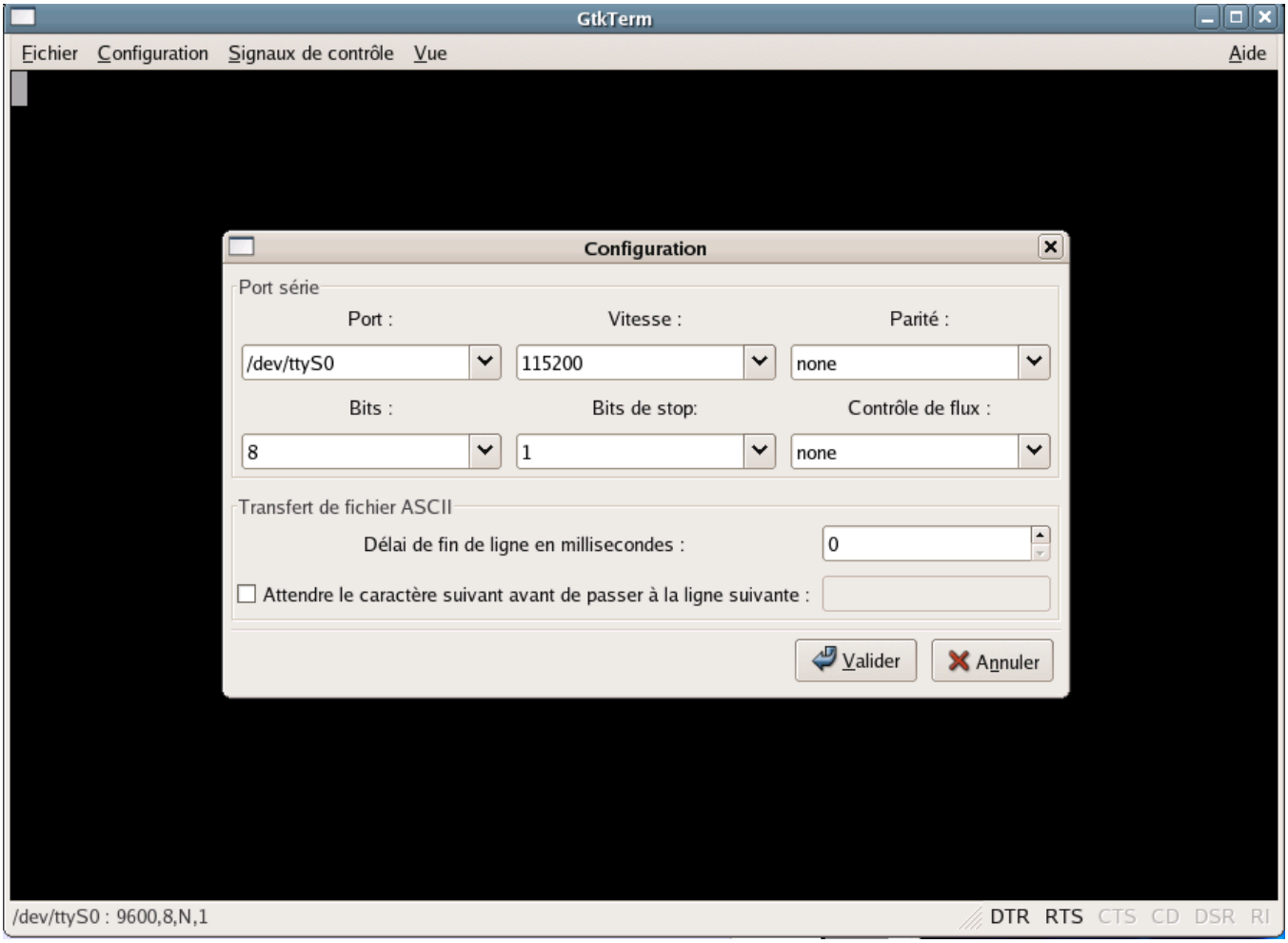
```
$ yum install gtkterm minicom ckermit
.../...
Installed: ckermit.i386 0:8.0.211-1 gtkterm.i386 0:0.99.4-4 minicom.i386
0:2.00.0-21
Complete!
```

Utilisation des terminaux série

La configuration est 115200 / 8 bits / 1 bit de stop / pas de parité / aucun contrôle de flux.

Nous supposons pour la suite que le port série est /dev/ttyS0.

Pour gtkterm : Configuration > Port :



Pour minicom :

l'utilisateur root doit créer le fichier de configuration par défaut :

```
$ minicom -s
```

```
┌──────────[configuration]──────────┐
| Noms de fichers et chemins        |
| Protocoles de transfert           |
| Configuration du port s+Rrie      |
| Modem et appel                    |
| Ecran et clavier                  |
| Enregistrer config. sous dfl      |
| Enregistrer la configuration sous...
| Sortir                            |
| Sortir de Minicom                 |
└───────────────────────────────────┘
```

Aller dans « Configuration du port série » :

A : <- , 0 , ENTER
E : I , ENTER

```
┌──────[Param+tres de communication]──────┐
|                                         |
| Actuellement : 115200 8N1               |
|                                         |
|   Vitesse        Parit+R       Donn+Res |
|                                         |
| A: 300           L: Aucune       S: 5   |
| B: 1200          M: Paire        T: 6   |
| C: 2400          N: Impaire      U: 7   |
| D: 4800          O: Marque       V: 8   |
| E: 9600          P: Espace              |
| F: 19200                 Bits d'arr+t   |
| G: 38400                        W: 1    |
| H: 57600                        X: 2    |
| I: 115200        Q: 8-A-1               |
| J: 230400        R: 7-P-1               |
|                                         |
|                                         |
| Choix, ou <Entr+Re> pour sortir ?       |
└─────────────────────────────────────────┘
```

F

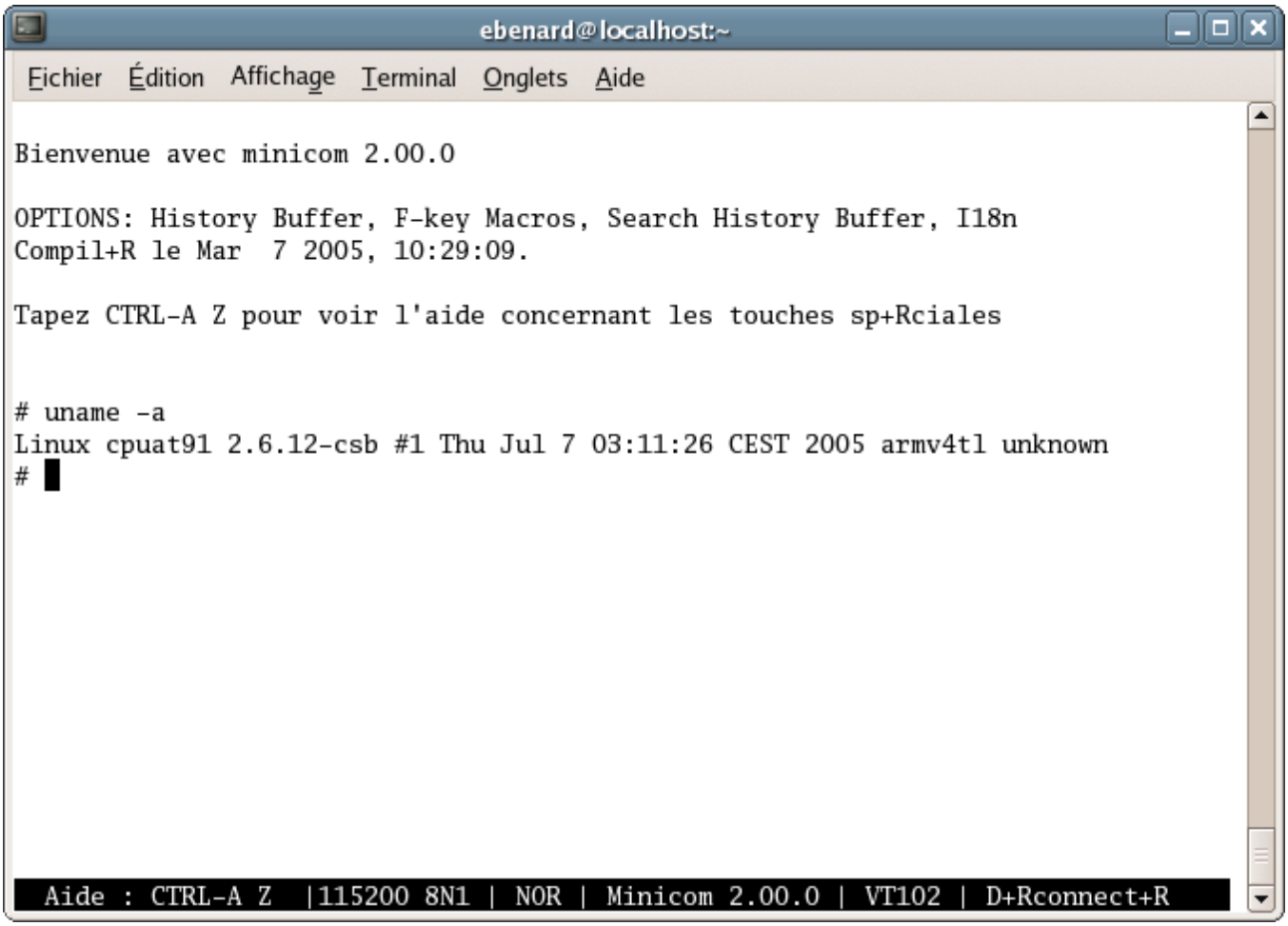
```
┌──────────────────────────────────────────────────────────────┐
| A -                          Port s+Rrie : /dev/ttyS0        |
| B - Emplacement du fichier de verrouillage : /var/lock       |
| C -            Programme d'appel int+Rrieur :                |
| D -            Programme d'appel ext+Rrieur :                |
| E -                      D+Rbit/Parit+R/Bits : 115200 8N1    |
| F -              Contr++le de flux mat+Rriel : Non           |
| G -              Contr++le de flux logiciel : Non            |
|                                                              |
|    Changer quel r+Rglage ?                                   |
└──────────────────────────────────────────────────────────────┘
```

ENTER

Sous « Modem et appel », nous recommandons d'effacer la chaîne d'initialisation et la chaîne de remise à zéro (A & B).

Enfin, choisir « **Enregistrer config. sous dfl** » puis sortir de Minicom.

L'utilisateur normal peut dès lors utiliser minicom :



Les commandes importantes sont :

CTRL A – Q : quitter

CTRL A – O : configuration

CTRLI A – S : envoyer un fichier

7.1.4- OUTILS DE TRANSFERT X/Y/Z MODEM

```
$ yum install lrzsz
.../...
Installed: lrzsz.i386 0:0.12.20-21
Complete!
```

7.1.5- INSTALLATION DE L'ENVIRONNEMENT

Décompresser l'archive CPUAT91_20.tar.bz2 à la racine du répertoire utilisateur :

```
$ mount /mnt/cdrom
$ cd ~/
$ tar xvjf /mnt/cdrom/CPUAT91_20.tar.bz2
```

POUR TOUTE LA SUITE NE PAS TRAVAILLER EN "ROOT" MAIS EN UTILISATEUR NORMAL.

8- MISE EN PLACE DES OUTILS DE DÉVELOPPEMENT

```
$ cd $HOME/CPUAT91_20
$ ./build_tools.sh
```

La construction des outils prend plusieurs dizaines de minutes.

Les outils générés par ce script sont :

arm-linux-addr2line	**arm-linux-g++**	**arm-linux-gdbtui**	**arm-linux-readelf**
arm-linux-ar	**arm-linux-gcc**	**arm-linux-ld**	**arm-linux-run**
arm-linux-as	**arm-linux-nm**	**arm-linux-size**	
arm-linux-c++	**arm-linux-gccbug**	**arm-linux-objcopy**	**arm-linux-strings**
arm-linux-c++filt	**arm-linux-gcov**	**arm-linux-objdump**	**arm-linux-strip**
arm-linux-cpp	**arm-linux-gdb**	**arm-linux-ranlib**	**mkfs.jffs2**
mkcramfs	**mkimage**		

8.1- Détail des outils générés

8.1.1- binutils

- **addr2line** : addr2line traduit les adresses d'un programme en nom de fichiers et numéros de ligne. A partir d'une adresse et d'un exécutable, il utilise les informations de déboguage présentes dans l'exécutable pour trouver quel nom de fichier et quel numéro de ligne sont associés à une adresse donnée.
- **ar** : Le programme ar crée, modifie et extrait des données depuis des archives. Une archive est un fichier contenant un ensemble d'autres fichiers dans une structure qui permet de retrouver les fichiers individuels d'origine (appelés membres de l'archive).
- **as** : as a d'abord été prévu pour assembler la sortie du compilateur GNU C, gcc, pour qu'elle puisse être utilisée par l'éditeur de liens ld.
- **ld** : ld combine plusieurs fichiers objets et archives, modifie l'emplacement de leurs données et lie les références des symboles. Souvent, la dernière étape dans la construction d'un nouveau programme compilé à exécuter est un appel à ld.
- **nm** : nm liste les symboles des fichiers objet.
- **objcopy** : L'utilitaire objcopy copie le contenu d'un fichier objet dans un autre. objcopy utilise la bibliothèque GNU BFD pour lire et écrire les fichiers objet. Il peut écrire le fichier objet destination dans un format différent de celui du fichier objet source.
- **objdump** : objdump affiche des informations sur un ou plusieurs fichiers objet. Les options déterminent quelles informations spécifiques sont à afficher. Ces informations sont essentiellement utiles pour les programmeurs qui travaillent sur des outils de compilation, à l'inverse des programmeurs qui veulent juste compiler leur programme pour qu'il fonctionne.
- **ranlib** : ranlib génère un index du contenu d'une archive, et le stocke dans l'archive. L'index liste chaque symbole défini par un membre d'une archive, qui est un fichier objet re-localisable.
- **readelf** : readelf affiche des informations sur les binaires de type elf.
- **size** : size liste les tailles des sections, et la taille totale, pour chaque fichier objet de sa liste d'arguments. Par défaut, une ligne est générée en sortie pour chaque fichier objet ou chaque module dans une archive.
- **strings** : Pour chaque fichier donné, strings affiche les séquences de caractères imprimables qui ont au moins une longueur de quatre caractères (ou le nombre spécifié en option du programme) et qui sont suivies d'un caractère non imprimable. Par défaut, il affiche seulement les chaînes des parties initialisées et chargées des fichiers objet. Pour les autres types de fichier, il affiche les chaînes de tout le fichier. strings est principalement utile pour déterminer le contenu des fichiers binaires.

- **strip** : strip débarrasse les fichiers objet de tous leurs symboles, ou de certains en particulier. La liste des fichiers objet peut inclure des archives. Au moins un fichier objet doit être fourni. strip modifie les fichiers fournis en argument, plutôt que d'écrire les copies modifiées sous d'autres noms.

Descriptif provenant de :
http://www.fr.linuxfromscratch.org/view/4.0-fr/chapter06/binutils.html

8.1.2- GLIBC

Librairies :

- **ld.so** : ld.so est le programme d'aide pour les exécutables utilisant des bibliothèques partagées.
- **libBrokenLocale, libBrokenLocale_p** : Utilisé par des logiciels, tels que Mozilla, pour résoudreles locales manquantes.
- **libSegFault** : libSegFault est un gestionnaire pour le signal 'segmentation fault'. Il essaie de capturer ces signaux.
- **libanl, libanl_p** : libanl est une bibliothèque de recherche de 'asynchronous name'.
- **libbsd-compat** : libbsd-compat procure une portabilité nécessaire pour lancer certains programmes sous Linux.
- **libc, libc_nonshared, libc_p** : Ces fichiers constituent la principale bibliothèque C. Celle-ci est une collection de fonctions couramment utilisées dans les programmes. Cela évite au développeur d'écrire ses propres fonctions pour toutes sortes de tâches. Les plus communes telles que l'affichage d'une chaîne de caractères à l'écran sont déjà présentes et à disposition du programmeur. La bibliothèque C (comme presque toutes les bibliothèques) est disponible en deux versions : dynamique et statique. En résumé, lorsqu'un programme utilise une bibliothèque C statique, le code de cette bibliothèque est copié dans l'exécutable. Lorsqu'un programme utilise une bibliothèque dynamique, cet exécutable ne contient pas le code de la bibliothèque, mais une routine qui charge la fonction de la bibliothèque au moment où le programme l'utilise. Cela réduit d'une façon significative la taille du programme. La documentation fournie avec la bibliothèque C décrit ce mécanisme plus en détails, il est trop compliqué de l'expliquer ici en une ou deux lignes.
- **libcrypt, libcrypt_p** : libcrypt est la bibliothèque de cryptographie.
- **libdl, libdl_p** : libdl est la bibliothèque d'interface pour le chargeur dynamique de liens.
- **libg** : libg est une bibliothèque de lancement pour g++.
- **libieee** : libieee est la bibliothèque des nombres flottants IEEE.
- **libm, libm_p** : libm est la bibliothèque de mathématiques.
- **libmcheck** : libmcheck contient du code exécuté au démarrage.
- **libmemusage** : libmemusage est utilisé par memusage pour aider à récupérer des informations sur l'utilisation de la mémoire par un programme.
- **libnsl, libnsl_p** : libnsl est la bibliothèque des services réseau.
- **libnss_compat, libnss_dns, libnss_files, libnss_hesiod, libnss_nis, libnss_nisplus** : L'idée principale est de mettre l'implémentation des différents services offerts pour accéder aux bases de données dans des modules séparés. Ceci a un certain nombre d'avantages: les contributeurs peuvent ajouter de nouveaux services sans les ajouter à la bibliothèque GNU C. les modules peuvent être mis à jour séparement. l'image de la bibliothèque C est plus petite.
- **libpcprofile** : Code utilisé par le noyau pour surveiller le temps CPU passé dans les fonctions, les lignes de codes sources et les instructions.
- **libpthread, libpthread_p** : La bibliothèques POSIX des threads.
- **libresolv, libresolv_p** : Les fonctions de cette bibliothèque permettent la création, l'envoi et l'interprétation des paquets des serveurs de noms Internet.
- **librt, librt_p** : Les fonctions de cette bibliothèque apportent la plupart des interfaces spécifiées par l'extension temps réel POSIX.1b.
- **libthread_db** : Les fonctions de cette bibliothèque sont utiles pour construire des debuggers pour les programmes multi-tâches.
- **libutil, libutil_p** : Contient du code pour les fonctions standards utilisé pour les différents utilitaires Unix.

Outils :

- **catchsegv** : catchsegv peut être utilisé pour créer une trace de la pile lorsqu'un programme s'interrompt avec une erreur 'segmentation fault'.
- **gencat** : gencat génère des catalogues de messages.
- **getconf** : getconf affiche les valeurs de configuration du système pour les variables spécifiques aux systèmes de fichiers.
- **getent** : getent récupère des entrées d'une base de données administrative.
- **glibcbug** : glibcbug crée un rapport de bug sur glibc et l'envoie par courrier électronique à l'adresse email pour les bugs.
- **iconv** : iconv réalise des conversions de jeux de caractères.
- **iconvconfig** : iconvconfig crée un fichier de configuration chargé rapidement pour le module iconv.
- **ldconfig** : ldconfig crée un cache des bibliothèques dynamiques pour l'éditeur de liens dynamiques.
- **ldd** : ldd affiche les bibliothèques partagées requis par chaque programme ou bibliothèque partagée spécifié sur la ligne de commande.
- **locale** : locale est un programme Perl indiquant au compilateur d'autoriser (ou non) l'utilisation des variables locales POSIX pour les opérations intégrées.
- **localedef** : localedef compile les spécifications pour locale.
- **mtrace** : mtrace affiche les chemins 'multicast' d'une source à un récepteur (une requête des traces IP).
- **nscd** : nscd est un démon procurant un cache pour les requêtes DNS les plus courantes.
- **nscd_nischeck** : nscd_nischeck vérifie si un mode de sécurité est nécessaire pour les recherches NIS+.
- **pcprofiledump** : pcprofiledump affiche l'information générée par 'PC profiling'.
- **pt_chown** : pt_chown indique l'utilisateur, le groupe et les permissions d'accès au terminal du pseudo terminal esclave correspondant au pseudo terminal maître passé sur le descripteur de fichier 3. C'est le programme d'aide pour la fonction 'grantpt'. Il n'a pas été conçu pour être lancé directement en ligne de commande.
- **rpcgen** : rpcgen génère le code C pour implémenter le protocole RPC.
- **rpcinfo** : rpcinfo fait un appel RPC vers un serveur RPC.
- **sln** : sln crée le lien symbolique entre une source et sa destination. C'est lié statiquement, sans nécessiter de lien dynamique. Donc, sln est utilisé pour créer des liens symboliques avec des bibliothèques dynamiques si, pour quelque raison que ce soit, le système de liens dynamiques n'était pas fonctionnel.
- **sprof** : sprof lit et affiche les données d'objets partagés profilés.
- **tzselect** : tzselect questionne l'utilisateur sur sa position géographique courante et affiche la description de fuseau horaire résultante sur la sortie standard.
- **xtrace** : xtrace trace l'exécution de programmes en affichant la fonction actuellement exécutée.
- **zdump** : zdump est le dumper du fuseau horaire.
- **zic** : zic est le compilateur de fuseau horaire.

Descriptif provenant de :
http://www.fr.linuxfromscratch.org/view/4.0-fr/chapter06/glibc.html

8.1.3- gcc

Librairies :

- **libgcc, libgcc_eh, libgcc_s** : Fichiers de support pour gcc.
- **libiberty** : libiberty est une collection de sous-routines utilisées par différents programmes GNU, comme getopt, obstack, strerror, strtol et strtoul.
- **libstdc++** : libstdc++ est la bibliothèque C++. Elle est utilisée par les programmes C++ et contient des fonctions couramment utilisées par eux. De cette façon, le développeur n'a pas besoin d'écrire certaines fonctions (comme écrire une ligne de texte à l'écran) à partir de rien, à chaque fois qu'il crée un programme.
- **libsupc++** : libsup++ procure une aide au langage de programmation c++. Entre autres

choses, libsup++ contient des routines pour la gestion des exceptions.

Outils :

- **cc, cc1, cc1plus, gcc** : Ce sont les compilateurs C. Un compilateur transforme le code source au format texte dans un format que l'ordinateur comprends. Après qu'un code source ait été compilé en un fichier objet, un éditeur de lien va créer un fichier exécutable à partir d'un ou plusieurs de ces fichiers générés par le compilateur.
- **c++, cc1plus, g++** : Ce sont les compilateurs C++, l'équivalent de cc et gcc, etc...
- **c++filt** : Le langage C++ procure la surcharge de fonctions, c'est-à-dire qu'il est possible d'écrire plusieurs fonctions possédant le même nom, à condition que chaque fonction utilise des paramètres de différents types. Tous les noms des fonctions C++ sont codés avec un label assembleur bas niveau (ce procédé est aussi connu sous le nom de mangling). Le programme c++filt réalise l'opération inverse; il décode les noms bas niveau en des noms de niveau utilisateur, de manière à ce que l'éditeur de liens ne confonde pas les fonctions surchargées.
- **collect2** : collect2 assiste à la compilation de constructeurs.
- **cpp, cpp0** : cpp réalise un travail préliminaire sur un fichier source, comme inclure le contenu des fichiers d'entêtes dans le code source. Il suffit d'insérer une ligne comme #include <filename> dans votre fichier source. Le préprocesseur va insérer le contenu de ce fichier dans le fichier source.
- **gccbug** : gccbug est un script shell, utilisé pour simplifier la création de rapports de bugs.
- **gcov** : gcov analyse les programmes pour aider à créer des codes plus efficaces, plus rapides avec une optimisation.

Descriptif provenant de :
http://www.fr.linuxfromscratch.org/view/4.0-fr/chapter06/gcc.html

8.1.4- GDB

- **gdb** : gdb est le debugger de la suite d'outils GNU.
- **gdbserver** : gdbserver est l'appendice de taille réduite qui une fois chargé et exécuté sur la cible permet de débugger une application par l'intermédiaire d'une interface série ou TCP/IP.
- **gdbreplay** : gdbreplay permet de "rejouer" à posteriori une séance de débug enregistrée en direct entre gdbserver & gdb.

9- GENERATION DES BINAIRES DU BOOTLOADER ET DU NOYAU

```
$ cd $HOME/CPUAT91_20
$ ./build_distrib.sh
```

Les binaires générés par ce script sont situés dans le dossier images :

- loader.bin : permet la programmation de la flash par le port série de débug
- u-boot_ram.bin : utilisé avec le loader ci-dessus lors de la première programmation de la flash
- u-boot_cpuat91.bin : bootloader programmé en flash
- uImage_cpuat91 : noyau programmé en flash
- rootfs_cpuat91.jffs2 : système de fichiers au format jffs2
- rootfs_cpuat91.cramfs : système de fichiers au format cramfs

Le script a aussi créé à la racine de CPUAT91_20 les répertoires suivants :

- build : contient tous les sources et objets, permet de recompiler manuellement un élément.
- log : contient les logs de l'exécution des scripts.
- rootfs : contient l'arborescence qui sera utilisée pour générer les images du système de fichiers.
- stages : contient des fichiers vides marquant la fin de chaque étape de construction.
- tools : contient les outils de compilation croisée.

u-boot dispose d'un outil permettant de générer des adresses MAC temporaires :

```
$ ./build/u-boot-1.2.0/tools/gen_eth_addr
0a:03:4a:c6:f3:a2
```

Nous utiliserons cette adresse par la suite.

Tous les fichiers contenus dans le dossier images doivent être copiés dans le répertoir eracine du serveur tftp :

```
$ cp images/* /tftpboot/
```

10- GÉNÉRATION DU SYSTÈME DE FICHIERS

10.1- PROCÉDURE STANDARD POUR REGÉNÉRER LE ROOTFS

```
$ cd $HOME/CPUAT91_20
$ ./rebuild_distrib.sh
.../...
Génération de l'image au format JFFS2
Génération de l'image au format CRAMFS
```

On obtient deux images binaires :

- l'une au format CRAMFS (compressé, lecture seule)
- l'autre au format JFFS2 (compressé, lecture / écriture, journalisé).

10.2- PERSONNALISATION

10.2.1- PERSONNALISATION DU SQUELETTE DU SYSTÈME DE FICHIERS

Cette personnalisation se fait à partir du squelette par défaut :

```
$ cd $HOME/CPUAT91_20/PKG_CPUAT91
$ mkdir tmp; cd tmp
$ tar xjf ../rootfstree-cpuat91.tar.bz2
$ ls
bin  dev  etc  lib  mnt  proc  root  sbin  sys  tmp  usr  var
```

Les fichiers importants à personnaliser sont :

- **etc/network/interfaces :** paramètres réseau

```
# Configure Loopback
auto lo eth0 usb0
iface lo inet loopback

#iface eth0 inet static
#       address 192.168.1.96
#       netmask 255.255.255.0
#       gateway 192.168.1.254

iface eth0 inet dhcp
        pre-up ifconfig eth0 hw ether 0a:03:4a:c6:f3:a2

iface usb0 inet static
        address 10.0.0.2
        netmask 255.255.255.0
        gateway 10.0.0.1
```

- etc/hostname : nom du système

```
cpuat91
```

- etc/hosts : correspondance statique de noms d'hôtes

```
127.0.0.1       cpuat91 localhost
```

- etc/resolv.conf : paramètres DNS (si non attribués par DHCP)

```
domain dev.null
nameserver 127.0.0.1
```

- etc/issue : message présenté au login

```
CPUAT91 - v2.0 - 2007-06
http://www.eukrea.com/ - contact@eukrea.com
```

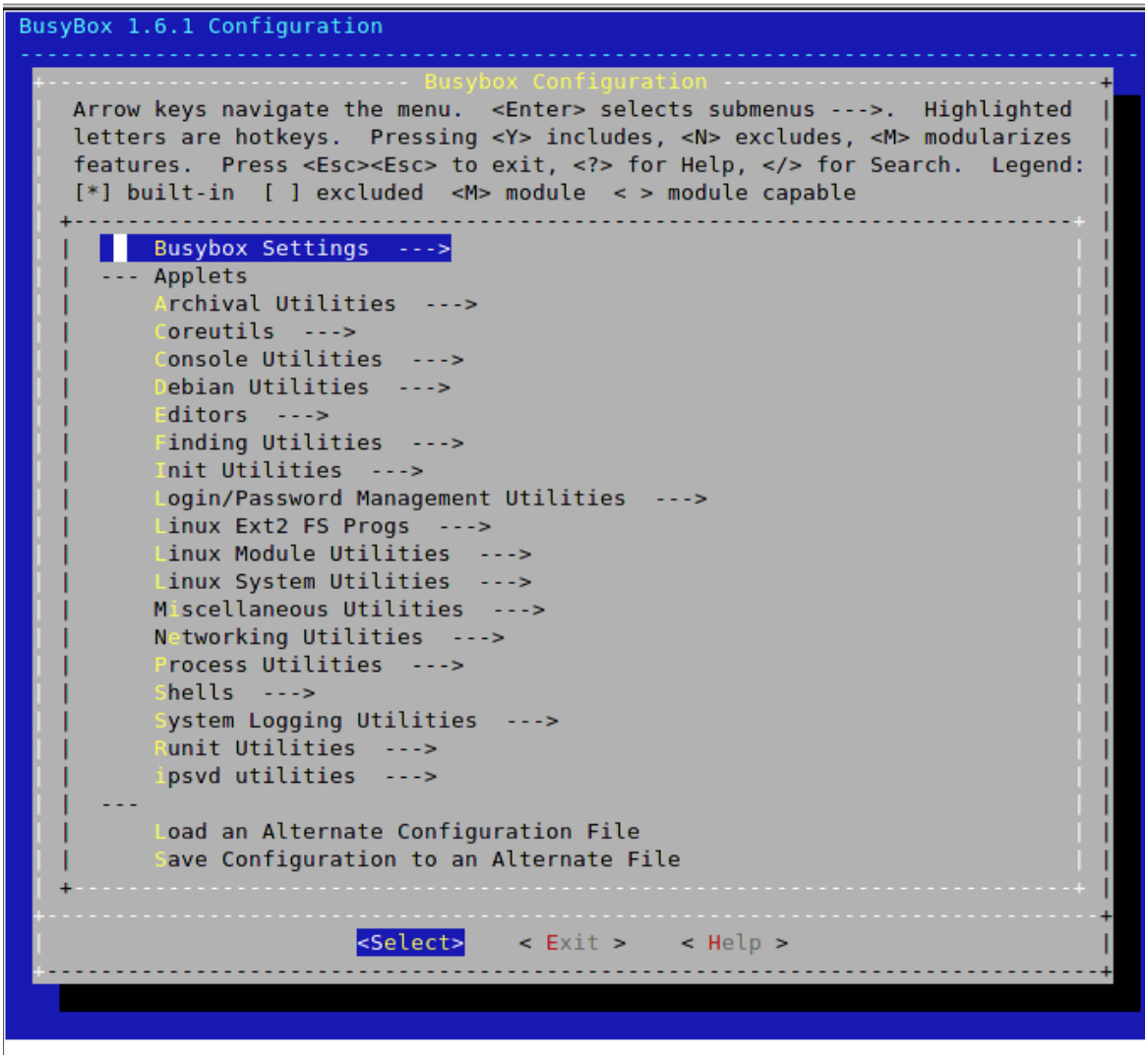
Une fois ces fichiers modifiés, il est nécessaire de regénérer l'archive de l'arborescence de base du rootfs :

```
$ tar cvjf ../rootfstree-cpuat91.tar.bz2 .
```

A la prochaine exécution du script build_rootfs.sh, cette arborescence sera utilisée pour construire le nouvelle système de fichiers.

10.2.2- PERSONNALISATION DE BUSYBOX

```
$ cd $HOME/CPUAT91_20/build/busybox-1.6.1
$ make menuconfig
```



Une fois busybox configuré, quitter en sauvegardant la configuration, puis copier le fichier de configuration généré dans le répertoire $HOME/CPUAT91/confs :

```
$ cp .config ../../PKG_CPUAT91/busybox_dotconf-1.6.1
```

A la prochaine exécution du script rebuild_distrib.sh, cette nouvelle configuration sera utilisée.

10.2.3- INSTALLATION D'APPLICATIONS COMPLÉMENTAIRES

Copier les binaires des applications dans l'arborescence située sous $HOME/CPUAT91_20/rootfs/

Puis générer les images binaires de la manière suivante :

```
$ ./build_images.sh
Construction des images jffs2
Flash 128 ko / secteur
Construction de l'image cramfs
Les binaires sont dans le dossier images !
```

A terme, il est recommandé d'enrichir les scripts fournis pour intégrer la construction des application rajoutées au fur et à mesure de votre développement.

11- PREMIERE PROGRAMMATION DU MODULE

Connecter le port série de DEBUG (DB9 couvert d'une pastille rouge) au PC au moyen d'un câble série croisé.

Installer le jumper JP11 (Jumper rouge).

Veiller à ce que les jumpers JP9 et JP10 soient installés (Jumpers bleus) afin que l'UART de Débug fonctionne.

```
$ cd $HOME/CPUAT91_20
$ ./flash_cpuat91.sh /dev/nom_du_port_série
```

exemples : /dev/ttyS0 : 1er port série physique du PC

/dev/tts/USB0 : 1er port série sur bus USB connecté au PC.

```
$ ./flash_cpuat91.sh /dev/ttyS0
Merci de fermer les applications utilisant le port série : /dev/ttyS0
Presser enter pour démarrer la procédure
ENTER
```

si une autre application utilise le port série, le script propose de la terminer

```
/dev/ttyS0:
Le processus  32204 utilise le port série /dev/ttyS0
Voulez vous tuer cette application ? (O/n)
ENTER
/dev/ttyS0:       32204
Relier le port série de débug au PC par un cable croisé
Placer le jumper JP11 sur le KIT AT91 et appuyer sur le bouton reset
puis presser enter pour continuer
ENTER
sx-at91 started...
Serial port : /dev/ttyS0
C
waiting for ACK,1,133,...Ok!
.../...
Sending file complete
Vous pouvez maintenant lancer un terminal série sur : /dev/ttyS0
```

démarrer le termine série de votre choix (par la suite, nous utiliseront ckermit) :

```
$ ckermit
Connecting to /dev/ttyS0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------

CPUAT91=> setenv ethaddr 0a:03:4a:c6:f3:a2
CPUAT91=> setenv serverip 192.168.1.5
CPUAT91=> setenv ipaddr 192.168.1.96
CPUAT91=> saveenv
Saving Environment to Flash...
.
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
.
```

```
Protected 1 sectors
CPUAT91=> tftp 20000000 u-boot_cpuat91.bin
TFTP from server 192.168.1.5; our IP address is 192.168.1.96
Filename 'u-boot_cpuat91.bin'.
Load address: 0x20000000
Loading: T #########################
done
Bytes transferred = 127560 (1f248 hex)
CPUAT91=> protect off 10000000 1001FFFF
.
Un-Protected 1 sectors
CPUAT91=> erase 10000000 1001FFFF
. done
Erased 1 sectors
CPUAT91=> cp.b 20000000 10000000 ${filesize}
Copy to Flash... done
CPUAT91=> protect on 10000000 1001FFFF
.
Protected 1 sectors
CPUAT91=> tftp 20000000 uImage_cpuat91
TFTP from server 192.168.1.5 our IP address is 192.168.1.96
Filename 'uImage_cpuat91'.
Load address: 0x20000000
Loading: ###################################################################
         ###################################################################
         ###################################################################
         ###############################################
done
Bytes transferred = 1228092 (12bd3c hex)
CPUAT91=> protect off 10040000 1019FFFF
...........
Un-Protected 11 sectors
CPUAT91=> erase 10040000 1019FFFF
........... done
Erased 11 sectors
CPUAT91=> cp.b 20000000 10040000 ${filesize}
Copy to Flash... done
CPUAT91=> tftp 20000000 rootfs_cpuat91.jffs2
TFTP from server 192.168.1.5; our IP address is 192.168.1.96
Filename 'rootfs_cpuat91.jffs2'.
Load address: 0x20000000
Loading: ###################################################################
         ###################################################################
         ###################################################################
         ###################################################################
         ###################################################################
         ###################################################################
         ###################################################################
         #############
done
Bytes transferred = 2393788 (2486bc hex)
CPUAT91=> protect off 101A0000 107FFFFF
...................................................
Un-Protected 51 sectors
CPUAT91=> erase 101A0000 107FFFFF
................................................... done
Erased 51 sectors
```

```
CPUAT91=> cp.b 20000000 101A0000 $(filesize)
Copy to Flash... done
```

Dans le cas où le système de fichiers installé est JFFS2 :

```
CPUAT91=> setenv bootargs root=/dev/mtdblock2 rootfstype=jffs2
console=ttyS0,115200 mtdparts=physmap-flash.0:256k(u-boot),1408k(kernel),-
(rootfs) mem=32M
```

Dans le cas où le système de fichiers installé est CRAMFS :

```
CPUAT91=> setenv bootargs root=/dev/mtdblock2 rootfstype=cramfs
console=ttyS0,115200 mtdparts=physmap-flash.0:256k(u-boot),1408k(kernel),-
(rootfs) mem=32M
CPUAT91=> setenv bootcmd bootm 10040000
CPUAT91=> setenv bootdelay 1
CPUAT91=> saveenv
```

Retirer le jumper JP11 (Jumper rouge)

Générer un reset en pressant le bouton.

12- PREMIER DEMARRAGE

```
U-Boot 1.2.0 (Jul 11 2007 - 23:07:29)

DRAM:  32 MB
Flash:  8 MB
In:    serial
Out:   serial
Err:   serial
Press SPACE to abort autoboot in 1 seconds
## Booting image at 10040000 ...
   Image Name:   Linux-2.6.22.1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1396660 Bytes =  1.3 MB
   Load Address: 20008000
   Entry Point:  20008000
   Verifying Checksum ... OK
OK

Starting kernel ...

Uncompressing
Linux.....................................................................
..................... done, booting the kernel.
Linux version 2.6.22.1 (ebenard@eric) (gcc version 4.1.2) #1 PREEMPT Wed
Jul 11 20:44:28 CEST 2007
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
Machine: EUKREA AT91RM9200 SBC
Memory policy: ECC disabled, Data cache writeback
Clocks: CPU 179 MHz, master 59 MHz, main 18.432 MHz
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists.  Total pages: 8128
Kernel command line: root=/dev/mtdblock2 rootfstype=jffs2
console=ttyS0,115200 mtdparts=physmap-flash.0:256k(u-boot),1408k(kernel),-
(rootfs) mem=32M
```

```
AT91: 96 gpio irqs in 3 banks
PID hash table entries: 128 (order: 7, 512 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory: 32MB = 32MB total
Memory: 29480KB available (2612K code, 235K data, 116K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
NET: Registered protocol family 16
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 1024 (order: 1, 8192 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
JFFS2 version 2.2. (NAND) Â© 2001-2006 Red Hat, Inc.
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
AT91 Watchdog Timer enabled (5 seconds, nowayout)
atmel_usart.0: ttyS0 at MMIO 0xfefff200 (irq = 1) is a ATMEL_SERIAL
atmel_usart.1: ttyS1 at MMIO 0xfffc0000 (irq = 6) is a ATMEL_SERIAL
atmel_usart.2: ttyS2 at MMIO 0xfffc4000 (irq = 7) is a ATMEL_SERIAL
atmel_usart.3: ttyS3 at MMIO 0xfffc8000 (irq = 8) is a ATMEL_SERIAL
atmel_usart.4: ttyS4 at MMIO 0xfffcc000 (irq = 9) is a ATMEL_SERIAL
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: module loaded
nbd: registered device at major 43
at91_ether: Your bootloader did not configure a MAC address.
eth0: Link now 100-FullDuplex
eth0: AT91 ethernet at 0xfefbc000 int=24 100-FullDuplex
(00:00:00:00:00:00)
eth0: Micrel KS8721 PHY
physmap platform flash device: 01000000 at 10000000
physmap-flash.0: Found 1 x16 devices at 0x0 in 16-bit bank
 Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
cfi_cmdset_0001: Erase suspend on write enabled
3 cmdlinepart partitions found on MTD device physmap-flash.0
Creating 3 MTD partitions on "physmap-flash.0":
0x00000000-0x00040000 : "u-boot"
0x00040000-0x001a0000 : "kernel"
0x001a0000-0x00800000 : "rootfs"
Generic platform RAM MTD, (c) 2004 Simtec Electronics
mtd-ram mtd-ram.0: registered mtd device
atmel_spi atmel_spi.0: Atmel SPI Controller at 0xfffe0000 (irq 13)
at91_ohci at91_ohci: AT91 OHCI
at91_ohci at91_ohci: new USB bus registered, assigned bus number 1
at91_ohci at91_ohci: irq 23, io mem 0x00300000
usb usb1: Product: AT91 OHCI
usb usb1: Manufacturer: Linux 2.6.22.1 ohci_hcd
usb usb1: SerialNumber: at91
```

```
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
udc: at91_udc version 3 May 2006
ether gadget: using random self ethernet address
ether gadget: using random host ethernet address
usb0: Ethernet Gadget, version: May Day 2005
usb0: using at91_udc, OUT ep2 IN ep1 STATUS ep4
usb0: MAC ae:22:ae:2e:a7:22
usb0: HOST MAC 1e:fa:49:94:38:83
usb0: RNDIS ready
mice: PS/2 mouse device common for all mice
at91_rtc at91_rtc: rtc core: registered at91_rtc as rtc0
AT91 Real Time Clock driver.
i2c /dev entries driver
at91_i2c at91_i2c: AT91 i2c bus driver.
AT91 MMC: 4 wire bus mode not supported - using 1 wire
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
at91_rtc at91_rtc: setting the system clock to 1998-01-01 00:00:06
(883612806)
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 116K
mmc0: host does not support reading read-only switch.  Assuming write-
enable.
mmcblk0: mmc0:cde9 SD04G 3932160KiB
 mmcblk0: p1
kjournald starting.  Commit interval 5 seconds
EXT3-fs warning: maximal mount count reached, running e2fsck is
recommended
EXT3 FS on mmcblk0p1, internal journal
EXT3-fs: recovery complete.
EXT3-fs: mounted filesystem with ordered data mode.
eth0: Setting MAC address to 4e:a0:7f:c0:7d:72
var/
var/lib/
var/log/
var/run/
var/tmp/
var/lock/
Starting network...
eth0: Setting MAC address to 0a:03:4a:c6:f3:a2
udhcpc (v1.6.1) started
eth0: Link now 100-FullDuplex
Sending discover...
Sending select for 192.168.1.10...
Lease of 192.168.1.10 obtained, lease time 864000
deleting routers
route: SIOC[ADD|DEL]RT: No such process
adding dns 212.27.54.252
adding dns 212.27.53.252
Will output 768 bit dss secret key to
'/etc/dropbear/dropbear_dss_host_key'
```

```
Generating key, this may take a while...
Public key portion is:
ssh-dss
AAAAB3NzaC1kc3MAAABhAIrSCoWjXkpJD2ppJeiOKYgCYtlguT2xfiKuwUplSutkvRqYH21GQe
Gqwc3rVzb9pi7PUJsfTFCvHtyhjMOL9fy9HcSO8y0gBG83A5VVAms/m8KXXZjf8e9+WbcikJ0C
TQAAABUAu26P/Tgmb8qNTbH3zn5QOp4PTLEAAABgGBRGKumljE0HgqT4W7LXfFFQQogv9NM1Qk
HrS0YzkPrnFPULr76JKHdFM06ciacVzdihw4GiTKCLW65qnhIMgUbPPtAWnUnUtoNzx0UA6lTM
DUBXuuBX+8ZYOaq9AzufAAAAYEOJIoWX7ITeLr0uA9XqMks6uG57/bHwnzY0wcT5QjGrmbiKMT
+Xq2oTVjr9/m467cFiTuGZN+sN/IHIFiEtV3ZOHyMoblFVb6ktmPvJ5gcqLYEn56b7NSzc2aNw
/9eJDw== root@
Fingerprint: md5 ba:4e:1d:40:75:f5:3b:a1:a0:18:60:99:52:0f:29:a6
Will output 768 bit rsa secret key to
'/etc/dropbear/dropbear_rsa_host_key'
Generating key, this may take a while...
Public key portion is:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAAAYwCVrWQlXKrdimxcTv0vqMAtKrs5tB9du6OlE4bxA8Yk3H
+5CAA7lpy0917jUD4bVxOBu53pwX+JW4EuvdEgb3tpPcdABZtYrd8Xx2AuOkUBHhLOXcOHQcCh
ATM1LgTFz7qgxw== root@
Fingerprint: md5 e3:32:31:44:a5:40:b4:f6:16:a4:f1:39:b4:76:d7:83
Starting dropbear sshd: OK
Starting telnet server: OK

CPUAT91 - v2.0 - 2007-06
http://www.eukrea.com/ - contact@eukrea.com

 login:     root
Password: eukrea
login[850]: root login on 'ttyS0'
#
```

Attention : au premier démarrage, le noyau «formate» l'espace disponible de la flash, le temps de réponse au login est donc rallongé. Il est inutile de resetter la carte, cette opération ne sera plus renouvelée ultérieurement.

12.1- APPLICATIONS DISPONIBLES

```
[               fakeidentd      makedevs        sha1sum
[[              false           md5sum          sleep
addgroup        fbset           mdev            softlimit
adduser         fdflush         mesg            sort
adjtimex        fdformat        mkdir           split
ar              fdisk           mkdosfs         start-stop-daemon
arp             fgrep           mkfifo          stat
arping          find            mkfs.minix      strace
ash             fold            mknod           strings
awk             free            mkswap          stty
basename        freeramdisk     mktemp          su
bbconfig        fsck            modprobe        sulogin
bunzip2         fsck.minix      more            sum
busybox         ftpget          mount           sv
bzcat           ftpput          mountpoint      svlogd
cal             fuser           mt              swapoff
cat             gdbserver       mv              swapon
catv            getopt          nameif          switch_root
```

```
chattr          getty           nc              sync
chgrp           grep            netstat         sysctl
chmod           gunzip          nice            syslogd
chown           gzip            nmeter          tail
chpst           halt            nohup           tar
chroot          hdparm          nslookup        taskset
chrt            head            od              tcpsvd
chvt            hexdump         openvt          tee
cksum           hostid          passwd          telnet
clear           hostname        patch           telnetd
cmp             httpd           pidof           test
comm            hwclock         ping            tftp
cp              id              ping6           time
cpio            ifconfig        pipe_progress   top
crond           ifdown          pivot_root      touch
crontab         ifup            poweroff        tr
cryptpw         inetd           printenv        traceroute
cut             init            printf          true
date            insmod          ps              tty
dc              install         pwd             udhcpc
dd              ip              rdate           udhcpd
deallocvt       ipaddr          readahead       udpsvd
delgroup        ipcalc          readlink        umount
deluser         ipcrm           readprofile     uname
devmem2         ipcs            realpath        uncompress
df              iplink          reboot          uniq
dhcprelay       iproute         renice          unix2dos
diff            iprule          reset           unlzma
dirname         iptunnel        resize          unzip
dmesg           kill            rm              uptime
dnsd            killall         rmdir           usleep
dos2unix        killall5        rmmod           uudecode
dosfsck         klogd           route           uuencode
dpkg            last            rpm             vconfig
dpkg-deb        length          rpm2cpio        vi
dropbear        less            run-parts       vlock
dropbearkey     linux32         runlevel        watch
du              linux64         runsv           watchdog
dumpkmap        ln              runsvdir        wc
dumpleases      loadfont        rx              wget
echo            loadkmap        scp             which
ed              logger          sed             who
eeprog          login           seq             whoami
egrep           logname         setarch         xargs
eject           logread         setconsole      yes
env             losetup         setkeycodes     zcat
envdir          ls              setlogcons      zcip
envuidgid       lsattr          setsid
ether-wake      lsmod           setuidgid
expr            lzmacat         sh
```

12.2- CONNEXION À LA CARTE PAR LE RÉSEAU

L'adresse IP de la carte est, par défaut attribuée par un serveur dhcp.

Cette adresse peut être modifiée de la manière suivante :

```
# ifconfig eth0 nouvelle_ip
```

Un serveur telnet est présent, ce qui permet de se connecter à la carte par l'intermédiaire du réseau.

```
$ telnet 192.168.1.10
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.

 login: root
Password: eukrea
#
```

Un serveur ssh est aussi présent sur le port 2222 :

```
$ ssh root@192.168.1.10 -p 2222
The authenticity of host '[192.168.1.10]:2222 ([192.168.1.10]:2222)' can't
be established.
RSA key fingerprint is e3:32:31:44:a5:40:b4:f6:16:a4:f1:39:b4:76:d7:83.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.1.10]:2222' (RSA) to the list of
known hosts.
root@192.168.1.10's password: eukrea
#
```

12.3- Utilisation d'une carte MMC :

L'insertion d'une carte MMC provoque le log suivant :

```
mmcblk0: mmc0:cde9 SD04G 3932160KiB
 mmcblk0: p1
```

La carte est montée automatiquement au boot si elle est présente.

Sinon, elle peut être montée de la manière suivante :

```
# mount /mnt/mmc
# df /mnt/mmc
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/mmc/mmc0/part1    3869415    178977   3493837   5% /mnt/mmc
# umount /mnt/mmc
```

Attention : le système de fichier FAT utilisé par défaut sur les cartes MMC n'étant pas robuste, il est fortement conseillé de toujours démonter la carte avant de l'éjecter ou d'utiliser un autre système de fichiers.

12.4- Utilisation d'une clef USB

L'insertion d'une clef USB provoque le log suivant :

```
usb 1-1: new full speed USB device using at91_ohci and address 2
usb 1-1: Product: DataTraveler II+
usb 1-1: Manufacturer: Kingston
usb 1-1: SerialNumber: 5B51050206B4
usb 1-1: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
scsi 0:0:0:0: Direct-Access     Kingston DataTraveler II+ 1.13 PQ: 0 ANSI:
0 CCS
sd 0:0:0:0: [sda] 1006592 512-byte hardware sectors (515 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
```

```
sd 0:0:0:0: [sda] 1006592 512-byte hardware sectors (515 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
sd 0:0:0:0: Attached scsi generic sg0 type 0
```

La clef peut être montée de la manière suivante :

```
# mount /mnt/usb/
# df /mnt/usb/
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda1               502648    145600    357048  29% /mnt/usb
# umount /mnt/usb/
```

12.5- CONNEXION USB À UN PC WINDOWS

Relier le port USB périphérique de la carte à un port USB hôte d'un PC fonctionnant sous Windows XP (ou Linux).

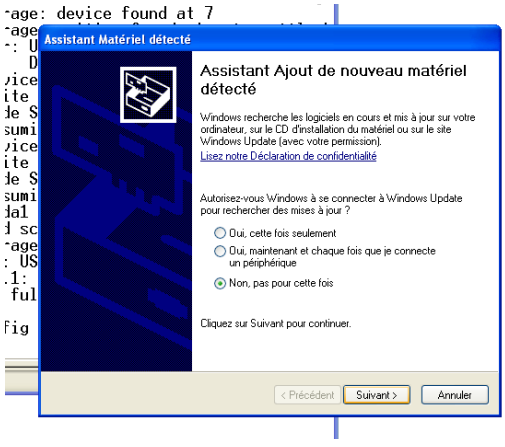
La carte détecte la connexion :

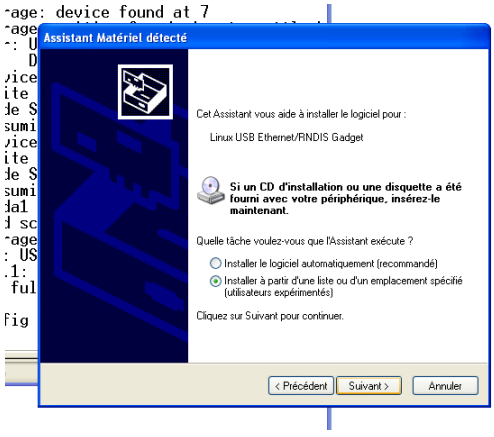
```
usb0: full speed config #1: 100 mA, Ethernet Gadget, using CDC Ethernet
```

L'adresse IP de l'interface USB de la carte est définie par défaut à 10.0.0.2 :

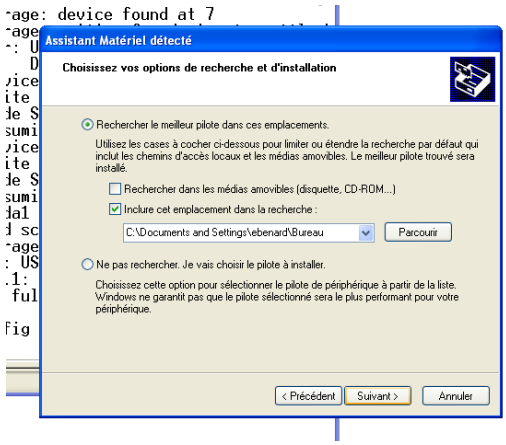
```
# ifconfig usb0
usb0      Link encap:Ethernet  HWaddr AE:22:AE:2E:A7:22
          inet addr:10.0.0.2  Bcast:10.0.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:56 (56.0 B)  TX bytes:0 (0.0 B)
```

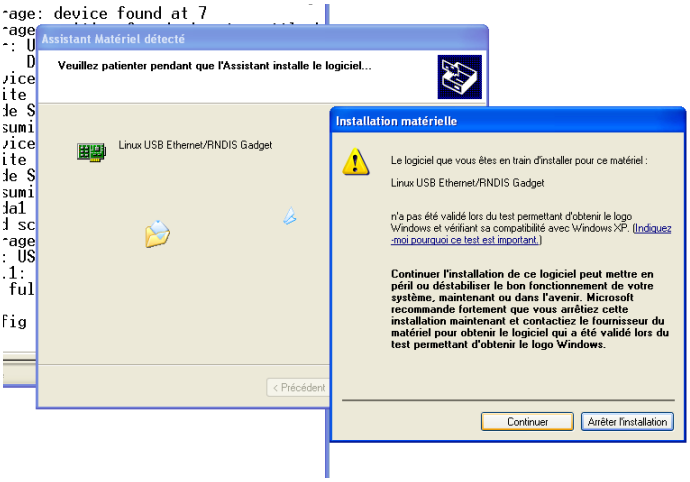
Le PC détecte un nouveau périphérique :

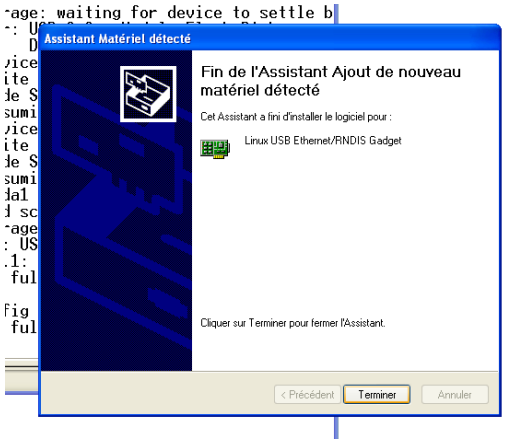




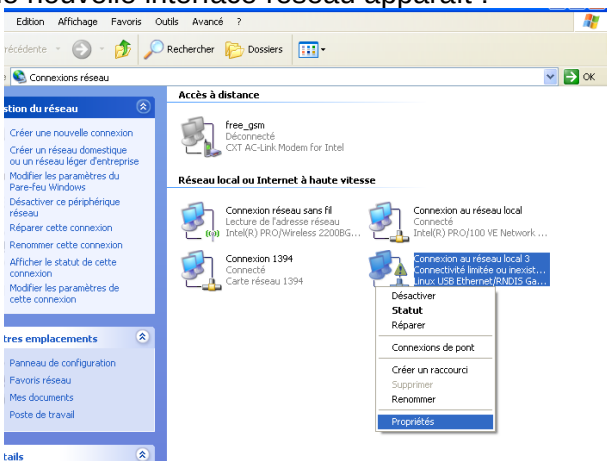
Fournir le chemin chez le répertoire contenant le fichier linux.inf (présent sur le CD) :



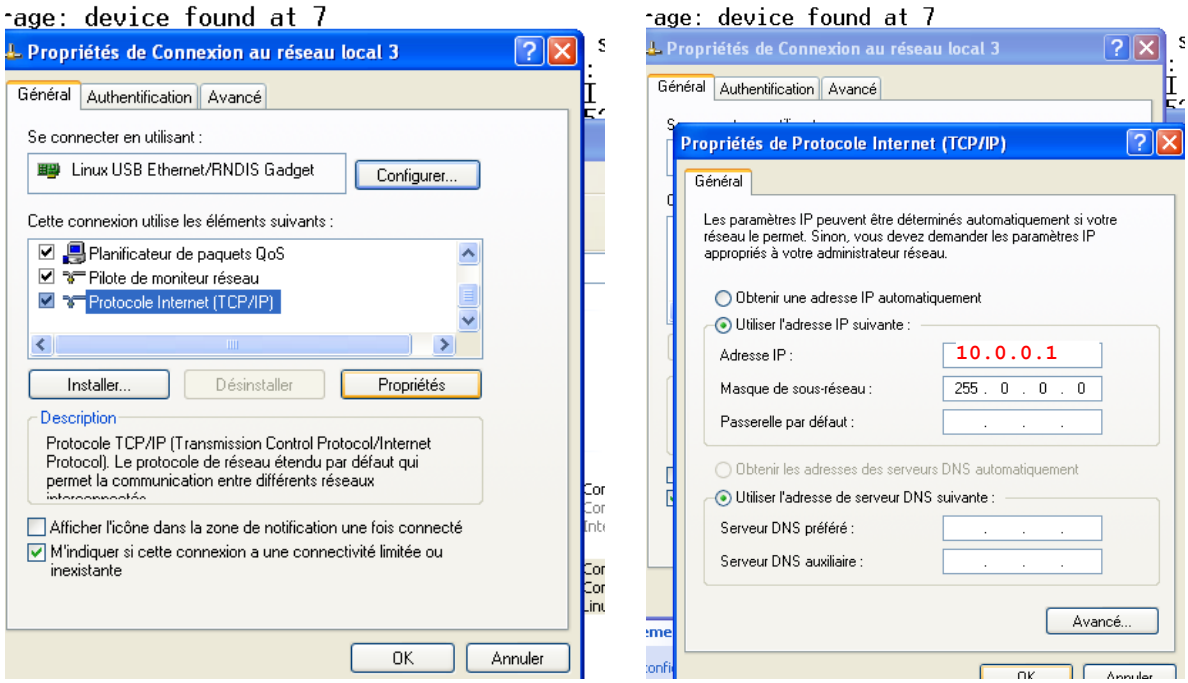




Une fois la driver installé, une nouvelle interface réseau apparaît :



Il est donc nécessaire de configurer son adresse IP :



Une fois les deux interfaces réseau paramétrées (coté carte et coté PC), il est possible d'effectuer du trafic IP entre les deux :

depuis la carte :

```
# ping -c 10 10.0.0.1
```

depuis le PC (dans une fenêtre DOS) :

```
# ping -n 10 10.0.0.2
```

12.6- TEST EEPROM I2C

L'outil eeprog permet de lire et d'écrire dans une eeprom I2C de type 24x :

12.6.1- ECRITURE

```
# echo " TEST I2C SUR EEPROM 24LC08 EUKREA" | eeprog -f /dev/i2c-0 0x54 -w
0:33
eeprog 0.7.6, a 24Cxx EEPROM reader/writer
Copyright (c) 2003-2004 by Stefano Barbato - All rights reserved.
  Bus: /dev/i2c-0, Address: 0x54, Mode: 8bit
  Writing stdin starting at address 0x0
.................................
```

12.6.2- LECTURE

```
# eeprog -f /dev/i2c-0 0x54 -r 0:34
eeprog 0.7.6, a 24Cxx EEPROM reader/writer
Copyright (c) 2003-2004 by Stefano Barbato - All rights reserved.
  Bus: /dev/i2c-0, Address: 0x54, Mode: 8bit
  Reading 34 bytes from 0x0
 TEST I2C SUR EEPROM 24LC08 EUKREA#
```

12.7- TEST GPIO

L'outil gpio permet de lire et modifier l'état des GPIO :

```
# gpio
AT91RM9200 user-space GPIO control utility
(C) Copyright 2004 Roman Avramenko <roman@imsystems.ru>
Usage: gpio token1 token2 ... tokenN
Where token is: [action][port][pin],
and action is: [+] set or [-] clear or [?] check; port is: [PA,PB,PC,PD];
pin is: [0..31]
Note: supported multiple set/clear and only one check request per each
call
Examples: gpio +PB2 -PB3 [set PB2, clear PB3]
          gpio ?PB2 (return 0 if pin PB2 in 0 state, or 1 overwise)
```

12.8- TEST UARTS

L'outil test_rs permet de tester la connectivité entre deux ports séries.

Exemples :

relier UART0 et UART1 au moyen d'un câble série croisé.

```
# test_rs /dev/ttyS1 /dev/ttyS2
Serial 1 : /dev/ttyS1 - Serial 2 : /dev/ttyS2
Test /dev/ttyS1 => /dev/ttyS2 @ speed : 9600 with 41 chars OK
Test /dev/ttyS2 => /dev/ttyS1 @ speed : 9600 with 41 chars OK
Test /dev/ttyS1 => /dev/ttyS2 @ speed : 19200 with 41 chars OK
Test /dev/ttyS2 => /dev/ttyS1 @ speed : 19200 with 41 chars OK
Test /dev/ttyS1 => /dev/ttyS2 @ speed : 38400 with 41 chars OK
Test /dev/ttyS2 => /dev/ttyS1 @ speed : 38400 with 41 chars OK
Test /dev/ttyS1 => /dev/ttyS2 @ speed : 57600 with 41 chars OK
Test /dev/ttyS2 => /dev/ttyS1 @ speed : 57600 with 41 chars OK
Test /dev/ttyS1 => /dev/ttyS2 @ speed : 115200 with 41 chars OK
Test /dev/ttyS2 => /dev/ttyS1 @ speed : 115200 with 41 chars OK
```

relier UART2 et UART3 au moyen d'un câble série croisé.

```
# test_rs /dev/ttyS3 /dev/ttyS4
Serial 1 : /dev/ttyS3 - Serial 2 : /dev/ttyS4
Test /dev/ttyS3 => /dev/ttyS4 @ speed : 9600 with 41 chars OK
Test /dev/ttyS4 => /dev/ttyS3 @ speed : 9600 with 41 chars OK
Test /dev/ttyS3 => /dev/ttyS4 @ speed : 19200 with 41 chars OK
Test /dev/ttyS4 => /dev/ttyS3 @ speed : 19200 with 41 chars OK
Test /dev/ttyS3 => /dev/ttyS4 @ speed : 38400 with 41 chars OK
Test /dev/ttyS4 => /dev/ttyS3 @ speed : 38400 with 41 chars OK
Test /dev/ttyS3 => /dev/ttyS4 @ speed : 57600 with 41 chars OK
Test /dev/ttyS4 => /dev/ttyS3 @ speed : 57600 with 41 chars OK
Test /dev/ttyS3 => /dev/ttyS4 @ speed : 115200 with 41 chars OK
Test /dev/ttyS4 => /dev/ttyS3 @ speed : 115200 with 41 chars OK
```

13- COMPILATION ET DÉBUG D'UN PROGRAMME EN C ET C++

13.1- GÉNÉRALITÉS

Le débug d'un programme s'effectue en exécutant ce programme sur la carte au travers de gdbserver et en contrôlant l'exécution depuis le logiciel arm-linux-gdb du PC de développement.
La communication entre les deux s'effectue par réseau ou par port série. Dans ce dernier cas, il est nécessaire qu'aucun autre logiciel ne fasse appel au port série.

Commandes clefs de GDB :

- **target remote** IP:PORT ou /dev/ttyS0 : sélectionne la cible
- **c** : continue
- **s** : step
- **print** x : affiche le contenu de la variable x
- **l** : liste le code en cours d'exécution
- **i b** : affiche les points d'arrêt positionnés

Les exemples suivants sont dans le répertoire $HOME/CPUAT91_20/exemples

```
$ cd $HOME/CPUAT91_20
$ ./setup_env.sh
Configuration de l'environnement pour la compilation croisée
Vous êtes à présent dans l'environnement de cross-compilation
Tapper CTRL+D pour restaurer l'environnement initial
$ cd exemples
```

L'adresse IP de la carte est 192.168.1.10, l'adresse IP du PC de développement est 192.168.1.5.

13.2- PROGRAMME EN C

Soit le programme de test suivant (test.c) :

```
#include <stdio.h>

main() {
  unsigned int i,j;

  printf("Hello World !\n");
  i = 3;
  j = 12;
  i = i + j;
  printf("i = %i - j = %i\n", i, j);
}
```

Compiler ce programme :

```
$ make
rm -f test test_g
arm-linux-gcc -mcpu=arm920t -mtune=arm920t -msoft-float -O0 -g test.c -o
test
cp test test_g
arm-linux-strip test
cp test /tftpboot
```

Nous choisissons volontairement un niveau d'optimisation nul afin que le compilateur conserve le calcul et les variables i et j.
Deux binaires sont générés : test qui est dépourvu des symboles et test_g qui possède les symboles :

```
-rwxrwxr-x  1 ebenard ebenard 3152 jui  7 16:52 test
```

```
-rwxrwxr-x  1 ebenard ebenard 7974 jui  7 16:52 test_g
```

- Côté Carte :

Nous opérerons dans le répertoire /tmp situé en SDRAM :

```
# cd /tmp
```

Charger le programme de test dans la carte (il est possible de charger une version ne comportant pas de symboles, réduite par arm-linux-strip).

```
# tftp -g -r test 192.168.1.5
```

Rendre test exécutable

```
# chmod 755 test
```

Exécuter test au travers de gdbserver

```
# gdbserver 192.168.1.5:2345 ./test
Process test created; pid = 734
Listening on port 2345
```

L'adresse IP doit correspondre à celle du PC de développement, le port doit être un port disponible.

- Côté PC :

```
$ arm-linux-gdb test_g
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-
linux"...
(gdb)  target remote 192.168.1.10:2345
Remote debugging using 192.168.1.10:2345
0x40000bd0 in ?? ()
(gdb) l
1       #include <stdio.h>
2
3       main() {
4               unsigned int i,j;
5
6               printf("Hello World !\n");
7               i = 3;
8               j = 12;
9               i = i + j;
10              printf("i = %i - j = %i\n", i, j);
(gdb) b 10
Breakpoint 1 at 0x84c8: file test.c, line 10.
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:10
10              printf("i = %i - j = %i\n", i, j);
(gdb) print i
$1 = 15
```

```
(gdb) print j
$2 = 12
(gdb) c
Continuing.

Program exited normally.
(gdb)
```

- Côté carte :

```
Remote debugging from host 192.168.1.5
Hello World !
i = 15 - j = 12

Child exited with retcode = 0

Child exited with status 0
GDBserver exiting
```

13.3- Programme en C++

Soit le programme de test suivant (test2.cpp) :

```
#include <iostream>
using std::cout;

int main() {
  cout << "Hello World!";
  return 0;
}
```

Compiler ce programme :

```
$ make
rm -f test2 test2_g
arm-linux-g++ -mcpu=arm920t -mtune=arm920t -msoft-float -O0 -g test2.cpp
-o test2cp test2 test2_g
arm-linux-strip test2
cp test2 /tftpboot
```

- Côté Carte :

Nous opérerons dans le répertoire /tmp situé en SDRAM :

```
# cd /tmp
```

Charger le programme de test dans la carte (il est possible de charger une version ne comportant pas de symboles, réduite par arm-linux-strip).

```
# tftp -g -r test2 192.168.1.5
```

Rendre test exécutable

```
# chmod 755 test2
```

Exécuter test au travers de gdbserver

```
# gdbserver 192.168.1.5:2345 ./test2
Process test created; pid = 734
Listening on port 2345
```

L'adresse IP doit correspondre à celle du PC de développement, le port doit être un port disponible.

- Côté PC :

```
$ arm-linux-gdb test2_g
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-
linux"...
(gdb) target remote 192.168.1.10:2345
Remote debugging using 192.168.1.10:2345
0x40000bd0 in ?? ()
(gdb) b main
Breakpoint 1 at 0x885c: file test2.cpp, line 5.
(gdb) c
Continuing.

Breakpoint 1, main () at test2.cpp:5
5         cout << "Hello World!";
(gdb) l
1       #include <iostream>
2       using std::cout;
3
4       int main() {
5         cout << "Hello World!";
6         return 0;
7       }
(gdb) b 6
Breakpoint 2 at 0x8868: file test2.cpp, line 6.
(gdb) c
Continuing.

Breakpoint 2, main () at test2.cpp:6
6         return 0;
(gdb) c
Continuing.

Program exited normally.
```

- Côté carte :

```
Remote debugging from host 192.168.1.5
Hello World!
Child exited with retcode = 0

Child exited with status 0
GDBserver exiting
```

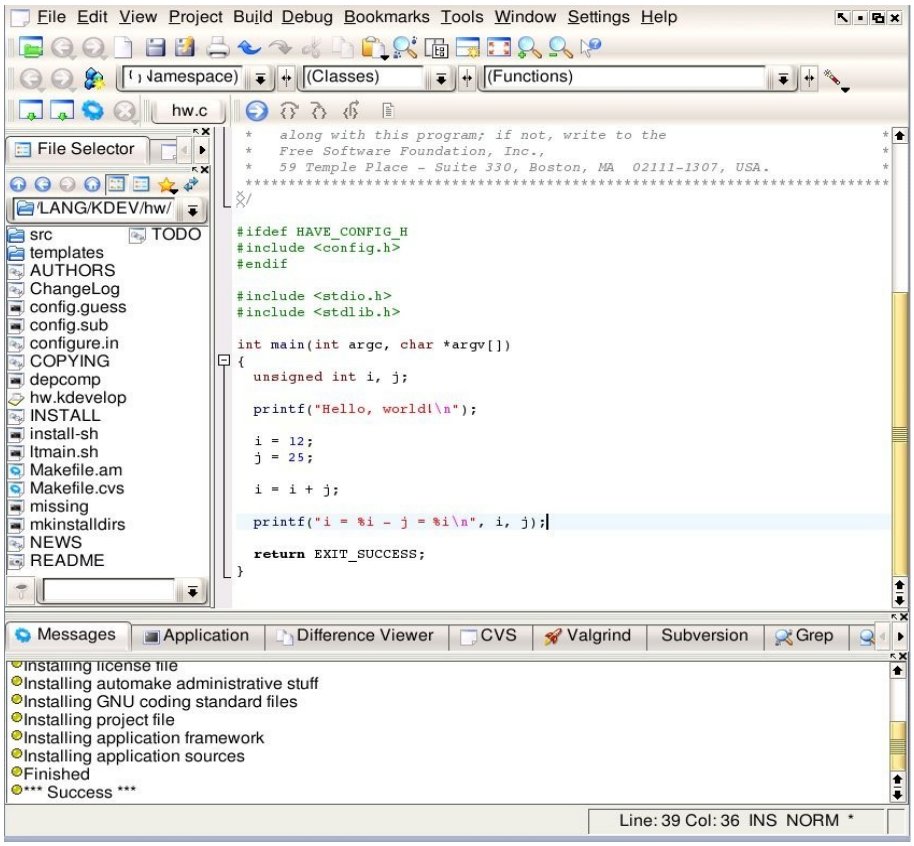
13.4- Edition / débug avec l'interface graphique Kdevelop3

Attention : il est important de démarrer kdevelop depuis le terminal dans lequel ./setup_env.sh

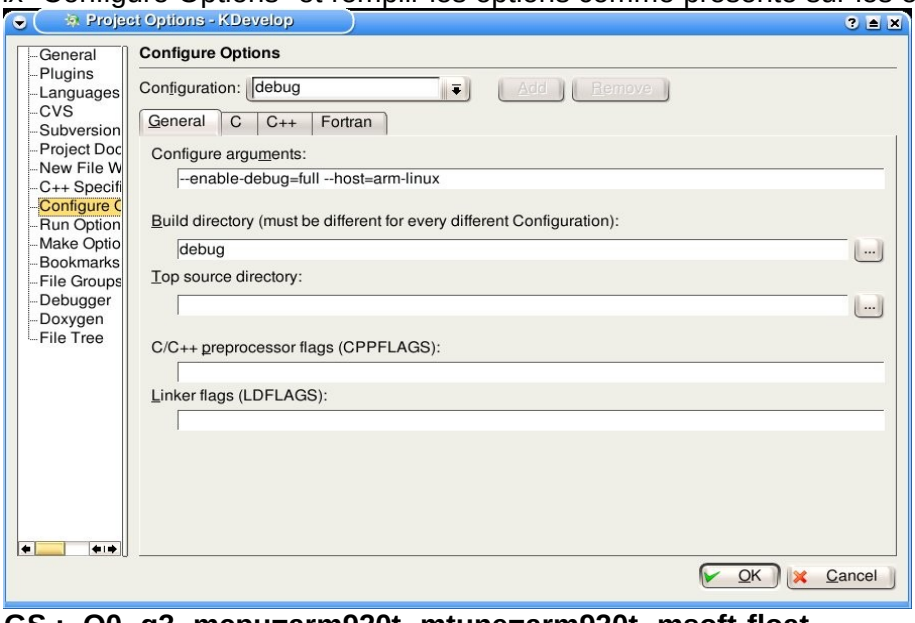
a été exécuté, sinon les outils ne seront pas visibles par kdevelop.

Menu : Project > New Project
Menu : C > Simple Hello World program

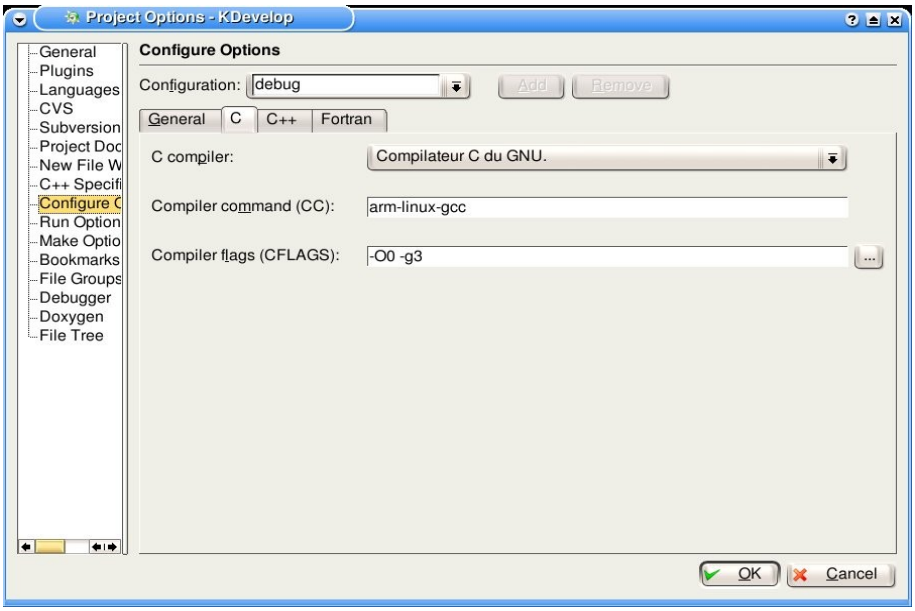
Saisir le programme dans la fenêtre d'édition comme sur l'écran suivant :



Menu : Project > Project Options
Aller dans le choix "Configure Options" et remplir les options comme présenté sur les écrans suivants :

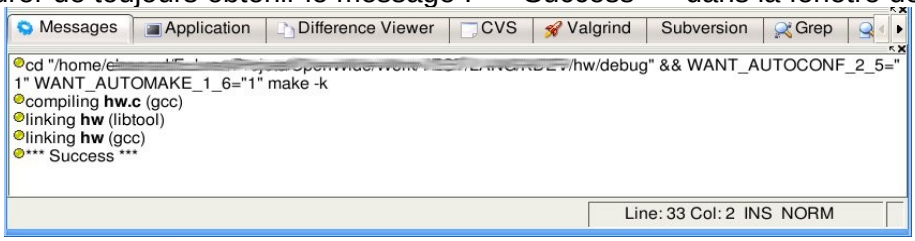


Attention : **CFLAGS : -O0 -g3 -mcpu=arm920t -mtune=arm920t -msoft-float**



Menu : Build > Run automake & friends
Menu : Build > Run configure
Menu : Build > Build project

Attention : s'assurer de toujours obtenir le message : *** Success *** dans la fenêtre de messages :



Le programme a donc été compilé avec succès. Il est disponible dans le dossier debug/src situé à la racine du dossier du projet ("debug" correspondant à la configuration sélectionnée dans l'onglet "Configure Options").
A noter : il est possible et intéressant de créer diverses configurations : certaines compilant pour x86 et permettant donc une exécution sur le PC, d'autres compilant pour ARM et permettant donc une exécution sur la carte. Chaque configuration sera compilée dans son propre dossier, distinct du code source du projet.

Il est maintenant possible de charger ce programme (après l'avoir réduit par arm-linux-strip) sur la carte et de l'exécuter au travers de gdbserver :

- Sur le PC :

```
$ cp debug/src/h< /tftpboot/hw
$ arm-linux-strip /tftpboot/hw
```

- Sur la Carte :

Nous opérerons dans le répertoire /tmp situé en SDRAM :

```
# cd /tmp
```

Charger le programme de test dans la carte (il est possible de charger une version ne comportant pas de symboles, réduite par arm-linux-strip).

```
# tftp -g -r hw 192.168.1.5
```

Rendre test exécutable

```
# chmod 755 hw
```

Exécuter test au travers de gdbserver

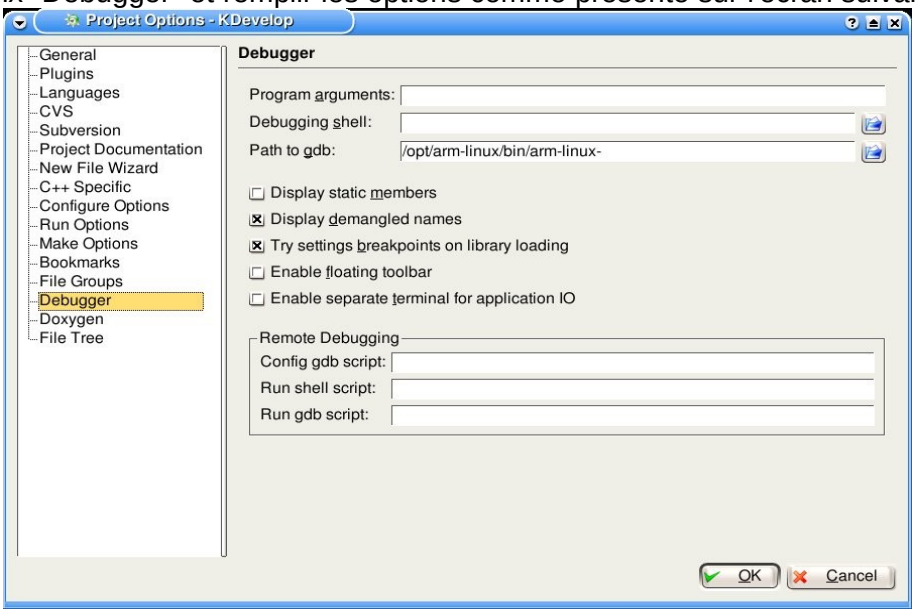
```
# gdbserver 192.168.1.5:2345 ./hw
Process hw created; pid = 77
```

L'adresse IP doit correspondre à l'IP du PC de développement, le port doit être un port disponible.

Le programme est maintenant prêt à être débuggé et attend les instruction d'une instance de arm-linux-gdb sur le PC de développement.

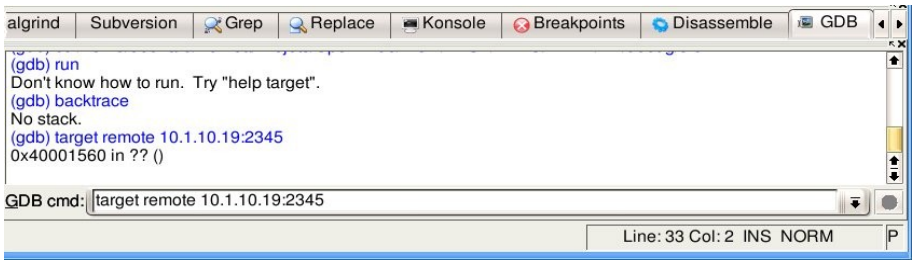
Menu : Project > Project Options

Aller dans le choix "Debugger" et remplir les options comme présenté sur l'écran suivant :

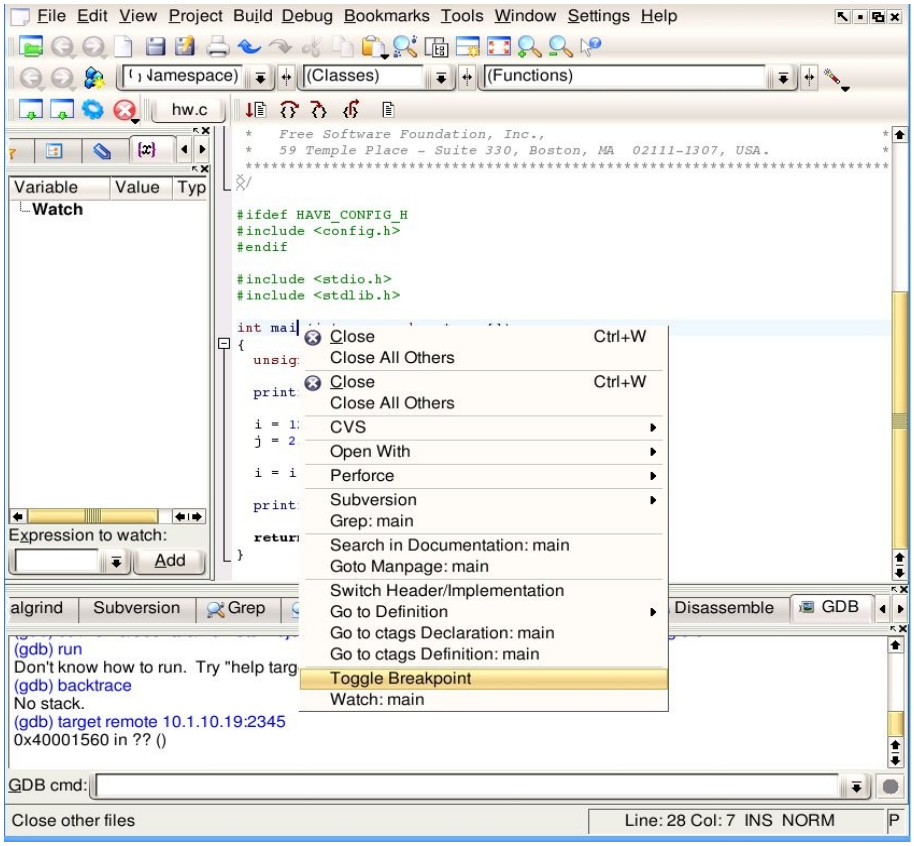


Menu : Debug > Start

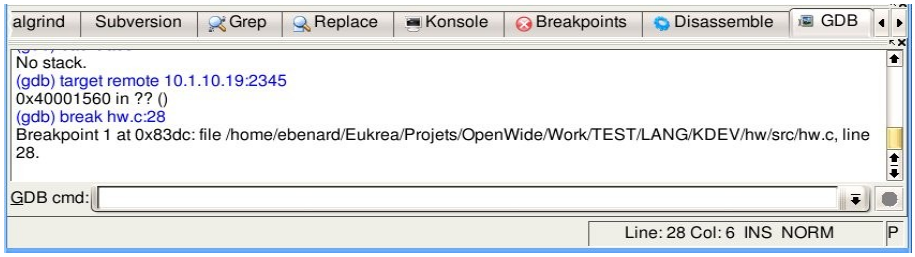
L'interface de GDB apparaît dans un onglet en bas de la fenêtre de kdevelop3.
saisir : **target remote 192.168.1.10:2345** dans le dialogue GDB cmd :



Il est possible de placer un point d'arrêt dans le code grâce au menu qui apparaît en pressant le bouton droit de la souris :

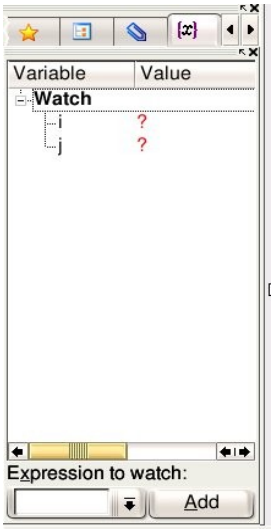


L'onglet GDB confirme la prise en compte du point d'arrêt :

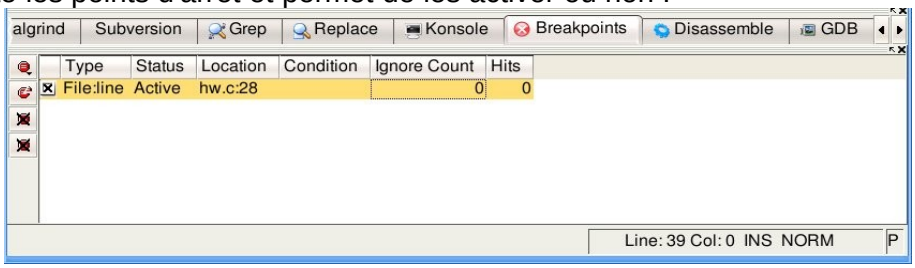


De la même manière, il est possible de surveiller des variables (par exemple : clic droit sur "i" et Watch:i).

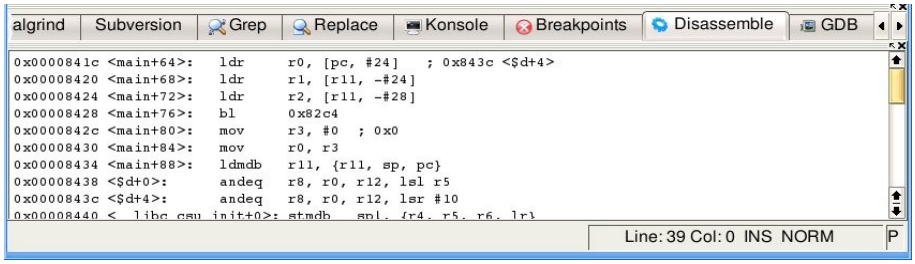
Les variables à surveiller apparaissent alors dans l'onglet adéquat :



A noter les onglets suivant qui présentent un intérêt :
Breakpoints : liste les points d'arrêt et permet de les activer ou non :

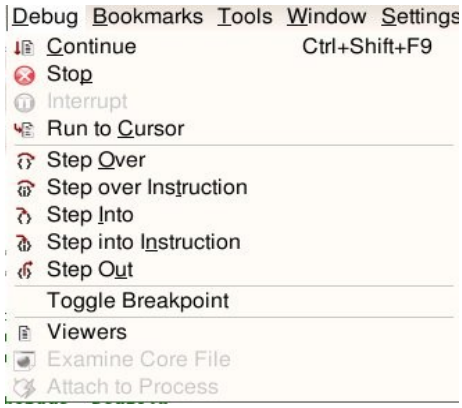


Disassemble : permet de désassembler le code en cours d'exécution :



Deux solutions s'offrent à nous pour contrôler l'exécution du programme :

Le menu Debug

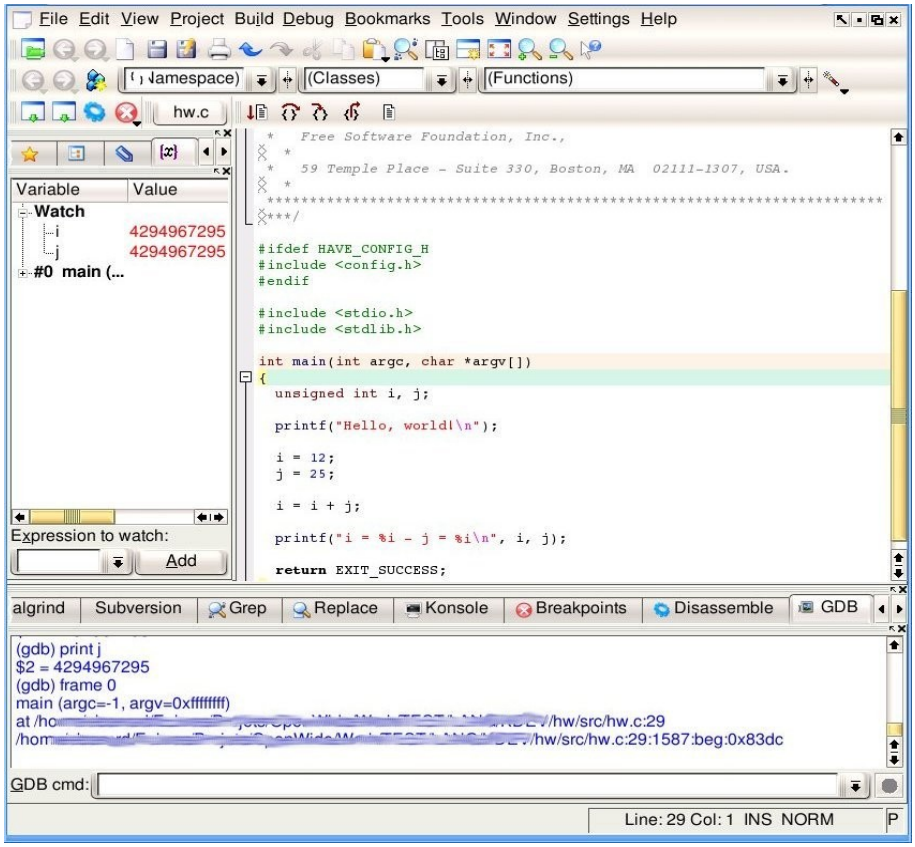


La palette de boutons Debug

Continue
Step Over
Step Into
Step Out
Viewers

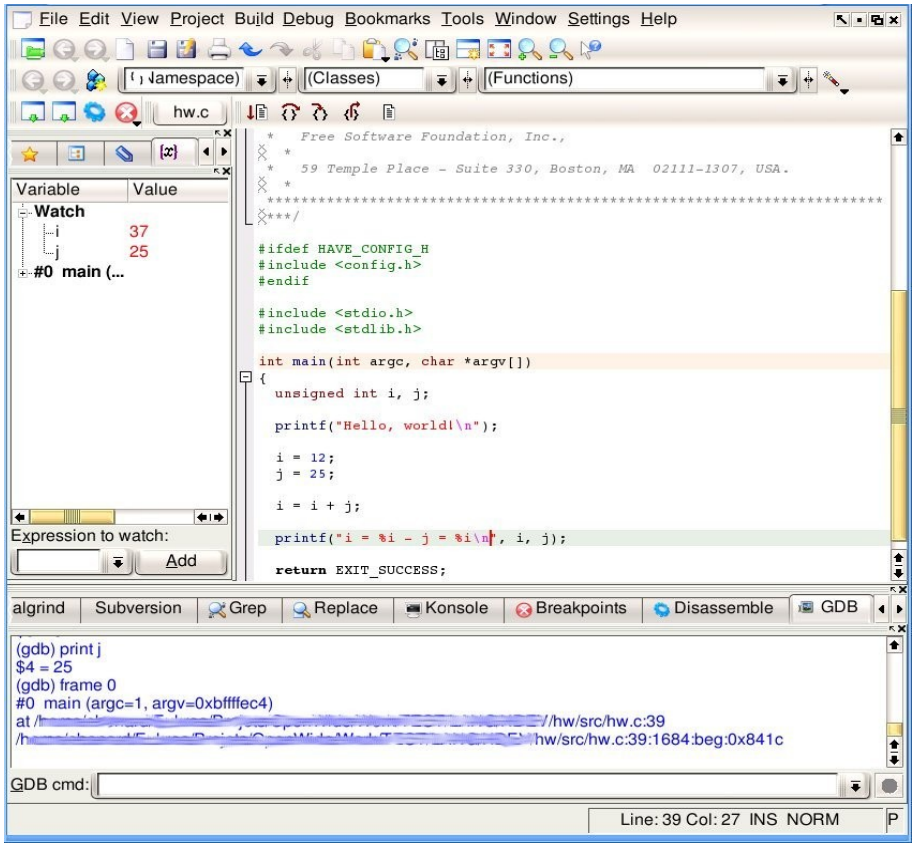
Pour lancer l'exécution du programme il suffit de faire "Continue".

Le programme s'exécute alors jusqu'au premier point d'arrêt :



La ligne de en cours d'exécution est surlignée en vert.
La ligne comportant un point d'arrêt est surlignée en orangé.

Plaçons un point d'arrêt sur la ligne comportant le second "printf" et relançons l'exécution ("Continue").



Nous observons que les variables ont été mises à jour dans l'onglet de gauche.

Un nouvel appel à "Continue" permet de terminer l'exécution du programme.

- Sur la Carte :

nous obtenons le log suivant

```
# ./gdbserver 192.168.1.5:2345 ./hw
Process hw created; pid = 83
Remote debugging from host 192.168.1.5
Hello, world!
i = 37 - j = 25

Child exited wih retcode = 0

Child exited with status 0
GDBserver exiting
```